Efficient Privacy-Preserving
Configurationless Service Discovery
Supporting Multi-Link Networks

Doctoral thesis for obtaining the
academic degree
Doctor of Natural Sciences (Dr. rer. nat.)

submitted by
Daniel Kaiser

at the
Universität
Konstanz

Faculty of Sciences
Department of Computer and Information Science

Konstanz, 2017
Date of the oral examination: August 16, 2017

1. Reviewer: Prof. Dr. Marcel Waldvogel

2. Reviewer: Prof. Dr. Oliver Haase

3. Reviewer: Prof. Dr. Bela Gipp
This work is licensed under a Creative Commons Attribution 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/.
If the only prayer you ever say in your entire life is thank you, it will be enough.

Meister Eckhart

To prevent social engineering attacks — after all, this thesis’ main focus is privacy — I will refrain from publishing a list of people whom I am grateful to as they already know.¹

¹A person who is in the top ranks of my gratefulness list pointed the following article, which ironically discusses overly long dedications in dissertations and aligns well with my opinion, out to me: http://www.spiegel.de/lebenundlernen/uni/skurrile-widmungen-in-doktorarbeiten-mein-dank-gilt-meinem-hund-a-752956.html, retrieved on 2/2/2017.
Abstract

Data is the pollution problem of the information age, and protecting privacy is the environmental challenge — this quotation from Bruce Schneier laconically illustrates the importance of protecting privacy. Protecting privacy — as well as protecting our planet — is fundamental for humankind. Privacy is a basic human right, stated in the 12th article of the United Nations’ Universal Declaration of Human Rights. The necessity to protect human rights is unquestionable.

Nothing ever threatened privacy on a scale comparable to today’s interconnected computers. Ranging from small sensors over smart phones and notebooks to large compute clusters, they collect, generate and evaluate vast amounts of data. Often, this data is distributed via the network, not only rendering it accessible to addressees, but also — if not properly secured — to malevolent parties. Like a toxic gas, this data billows through networks and suffocates privacy.

This thesis takes on the challenge of protecting privacy in the area of configurationless service discovery. Configurationless service discovery is a basis for user-friendly applications. It brings great benefits, allowing the configurationless network setup for various kinds of applications; e.g. for communicating, sharing documents and collaborating, or using infrastructure devices like printers.

However, while today’s various protocols provide some means of privacy protection, typical configurationless service discovery solutions do not even consider privacy. As configurationless service discovery solutions are ubiquitous and run on almost every smart device, their privacy problems affect almost everyone.

The quotation aligns very well with configurationless service discovery. Typically, configurationless service discovery solutions realize configurationlessness by using cleartext multicast messages literally polluting the local network and suffocating privacy. Messages containing private cleartext data are sent to everyone, even if they are only relevant for a few users. The typical means for mitigating the network pollution problem caused by multicast per se, regardless of the privacy aspects, is confining multicast messages to a single network link or to the access network of a WiFi access point; institutions often even completely deactivate multicast. While this mitigates the privacy problem, it also strongly scales configurationless service discovery down, either confining it or rendering it completely unusable.

In this thesis, we provide an efficient configurationless service discovery framework that protects the users’ privacy. It further reduces the network pollution by reducing the number of necessary multicast messages and offers a mode of operation that is completely independent of multicast. Introducing a multicast independent mode of operation, we also address the problem of the limited range in which services are discoverable. Our framework comprises components for device pairing, privacy-preserving service discovery, and multi-link scaling. These components are independent and — while usable in a completely separated way — are meant to be used as an integrated framework as they work seamlessly together.

Based on our device pairing and privacy-preserving service discovery components, we published IETF Internet drafts specifying a privacy extension for DNS service discovery over multicast DNS, a wildly used protocol stack for configurationless service discovery. As our drafts have already been adopted by the dnssd working group, they are likely to become standards.
Kurzfassung

Data is the pollution problem of the information age, and protecting privacy is the environmental challenge — dieses Zitat von Bruce Schneier zeigt auf lakonische Weise wie wichtig es ist die Privatsphäre zu schützen. Der Schutz der Privatsphäre ist — wie das Schützen unseres Planeten — fundamental für die Menschheit. Privatsphäre ist ein grundlegendes Menschenrecht, welches im 12. Artikel der Menschenrechtserklärung der Vereinten Nationen festgelegt ist. Die Notwendigkeit die Privatsphäre zu schützen ist unstrittig.

Nichts hat bis jetzt in dem Maß die Privatsphäre bedroht wie durch Netzwerke miteinander verbundene Computer. Diese Computer, die alles von kleinen Sensoren über Smartphones und Notebooks bis zu großen Rechenclustern umfassen, sammeln, generieren und evaluieren riesige Datenmengen. Diese Daten werden oft über das Netzwerk verteilt, was sie nicht nur für Adressaten zugänglich macht, sondern auch, falls sie nicht richtig gesichert sind, für böswillige Dritte. Diese Daten wabern wie giftiges Gas durch die Netzwerke und ersticken die Privatsphäre.


Während jedoch viele der heutigen Protokolle Mittel zum Schutz der Privatsphäre bieten, beachten typische Lösungen für konfigurationslose Service-Discovery das Thema Privatsphäre nicht einmal. Da Lösungen für konfigurationslose Service-Discovery allgegenwärtig sind und auf fast jedem smarten Gerät laufen, ist beinahe jeder von deren Privatsphäreproblemen betroffen.

Das Zitat passt sehr gut zu konfigurationsloser Service-Discovery. Lösungen für konfigurationslose Service-Discovery realisieren die Konfigurationslosigkeit typischerweise durch Verwendung von klartextbasierten Multicastnachrichten, welche im wahrsten Sinne des Wortes das lokale Netzwerk verschmutzen und die Privatsphäre ersticken. Nachrichten, welche private Klartextdaten enthalten, werden an jeden gesandt, auch wenn sie lediglich für eine geringe Nutzerzahl relevant sind. Ein Mittel, was typischerweise eingesetzt wird um das durch Multicast hervorgerufene Netzwerkverschmutzungsproblem zu mildern, ist die Ausbreitung von Multicastnachrichten auf einen einzelnen Netzwerk-Link oder das Access-Network eines WLAN-Accesspoints einzuschränken. Institutionen deaktivieren Multicast oftmals sogar vollständig. Das mildert zwar das Privatsphäreproblem, schränkt aber auch stark die Skalierbarkeit konfigurationsloser Service-Discovery ein oder macht sie sogar unbenutzbar.

— dafür gedacht als zusammenhängendes Framework benutzt zu werden, da sie reibungslos zusammen arbeiten.

## Contents

1 Introduction .................................................................................................................. 1
   1.1 Motivation .................................................................................................................. 2
      1.1.1 Privacy: Leaking a Lot of Information ................................................................. 2
      1.1.2 Locality: Limiting Connectivity and Collaboration .............................................. 3
   1.2 Research Questions ................................................................................................... 4
      1.2.1 Device Pairing ..................................................................................................... 6
      1.2.2 Directory Discovery ............................................................................................ 6
      1.2.3 Service Exploration ............................................................................................ 8
      1.2.4 Unavoidable User Interactions .......................................................................... 8
   1.3 Contributions ............................................................................................................ 9
      1.3.1 Publications ......................................................................................................... 10
      1.3.2 Software ............................................................................................................ 11
   1.4 Structure of this Thesis ............................................................................................ 13

2 Service Discovery Overview .......................................................................................... 15
   2.1 Introduction ................................................................................................................ 16
   2.2 Related Work ............................................................................................................ 17
   2.3 Components of Service Discovery Systems ............................................................. 21
      2.3.1 Service ................................................................................................................ 21
      2.3.2 Client .................................................................................................................. 23
      2.3.3 Service Provider ................................................................................................. 23
      2.3.4 Service Directory ............................................................................................... 23
   2.4 Interaction of Service Discovery Components ......................................................... 25
      2.4.1 Service Registration ............................................................................................ 25
      2.4.2 Directory Discovery ............................................................................................ 26
      2.4.3 Service Exploration ............................................................................................ 29
   2.5 Configurationless Service Discovery ........................................................................ 32
      2.5.1 Zeroconf and DNS-SD/mDNS ............................................................................ 32
      2.5.2 UPnP and SSDP .................................................................................................. 34
      2.5.3 Comparing Zeroconf and UPnP ......................................................................... 39
      2.5.4 Bluetooth and Bluetooth SDP ............................................................................. 40
      2.5.5 Privacy Considerations ....................................................................................... 42
      2.5.6 Multi-Link Considerations .................................................................................. 43
      2.5.7 Interoperability and Bridging ............................................................................. 46
   2.6 Conclusion and Prospects ......................................................................................... 46
## Contents

3 DNS Service Discovery 49
   3.1 Introduction ............................................. 50
   3.2 Related Work ........................................... 51
   3.3 DNS .................................................... 51
      3.3.1 Domain Names ..................................... 52
      3.3.2 Resource Records .................................. 54
      3.3.3 The DNS Protocol ................................ 55
      3.3.4 The Hierarchical DNS Architecture ............... 56
      3.3.5 EDNS .............................................. 59
      3.3.6 DNS Service Discovery (DNS-SD) .................. 60
      3.3.7 DNS Security ...................................... 62
   3.4 Multicast DNS (mDNS) .................................... 64
      3.4.1 Distinction from DNS ............................... 65
      3.4.2 Providing Resource Records ....................... 66
      3.4.3 Query Messages ................................... 67
      3.4.4 Answer Messages .................................. 68
      3.4.5 Mechanisms for Reducing the Network Load ...... 68
   3.5 DNS Service Discovery over Multicast DNS ............... 69
      3.5.1 Query Process and Privacy Considerations ... 69
      3.5.2 Multi-Link Considerations ....................... 72
      3.5.3 Network Load Considerations .................... 73
   3.6 Conclusion and Prospects ............................... 73

4 A Secure, Scalable, and User-Friendly SD Framework 75
   4.1 Introduction ............................................. 76
   4.2 Related Work ........................................... 77
   4.3 Abstract Concepts ..................................... 77
      4.3.1 Pairing ........................................... 78
      4.3.2 Privacy-Preserving Service Discovery ............ 79
      4.3.3 Multi-Link and Scopes ........................... 81
   4.4 Integration and Architecture ........................... 82
      4.4.1 Client Interaction ................................ 83
      4.4.2 Components of the Framework .................... 84
   4.5 User Interface ........................................... 84
      4.5.1 Main Views ....................................... 85
      4.5.2 Menu ............................................. 87
      4.5.3 Pairing .......................................... 88
      4.5.4 Privacy and Scope Settings for Service Discovery 91
   4.6 Implementation .......................................... 92
   4.7 Network Impact ......................................... 92
   4.8 Conclusion and Future Work ............................ 94

5 Device Pairing 97
   5.1 Introduction ............................................. 99
   5.2 Related Work ........................................... 101
      5.2.1 Surveys .......................................... 101
      5.2.2 Usability Studies ................................. 102
      5.2.3 Secure Pairing Protocols using PKI or Passwords 102
      5.2.4 Secure Manual Pairing Protocols ................ 103
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.2.5</td>
<td>Formal Analysis</td>
<td>105</td>
</tr>
<tr>
<td>5.2.6</td>
<td>Authentication Channels</td>
<td>105</td>
</tr>
<tr>
<td>5.2.7</td>
<td>Secure In-Band Pairing</td>
<td>105</td>
</tr>
<tr>
<td>5.2.8</td>
<td>Practical Applications</td>
<td>106</td>
</tr>
<tr>
<td>5.3</td>
<td>Device Pairing Background</td>
<td>108</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Diffie-Hellman key exchange</td>
<td>109</td>
</tr>
<tr>
<td>5.3.2</td>
<td>The Man in the Middle (MitM) Problem</td>
<td>110</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Towards MitM Protection using OOB channels</td>
<td>111</td>
</tr>
<tr>
<td>5.3.4</td>
<td>MiTM attacks against Authenticated Key Exchange</td>
<td>113</td>
</tr>
<tr>
<td>5.3.5</td>
<td>MitM-proof Pairing and Optimized Verification Overhead</td>
<td>118</td>
</tr>
<tr>
<td>5.3.6</td>
<td>Realization of the OOB Authentication Channel</td>
<td>122</td>
</tr>
<tr>
<td>5.3.7</td>
<td>Manually Authenticated Certificates</td>
<td>124</td>
</tr>
<tr>
<td>5.4</td>
<td>A User-friendly Pairing Solution</td>
<td>125</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Discovery</td>
<td>126</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Secure Pairing Data Exchange</td>
<td>129</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Manual Authentication</td>
<td>131</td>
</tr>
<tr>
<td>5.4.4</td>
<td>Summarizing Overview</td>
<td>133</td>
</tr>
<tr>
<td>5.4.5</td>
<td>Single Interaction Pairing</td>
<td>134</td>
</tr>
<tr>
<td>5.4.6</td>
<td>Pairing Data Synchronization</td>
<td>134</td>
</tr>
<tr>
<td>5.4.7</td>
<td>Automatic Pairing</td>
<td>136</td>
</tr>
<tr>
<td>5.5</td>
<td>Pairing Daemon</td>
<td>136</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Database</td>
<td>137</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Establishing the Pairing</td>
<td>139</td>
</tr>
<tr>
<td>5.5.3</td>
<td>Utilizing the Pairing Daemon</td>
<td>139</td>
</tr>
<tr>
<td>5.6</td>
<td>Implementation</td>
<td>141</td>
</tr>
<tr>
<td>5.7</td>
<td>Conclusion and Future Work</td>
<td>141</td>
</tr>
<tr>
<td>6</td>
<td>Efficient Privacy-Preserving DNS-SD</td>
<td>143</td>
</tr>
<tr>
<td>6.1</td>
<td>Introduction</td>
<td>145</td>
</tr>
<tr>
<td>6.2</td>
<td>Related Work</td>
<td>146</td>
</tr>
<tr>
<td>6.3</td>
<td>Problem Analysis and Requirements</td>
<td>147</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Threat Model</td>
<td>148</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Requirements</td>
<td>148</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Consequences</td>
<td>149</td>
</tr>
<tr>
<td>6.4</td>
<td>Towards Privacy-Preserving DNS-SD/mDNS</td>
<td>149</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Pairing</td>
<td>149</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Hiding the Instance Name</td>
<td>150</td>
</tr>
<tr>
<td>6.4.3</td>
<td>Hiding the Service Type</td>
<td>150</td>
</tr>
<tr>
<td>6.4.4</td>
<td>Querying and Responding</td>
<td>152</td>
</tr>
<tr>
<td>6.4.5</td>
<td>Implementation</td>
<td>152</td>
</tr>
<tr>
<td>6.4.6</td>
<td>Performance Analysis</td>
<td>152</td>
</tr>
<tr>
<td>6.4.7</td>
<td>Results</td>
<td>153</td>
</tr>
<tr>
<td>6.4.8</td>
<td>Discussion of Problems and Improvements</td>
<td>154</td>
</tr>
<tr>
<td>6.5</td>
<td>Efficient Two-Stage Privacy Extension</td>
<td>156</td>
</tr>
<tr>
<td>6.5.1</td>
<td>Pairing</td>
<td>156</td>
</tr>
<tr>
<td>6.5.2</td>
<td>Private Service Directory Discovery (first stage)</td>
<td>157</td>
</tr>
<tr>
<td>6.5.3</td>
<td>Service Discovery via the PSDS (second stage)</td>
<td>162</td>
</tr>
<tr>
<td>6.6</td>
<td>Implementation</td>
<td>166</td>
</tr>
</tbody>
</table>
## 6.7 Privacy and Security Considerations

### 6.8 Network Performance Analysis
- **6.8.1 Analyzing the Number of Necessary Discovery Actions**
- **6.8.2 Comparing the Number of Necessary Discovery Actions**
- **6.8.3 Analyzing the Influence of Various Parameters**
- **6.8.4 Unicast versus Multicast**
- **6.8.5 Host Performance Analysis**

### 6.9 Privacy-Preserving Hostname Resolution

### 6.10 Privacy-Preserving DNS-SD/DNS

### 6.11 Conclusion and Future Work

## 7 Scalable Scopes for DNS-SD

### 7.1 Introduction

### 7.2 Related Work
- **7.2.1 Scalable Multicast DNS-SD in Low-Power Networks**
- **7.2.2 Centralized DNS Related Service Discovery Solutions**

### 7.3 Requirements

### 7.4 DNS-SD over Stateless DNS
- **7.4.1 Service Discovery Domains**
- **7.4.2 Providing the NS Records**
- **7.4.3 Establishing SNSes**
- **7.4.4 Synchronization of SNS data**
- **7.4.5 Deciding on a new SNS**
- **7.4.6 Updating the NS Records**
- **7.4.7 Querying the SNSes**
- **7.4.8 Hierarchical SNSes**
- **7.4.9 Security and Privacy Considerations**

### 7.5 Implementation

### 7.6 Analysis
- **7.6.1 Host**
- **7.6.2 SNS**
- **7.6.3 SNS Leader**
- **7.6.4 Reflector**
- **7.6.5 DNS Cache**
- **7.6.6 Effect of Varying the Parameters**
- **7.6.7 Unicast vs. Multicast**

### 7.7 Conclusion and Future Work

## 8 Stateless DNS

### 8.1 Introduction

### 8.2 Related Work
- **8.2.1 DNS Caches**
- **8.2.2 P2P networks**

### 8.3 DNS Caching Rules
- **8.3.1 Label-Matching Rule**
- **8.3.2 Bailiwick**
- **8.3.3 Trust-Level**
- **8.3.4 Stricter Rules**

### 8.4 Functioning of Stateless DNS
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.4.1</td>
<td>Basic Example</td>
<td>208</td>
</tr>
<tr>
<td>8.4.2</td>
<td>Method Descriptions</td>
<td>208</td>
</tr>
<tr>
<td>8.4.3</td>
<td>A: Direct Storage</td>
<td>209</td>
</tr>
<tr>
<td>8.4.4</td>
<td>C*: CNAME Aliasing</td>
<td>211</td>
</tr>
<tr>
<td>8.4.5</td>
<td>CC*: CNAME Chaining</td>
<td>212</td>
</tr>
<tr>
<td>8.4.6</td>
<td>D*: DNAME Aliasing</td>
<td>215</td>
</tr>
<tr>
<td>8.4.7</td>
<td>N*: Dynamic Delegation using NS Records</td>
<td>218</td>
</tr>
<tr>
<td>8.5</td>
<td>Implementation</td>
<td>222</td>
</tr>
<tr>
<td>8.6</td>
<td>Applications</td>
<td>223</td>
</tr>
<tr>
<td>8.6.1</td>
<td>Service Discovery</td>
<td>223</td>
</tr>
<tr>
<td>8.6.2</td>
<td>Further Applications</td>
<td>223</td>
</tr>
<tr>
<td>8.7</td>
<td>Evaluation</td>
<td>224</td>
</tr>
<tr>
<td>8.7.1</td>
<td>Bind9</td>
<td>224</td>
</tr>
<tr>
<td>8.7.2</td>
<td>Unbound</td>
<td>225</td>
</tr>
<tr>
<td>8.7.3</td>
<td>Dnsmasq</td>
<td>225</td>
</tr>
<tr>
<td>8.7.4</td>
<td>Public DNS Servers</td>
<td>225</td>
</tr>
<tr>
<td>8.8</td>
<td>Conclusion and Future Work</td>
<td>227</td>
</tr>
<tr>
<td>9</td>
<td>Conclusion</td>
<td>229</td>
</tr>
<tr>
<td>9.1</td>
<td>Concluding Summary</td>
<td>230</td>
</tr>
<tr>
<td>9.2</td>
<td>Revisiting the Research Questions</td>
<td>230</td>
</tr>
<tr>
<td>9.2.1</td>
<td>Device Pairing</td>
<td>230</td>
</tr>
<tr>
<td>9.2.2</td>
<td>Directory Discovery</td>
<td>232</td>
</tr>
<tr>
<td>9.2.3</td>
<td>Service Exploration</td>
<td>233</td>
</tr>
<tr>
<td>9.2.4</td>
<td>General Research Questions</td>
<td>233</td>
</tr>
<tr>
<td>9.3</td>
<td>Future Work</td>
<td>235</td>
</tr>
</tbody>
</table>
Privacy is not something that I’m merely entitled to, it’s an absolute prerequisite.

Marlon Brando

1 Introduction

Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Motivation</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Research Questions</td>
<td>4</td>
</tr>
<tr>
<td>1.3 Contributions</td>
<td>9</td>
</tr>
<tr>
<td>1.4 Structure of this Thesis</td>
<td>13</td>
</tr>
</tbody>
</table>

Configurationless service discovery makes network applications easier and more convenient to handle for both users and developers. But existing configurationless service discovery solutions lack privacy, do not scale, are inefficient, or cannot be conveniently and incrementally integrated into today’s network infrastructure. Addressing these problems by designing and developing a privacy-preserving, scalable and efficient service discovery framework that is easy to deploy both yields great practical benefit and poses a challenging and interesting research topic.

This introductory chapter motivates this research area, states our research questions, and provides an overview of our contributions. Our main research question boils down to how to design such a service discovery framework; we derive sub-questions and provide short answers including an outlook to the corresponding chapters. Summarizing our contributions, we also list our publications and the software developed in the context of this thesis. Our publications emphasize both the scientific aspects and the practical impact, as they are comprised of both peer reviewed papers and IETF Internet drafts adopted by the dnssd working group; as they are already adopted, these drafts are likely to become standards.
Chapter 1. Introduction

1.1 Motivation

The number of mobile devices is constantly increasing. A lot of people have a variety of smart devices with them at all times: smart phones, notebooks, tablets, smart watches, and fitness trackers. The real value users gain from these devices is due to the fact that they are interconnected. Devices aggregate information connecting to central services via the Internet, but also connect to services offered by each other, which gains more and more popularity not least because mobile devices are literally everywhere.

Before being able to communicate, devices have to discover each other; they need a means of finding out how to connect to the other device. This discovery can be handled by a central entity, which acts as a rendezvous point. But depending on a central entity comes with inherent drawbacks: The server has to be set up and requires administration, it has to be available and reachable, and users have to trust it. When Alice uploads files from her mobile phone to her notebook in a campus in Japan, she might use a server located in America as rendezvous point, and send the data from Japan to America and wait for the answer. Today, such synchronization services, which in this case simply serve the purpose of a rendezvous point, are mainly realized as could services yielding the same problems. Alice has to configure her device for using the service — maybe by downloading an application — she is dependent on the Internet backbone, and has to trust the service, which most likely persistently stores her data. Wouldn’t it be nice if she could discover her second device directly within the edge network she currently sojourns in? And, if possible, without any configuration?

This is exactly what configurationless service discovery solutions for local networks offer; widely used and well known solutions are DNS service discovery [CK13a] over multicast DNS [CK13b] (DNS-SD/mDNS — better known by its implementations’ names Bonjour and Avahi), Universal Plug and Play (UPnP) [PFKL08], and the Bluetooth Service Discovery Protocol (BT SDP) [Blu14]. There are many services that may leverage direct device-to-device service discovery in local networks, among them inter-user communication (chat), synchronization, document sharing, screen sharing, and games; further arbitrary centralized applications can leverage local network inter device communication to build and utilize a local cache for saving upstream bandwidth.

However, this comfort comes at a price. Since these solutions are based on multicast they suffer from problems inherent in multicast-based service discovery solutions: privacy breaches and being confined to a single network link.

1.1.1 Privacy: Leaking a Lot of Information

Every machine in the same network will automatically receive all the multicast announcement traffic and thus obtain a lot of information about the users in the network without having to send a single packet itself. Depending on the specific protocol, devices publish various private information, e.g. the hostname, commonly containing the user’s name, unique device IDs, and abundant information about offered and requested services. Let us look at our example student — Alice — who uses a configurationless service discovery solution in her campus network. When she enters her campus network, her notebook announces “Alice’s notebook joined” to all devices in the network. This information might either be published literally or easily derivable. The device will further announce and query for services, e.g. query for a synchronization service, announce a document sharing
service, and both offer and query for a chat service, which effectively provides the following information to every device that is listening.

- “I want to synchronize Alice’s mobile folder with Alice’s smartphone.”
- “I share the folders /home/alice/share and /home/bob/share.”
- “I am online and my chat status is gaming. Who wants to join?”

Many users are completely unaware of how chatty their devices are [KBSW13]. Most users do not consent to this information being published whenever they approach a McDonald’s or Starbucks. However, there is no user-accessible mechanism to limit or prevent this chattyness. Offering shares might allow inferring names of family members, furthermore opening pathways to social engineering attacks, while a chat application shows the user’s activity status to everyone in the same network. Most users do not know how much information is published every time they connect their mobile devices to a network or come close to a known WLAN access point.

1.1.2 Locality: Limiting Connectivity and Collaboration

Further, devices can only engage in service discovery with devices that are on the same network link because configurationless solutions are typically based on link-local multicast which does not propagate across links; and personal area network techniques like Bluetooth confine service discovery to devices that are part of the same short range radio network. Many universities confine multicast in the WiFi network to a single access point, or completely deactivate it for performance reasons. When multicast is disabled, users cannot benefit from IP-based configurationless service discovery solutions at all. Our example student, Alice, loses comfort due to these limitations in several ways. Since she joins her campus’ IP network anyway, it would be most comfortable to use this network for service discovery. This is traditionally only possible if multicast is enabled. But even if it is enabled, restrictions arise both when only working with personal devices, and when engaging in service discovery with friends all over the campus for communication, collaboration, and document sharing. Synchronization, a typical personal area service, does not work if Alice connects her notebook to the wired network, e.g. at a desk in the library, and wants to synchronize with her smart phone and tablet which are connected to the WiFi network, and thus most likely are on a different network link. Service discovery with friends might not work even if a friend stands next to her because the respective devices might be associated with different access points that do not forward multicasts. It would also be beneficial if she could communicate with friends all over the campus; she could use one of the many centralized services to this end, but this comes at the aforementioned price inherent to centralized solutions. If campus-wide configurationless service discovery was possible, Alice could just engage in communication with her friends as soon as they join the campus network. While for longer collaboration on a campus, students normally sit next to each other, they might want to just shortly check something out while at different locations. This might not be an everyday scenario, but it is limiting if it is not possible at all. Further, it would be advantageous if service discovery would even work if the university was distributed among several buildings across a city, which all are part of a joint virtual network. Just making multicast propagate across links is not a feasible solution, as multicast would cause excessive network load.

\[1\text{Users might also connect to a device masquerading as a known access point.}\]
Chapter 1. Introduction

Our goal is enhancing local configurationless service discovery by making it privacy-preserving and efficient, and further providing means to allow service discovery in multi-link networks; e.g. in a university network. We especially care about user-friendliness, efficiency and deployability in today’s network infrastructure.

1.2 Research Questions

Our main research questions can be derived from the above discussed problems and missing features inherent in today’s configurationless services discovery solutions for local networks.

When designing solutions answering these questions, we especially cared about user-friendliness, which renders designing solutions challenging. Users are in some sense spoiled by the configurationless experience offered by existing service discovery solutions and are willing to pay little to nothing for privacy in terms of both interactions and efficiency. Efficiency influences the time users have to wait as well as the battery life time of mobile devices. Typically, configurationlessness is provided by relying on multicast, which both breaches privacy, and makes scaling impossible for efficiency reasons.

1. **How can configurationless service discovery be realized without breaching the users’ privacy?**

   This problem is twofold. On the one hand, for any form of private communication, devices need prior knowledge about each other which allows for both identifying the connection end point and securing the connection. On the other hand — once prior knowledge has been exchanged, effectively establishing a relationship between two devices — devices need to be able to discover services offered by each other at any later point in time. Not knowing whether known devices are present makes this discovery process challenging.

   We solve the first problem by utilizing a manually authenticated device pairing mechanism, which — despite using minimal interaction — is designed to be especially user-friendly. The latter problem is addressed by dividing the typically multicast-based discovery process into two stages. The first stage comprises discovering relevant parts of a distributed service directory — which, for our solution, corresponds to discovering available paired devices as these maintain possibly relevant directory parts — using privacy-preserving ephemeral identifiers derived from the secret exchanged during pairing. The second stage handles the exploration of available services comprising asking online paired devices for service lists, and, after a selection process, asking for the network parameters of chosen services. We address the pairing mechanism in Chapter 5 and privacy-preserving service discovery in Chapter 6.

2. **How can configurationless service discovery be realized for multi-link networks?**

   Since a mechanism that allows service discovery in multi-link networks cannot relay on multicast for compatibility and efficiency reasons, and multicast is traditionally the very means for providing configurationlessness, this question is challenging. Generally, it is not solvable without a bootstrap mechanism. The challenge lies in designing a bootstrap mechanism that does not involve any additional configuration on host devices or servers.

   To answer this question, we can leverage insights gained from answering the previous question for privacy. Extending the area of discovery also benefits from dividing
1.2 Research Questions

the service discovery process into two stages: A special means for bootstrapping the discovery process is only necessary for discovering directory parts; service exploration can be performed by directly communicating with these directory parts, e.g. via IP unicast. The bootstrap mechanism could be realized via an overlay network; but we leverage the existing DNS infrastructure to this end. We address the multi-link discovery process in Chapter 7 and propose a special technique that allows the configurationless utilization of a DNS cache in Chapter 8.

3. How can such a service discovery solution be designed to be efficient especially with respect to network bandwidth?

Typically, adding privacy to existing mechanisms comes at the cost of efficiency. However, our pairing mechanism does not only benefit the users’ privacy but also the network load; because instead of sending information to everyone as it might possibly be relevant, we only send the information directly to hosts for whom the information actually is relevant. This does not only reduce the network load but also increases the precision of discovery queries without having a negative effect on the recall.

The additional computation time on hosts — even in large networks — is insignificant, because the operations that are dependent on the number of hosts in the network are simple, and asymmetric cryptography is used only once per connection establishment. Since before sending mDNS discovery messages hosts typically wait a random time, and this time is much longer than the time necessary to perform the privacy related operations, it further adds to the insignificance of additional computation time on hosts.

4. How can such a service discovery solution be designed to be easily deployable in today’s network infrastructure?

To be easily deployable a service discovery solution must work on the application layer and not demand changes in any deeper layer protocol. Our solutions are designed to be applicable as extensions to existing service discovery mechanisms with minimal changes.

We chose DNS service discovery [CK13a] over multicast DNS [CK13b] (DNS-SD/mDNS) as a basis for the realization of our solution, which is a widely used solution for configurationless service discovery in local networks. The components we developed for addressing the afore stated problems can be integrated seamlessly. Realizing the components as extensions to DNS-SD/mDNS allows an easy deployment as it only demands an update of the client software. This includes our bootstrap mechanism that allows direct discovery across multicast boundaries. The solution is agnostic to the network infrastructure.

As an added benefit, we can identify four main components, namely device pairing, directory discovery, service exploration, and multi-link discovery. Figure 1.1 illustrates the stages of our solution in an abstract way (we give a more detailed overview of our solution in Chapter 4). With these components being identified, we can formulate specific research questions for each of them as discussed in the following subsections. All of these must and can be realized involving little to no user interaction.

For users that want to control and tune privacy-preserving services and discovery scopes, we provide a user-interface. Except for initiating the pairing process, utilizing this
interface is purely optional; sensible default settings protect users without any further interaction.

1.2.1 Device Pairing

The specific research question for our pairing component is:

How can two devices that do not have any common knowledge exchange an authenticated shared secret in a privacy-preserving way?

Parts of this question are well answered in the literature as we show in the related work and background sections of Chapter 5. However, the question is comprised of three problems which we wish to address seamlessly and in a privacy-preserving way.

- Devices that do not know anything about each other cannot establish a connection (not even an insecure one). This demands for a means of discovery, which must not compromise the users privacy. As we stated before, automatic privacy-preserving discovery demands prior knowledge. Our solution to this sub-problem is establishing ad-hoc shared knowledge through a minimal user interaction (see Chapter 5).
- Devices need to agree on shared knowledge that can be used for an authentication at any later point in time.
- The shared knowledge has to be authenticated. To this end, we leverage manually authenticated pairing protocols, which we discuss in the background section of Chapter 5.

Designing a pairing component that seamlessly integrates these steps with as little effort as possible for users leads to a challenging balancing act between convenience and privacy. Besides secure pairing methods, we also provide a fully configurationless pairing method for trusted networks, e.g. for the home network, which greatly increases convenience when maintaining Internet of Things (IoT) devices.

1.2.2 Directory Discovery

The first specific research question for our directory discovery component is:

How can devices efficiently discover directory parts managed by peer devices in a privacy-preserving way without configuration?

Having prior knowledge — due to the pairing component — might lead to the wrong assumption of any subsequent discovery process being trivial. As stated before, hosts do not know the network parameters of devices that maintain desired parts of the directory, and — posing a special challenge — do not even know which of these devices are currently in the network. To provide a privacy-preserving means of directory discovery, we use ephemeral identifiers that are derived from the authenticated shared secret exchanged during pairing. The identifiers are the only information that is communicated in a way that everyone can access it; paired peers are able to relate these identifiers to the corresponding shared secrets, which in turn allows them to establish a secure connection. We detail this discovery process in Chapter 6.

The second specific research question for our directory discovery component is:
1.2 Research Questions

Figure 1.1: Abstract overview of the stages of our service discovery framework. Each stage is illustrated by the sketch of a possible realization, but may be realized in any way that satisfies the stage’s interface. The intra-link part of this figure abstractly illustrates the answer to our first research question. Device pairing has to be performed only once per pair of users. This figure sketches it as a close proximity pairing method, e.g. manually authenticated pairing (see Chapter 5). The inter-link part represents the scaling to multi-link networks, abstractly illustrating the answer to the second research question. The multi-link layer may be realized by various methods, among them our Stateless DNS technique (see Chapter 7), which the figure sketches as representative for realizing this layer, and Distributed Hash Tables (discussed in Chapter 2). The service exploration stage is independent of the network topology and works across links because it runs over a (virtual) secure unicast connection established during directory discovery.
How can this discovery process be realized even if multicast is not available, and how can it support discovery scopes with little to no configuration?

For privacy-preserving service discovery in network links that allow multicast, we can rely on multicast for distributing the ephemeral identifiers. To render directory discovery feasible where multicast is not available, we need other means for making these ephemeral identifiers available. While any efficient configurationless means could be used to this end — and thus could be integrated into our service discovery framework — we leverage the existing DNS infrastructure. This allows for both utilizing a widely deployed infrastructure and a seamless integration with DNS-SD over mDNS, which we use as basis for realizing our solutions. For a configurationless provision of information via the local DNS caches, we utilize our Stateless DNS technique detailed in Chapter 8.

1.2.3 Service Exploration

Treating directory discovery and service exploration as two distinct stages allows using specialized and thus optimized protocols for each layer. Most importantly, the protocol for service exploration can presume the network parameters of the corresponding device are already known. While we perform directory discovery either via multicast or via our Stateless DNS technique, service exploration is performed via a direct secure and authenticated connection.

The more practical question arising when realizing this stage is

How can a secure unicast connection for transmitting service related data be opened, given the network parameters obtained during the directory discovery step, and can this connection be authenticated in such a way that both participants only disclose information about themselves if they both are who they claim to be?

While there is research on both establishing a secure authenticated connection and private mutual authentication, given an authenticated secret, there is a lack of practical designs incorporating both. We discuss both the theory and a practical solution for establishing such a connection in Chapter 6. For the latter, we leverage TLS using the Pre-Shared Key (PSK) mode of operation [Res16].

1.2.4 Unavoidable User Interactions

During the discussion of the research questions we pointed out the fact that we rely on minimal user interaction during the pairing phase. A user interaction is necessary for (1) the privacy-preserving discovery phase during pairing, and (2) for authenticating the shared secret, respectively. The end of gaining privacy protection justifies — in our opinion — by far the means of two user interactions, especially because

• these interactions are only necessary once per pair of users,
• they are just necessary during pairing and are not required in the service discovery process at all,
• they are very simple: comparing and confirming a short number displayed on both devices, or alternatively scanning QR codes,
1.3 Contributions

- for the QR code method, we can integrate both discovery and authentication into a single interaction, and
- we provide a completely configurationless mode of operation for pairing in trusted networks, e.g. the home network.

We detail these interactions in Chapter 6.

1.3 Contributions

The main contribution of this thesis is the design, description and discussion of an efficient configurationless service discovery framework that preserves privacy and allows discovery across multicast boundaries. The framework comprises components for *device pairing*, *directory discovery*, *service exploration*, and *multi-link discovery*; these components are independent not only from a programmer’s points of view, but also work completely independently, which is especially helpful for the pairing component, as application areas besides service discovery can benefit.

Overview & Discussion.

- Our service discovery overview (Chapter 2) provides an easy and integrated access to the research area of service discovery, discussing survey papers in various fields of service discovery. We further provide a subdivision of service discovery systems into components and discuss these components’ interactions. We also discuss widely used configurationless service discovery solutions.
- We give an overview on DNS-SD/mDNS (Chapter 3), the service discovery solution that we chose as a basis for the realization of our framework.
- We give an overview on manually authenticated pairing protocols (Chapter 5).
- We present the first attack on an efficient pairing protocol from the literature claimed to be secure (Chapter 5).
- We provide an overview over our service discovery framework, and describe how our components can be integrated (Chapter 4).

Techniques.

- We provide a pairing mechanism that integrates all stages of pairing *discovery*, *pairing data exchange*, and *authentication*; our pairing mechanism preserves privacy in all stages (Chapter 5). This pairing mechanism is not dependent on our service discovery solution and can be utilized for any application area where establishing secure authenticated connections among user controlled devices are required.
- We propose a *pairing daemon* that allows managing pairings for various applications on a system. This provides further convenience for users as a single pairing allows several applications to establish an authenticated and secure connection.
- We introduce a privacy component for configurationless service discovery, which can be utilized by existing service discovery solutions. We detail a privacy extension for DNS-SD/mDNS (Chapter 6).
- We offer a mechanism that allows configurationless service discovery in multi-link networks, which utilizes the existing DNS infrastructure (Chapter 7).
- We present Stateless DNS which acts as an enabling technique for our multi-link discovery method. Stateless DNS further provides the means to utilize a local DNS cache as key-value store and may thus be used for various applications (Chapter 8).
Chapter 1. Introduction

Analysis.

• We analyze the network impact caused by multicast and a few unicasts, respectively, in networks with a shared medium half-duplex physical layer (Chapter 4). This is important for both our privacy and scope solutions as both substitute a few unicasts for a multicast.
• We analyze the effect of our privacy component on the network load (Chapter 6).
• We analyze the effect of our multi-link discovery component on the network load (Chapter 7).

1.3.1 Publications

The following publications are related to this thesis and have influenced parts of this thesis.

Adopted Internet Drafts.

[HK16b]: Christian Huitema and Daniel Kaiser. Privacy extensions for DNS-SD. Internet-Draft draft-ietf-dnssd-privacy-00, IETF Secretariat, October 2016

The criteria for the adoption of an Internet draft — which our drafts have fulfilled — are specified in RFC 7221 [FC14] Section 2.2.

The solutions I designed for Chapter 5 and Chapter 6 are the basis for the specification in [HK16b] and [HK16a], respectively. Parts of these drafts also had a feedback-effect on these chapters of this thesis, leading to mutual enrichment. I wrote Chapter 5 and Chapter 6 from scratch with only minor parts being previously published (see below).

Peer-Reviewed Papers.

[RKW15]: Andreas Rain, Daniel Kaiser, and Marcel Waldvogel. DNS and mDNS models for INET/OMNeT++. In Proc. of the 2nd OMNeT++ Community Summit, IBM Research - Zurich, 2015
1.3 Contributions

The papers [KW14a, KW14c] describe early versions of the privacy extension proposed in Chapter 5. I designed these early versions and wrote most of the text of these papers. While the naive solution described in Chapter 6 is very close to the solution described in [KW14a], only minor parts of these papers were used for describing the main contribution of Chapter 6.

Large parts of Chapter 7 have been published in [KWSH16]; an early version of a part of this solution has been published in [KRWS15]. For these papers I designed the solution and wrote most of the text.

Our paper on OmNET++/INET modules [RKW15] has been mainly written by Andreas Rain who also implemented the modules. No parts of this paper were used in this thesis.

Technical Reports.


The report on multicast DNS privacy [KW14b] is an early version of [KW14a] (see above).

Our Stateless DNS technique, which is described in [KFWD14], was developed in collaboration of Marcel Waldvogel, Matthias Fratz, and myself. The idea for Stateless DNS has been developed by Marcel Waldvogel, myself, and Thomas Zink in this order of contribution. Matthias Fratz provided the first realization of most of the Stateless DNS methods; he further implemented a comprehensive Java-based Stateless DNS echo server. Valentin Dietrich analyzed the functioning of Stateless DNS on various DNS server implementations; data gained through his analysis has been used in this techreport. I wrote most of the Stateless DNS techreport, which is why I use large parts of it for Chapter 8. Further, I have analyzed caching rules, enhanced Stateless DNS methods, and applied them for realizing multi-link service discovery (see Chapter 7).

1.3.2 Software

The following implementations either directly realize components proposed in this thesis or are strongly related to this thesis. The software has mainly been developed by Bachelor students, Master students and student research assistants supervised by the author of this thesis. We mention the names of the main developers alongside the following project listing.

To make the source code easily accessible via a single repository, all projects including the ones developed by students have been cloned in a GitLab repository\(^2\) dedicated for this thesis. In addition to the provided GitLab links, the corresponding Git repositories are also on the DVD attached to this thesis.

\(^2\)https://gitlab.com/kaiserd
Chapter 1. Introduction

Pairing.

- Pairing Daemon for Linux and Android (Valentin Dietrich)
  https://gitlab.com/kaiserd/pairing_daemon.git
  This is an implementation of the pairing daemon proposed and discussed in Section 5.5.

Privacy.

- One-stage privacy extension for Avahi (Daniel Kaiser)
  https://gitlab.com/kaiserd/avahi_privacy/tree/paper1
  While this naive implementation of a privacy extension for the Avahi\(^3\) Zeroconf daemon is not feasible for deployment, it demonstrates that privacy can be achieved with little changes in an existing configurationless service discovery daemon. We discuss the naive approach for gaining privacy in DNS-SD/mDNS in Section 6.4.

- Two-stage privacy extension for Avahi (Andreas Rain)
  https://gitlab.com/kaiserd/avahi_privacy/tree/paper2
  This implementation provides a realization of our privacy-preserving two-stage service discovery approach for the Avahi Zeroconf daemon. It is based on our research paper [KW14c]. Chapter 6 provides a substantially enhanced version of the two-stage privacy extension.

Multi-Link.

- Mini Stateless DNS reflector (Daniel Kaiser)
  https://gitlab.com/kaiserd/mini_reflector
  A minimal Stateless DNS reflector written in Perl providing the Stateless DNS means that are necessary for our multi-link extension (see Chapter 7).

- Multi-link service discovery library \texttt{libmlsd} (Holger Strittmatter).
  https://gitlab.com/kaiserd/mlsd
  Our multi-link service discovery library offers the multi-link service discovery functionality proposed in Chapter 7. It provides an interface for publishing and querying DNS-SD service instances via Stateless DNS. This library was developed in conjunction with a Master thesis [Str16].

- Avahi multi-link extension (Holger Strittmatter)
  https://gitlab.com/kaiserd/mlsd_avahi
  This implementation integrates our multi-link extension with the Avahi Zeroconf daemon. It makes minimal changes in the Avahi code mainly adding calls to our \texttt{libmlsd}. We proposed the privacy aspects of our multi-link Avahi extension in [KRWS15].

- Prototypical Scope Name Server (SNS) implementation (Holger Strittmatter)
  https://gitlab.com/kaiserd/sns
  This implementation features an SNS-capable service discovery daemon as proposed in Chapter 7. It does not provide replication and uses just a single SNS per scope.

\(^3\)http://avahi.org
1.4 Structure of this Thesis

Stateless DNS.

- **Mini Stateless DNS Server (Daniel Kaiser)**
  [https://gitlab.com/kaiserd/minisdns](https://gitlab.com/kaiserd/minisdns)
  Our minimal Stateless DNS Server is a small but fully operational Stateless DNS echo server implementation in Perl using Net::DNS. It supports the Stateless DNS methods described in Chapter 8.

- **Flexible Stateless DNS Server (Holger Strittmatter)**
  [https://gitlab.com/kaiserd/sdns](https://gitlab.com/kaiserd/sdns)
  The flexible Stateless DNS Echo Server written in C supports the Stateless DNS methods proposed in Chapter 8. Stateless DNS methods are defined in template files, which makes this Stateless DNS Server flexible allowing easily add new methods. This Stateless DNS echo server was developed in conjunction with a Master thesis [Str16].

Efficiency & Analysis.

- **OmNET++/INET models for DNS and mDNS (Andreas Rain)**
  [https://github.com/saenridanra/inet-dns-extension](https://github.com/saenridanra/inet-dns-extension)
  These models add DNS and mDNS support to OmNET++/INET. They were proposed and discussed in [RKW15]; the Master thesis [Rai15] analyzes the effect of mDNS on WiFi networks based on these models.

- **Stateless DNS Test and Analysis (Valentin Dietrich)**
  [https://gitlab.com/kaiserd/sdns_eval](https://gitlab.com/kaiserd/sdns_eval)
  These scripts were used for the evaluation of the Stateless DNS support of various name server implementations and various name servers deployed in the Internet. They were developed in the context of a Bachelor thesis [Die15]. The evaluation in Chapter 8 includes results based on these scripts.

1.4 Structure of this Thesis

On the one hand the chapters of this thesis are strongly connected as the main contribution is a seamless service discovery framework. On the other hand the components described in the chapters are loosely coupled in a sense that each of them can be used in a stand-alone way. While our components can be used independently, they gain a lot when they work together. With respect to the structure, this thesis is similar to software comprising loosely coupled components that together make up a seamless whole. We did not try to build a seamless whole from independent parts but rather subdivided the whole into seamlessly integrateable components.

For each chapter — including the overview chapters 2 and 3 — we provide an abstract, an introduction, a related work section, and a conclusion allowing readers (1) to get an overview over the chapters’ contents, (2) a motivation for the discussed component or surveyed subject, respectively, (3) an easy access to related research, which in the case of overview chapters comprises survey papers, and (4) a summary of the gained insights and the proposed techniques.

Figure 1.2 illustrates the relations of the chapters. Chapter 1 motivates the research of this thesis, formulates the research questions and summarizes our contributions. Chapter 2 provides an overview over the broad area of service discovery, divides service discovery
solutions into components and describes their interactions, and discusses configurationless service discovery solutions in detail. Chapter 3 details DNS-SD, the service discovery solution that acts as a basis for the realization of our service discovery framework. Chapter 4 gives an overview over our configurationless privacy-preserving service discovery framework supporting multi-link networks. It further demonstrates our user interface. The following chapters detail the components of our service discovery framework; Chapter 5 details our device pairing module, Chapter 6 details our privacy module, and Chapter 7 details our multi-link module that leverages Stateless DNS (sDNS), which we propose in Chapter 8. Chapter 9 concludes the thesis providing a summary and discussing future work.
The only thing that you absolutely have to know, is the location of the library.

Albert Einstein

2

Service Discovery Overview

Contents

2.1 Introduction ......................................................... 16
2.2 Related Work ......................................................... 17
2.3 Components of Service Discovery Systems ..................... 21
2.4 Interaction of Service Discovery Components ............... 25
2.5 Configurationless Service Discovery .......................... 32
2.6 Conclusion and Prospects ....................................... 46

The terms service and service discovery are omnipresent in today’s computer science literature, pertinent blogs and in the daily conversations of computer scientists. There are many different application areas for which service discovery became a crucial enabling technique.

This chapter (1) provides a broad overview over this thesis’ research area and related research areas discussing surveys for various kinds of service discovery. This chapter further provides (2) a subdivision of service discovery systems into components and discusses these components’ interactions, giving a means for both gaining a deeper understanding and classifying service discovery systems. Since the focus of this thesis is on service discovery systems that provide a configurationless mode of operation, this chapter (3) discusses widely used solutions for this application area and argues why we chose DNS-SD/mDNS as basis for the practical realization of our service discovery framework.
Chapter 2. Service Discovery Overview

(a) Direct service discovery. Each query is sent to each peer in the current network.

(b) Two-stage approach. In the first stage, relevant directory parts are discovered by querying each peer in the network; in the second stage, the relevant directory parts are directly queried.

Figure 2.1: Sending discovery queries for three services using typical configurationless service discovery and our two stage approach, respectively.

2.1 Introduction

Services of various kinds are widely used in today’s networks; they are involed in networks ranging from small personal area networks to huge cloud computing architectures. Similar to books in a library, these services are only of value if they can be efficiently discovered; a book — as well as a service — containing invaluable information is de-facto worthless if nobody is able to find (discover) its whereabouts.

The Oxford dictionary defines service as “the action of helping or doing work for someone”; it further defines discoverability as “the quality of being able to be discovered or found”\(^1\). In this thesis, we limit services to running network applications, which do in fact help or work for clients, and further require that these services are efficiently discoverable, as an eventual discovery at an arbitrary point in time is not sufficient. While we focus on discovering running network applications, discovery solutions can also be utilized to discover other discoverable entities.

Typically, the entity that maintains information about discoverable services is referred to as directory. Such a directory is what makes services discoverable in the first place. Books can only be efficiently found in a library if there is an index of the available books. In this thesis, we subdivide the service discovery process into directory discovery and service exploration (see Figure 2.1). Further entertaining the analogy to books, directory discovery corresponds to finding a library (more precisely its index) that might have the desired books. It is much more efficient to first discover a library that keeps many if not all of the desired books and retrieve them all, instead of directly searching for books in a global catalog and retrieving each desired book in a dedicated operation.

\(\text{discover, in turn, is defined as “find unexpectedly or during a search”}\)
2.2 Related Work

Today, everything exists as a service; cloud computing providers offer services on different abstraction levels ranging from IaaS (Infrastructure as a Service) over PaaS (Platform as a Service) to SaaS (Software as a Service). In the web, a myriad of network applications are provided as a service; even services that allow services to react on events caused by other services, e.g. IFTTT\(^2\), are available. In local networks, e.g. file shares, music libraries, chat connections, or printers are made available as services; since the limited scope of a single link network renders link-local IP multicast a feasible means for announcing and requesting services, these services can be discovered without any configuration, rendering them especially easy-to-use and attractive for users. Service discovery in the whole Internet without any central entity can be realized by P2P overlay networks, which, however, demands a bootstrap method as hosts that want to join the overlay network need a means for discovering at least one of the participating peers.

We mainly distinguish between centralized and decentralized discovery systems. In the context of this thesis, the most important property of centralized discovery systems is the fact that the network parameters of the service directory are known or easily retrievable. While for centralized discovery, the main issue is how to choose a fitting service among the offered ones, the main issue for decentralized systems is how to efficiently discover the directory, e.g. a node that can provide the desired service information. Improved querying techniques developed in the context of centralized systems could also be leveraged by decentralized solutions. We also assign hierarchical and federated solutions to the centralized group, as long as the topology is not dynamic. Further, we distinguish the limited scope group, which is the main focus of this thesis; it is comprised of solutions, where the scope of discovery is limited to gain advantages. Configurationless service discovery protocols are part of this group. These protocols typically limit the scope of discovery to a single-link network, which — by rendering link-local multicast a feasible discovery technique — yields configurationless discovery. Automatic discovery of services is of the essence in local networks as a static configuration would pose an unacceptable maintenance overhead. In local networks configurationless service discovery is a big issue, becoming more and more important as the number of mobile devices rapidly grows. While solutions within this group are mostly P2P based, centralized components might be involved.

In this chapter, we provide

1. a discussion of surveys for various kinds of service discovery areas,
2. a subdivision of service discovery systems into components,
3. a discussion of these components’ interactions,
4. a discussion of privacy and security aspects, and
5. a discussion of widely used solutions for configurationless services discovery.

2.2 Related Work

Since this chapter provides a survey on existing service discovery solutions, the related work for this chapter includes survey papers and comparisons of existing service discovery solutions as well as books we recommend for further reading. The introduced papers are categorized by the field of service discovery the examined solutions belong to. We also provide further reading for fields of service discovery that are out of focus for this thesis, e.g. semantic service queries.

Chapter 2. Service Discovery Overview

General. Vanthournout et al. [VDB05] present a thorough taxonomy for resource discovery in general. Since a service is a kind of resource, as discussed below, the taxonomy applies to service discovery as well. The authors present several design aspects, which allow distinguishing resource discovery solutions. They also present classes of commonly used combinations of these design aspects. While explaining the functioning of existing resource and service discovery solutions is out of focus, a summary classifying existing solutions with respect to the proposed design aspects is provided.

Meshkova et al. [MRPM08] provide a detailed survey of existing resource and service discovery mechanisms. They also provide a taxonomy used to classify and categorize service discovery solutions. Further, enabling techniques for service discovery, e.g. packet propagation, multicast, push and pull, are thoroughly covered. P2P Networks — both unstructured and structured — are covered in detail examining used techniques like search methods and replication strategies and — in contrast to [VDB05] — also presenting a lot of concrete designs and implementations. Service discovery systems for enterprise networks as well as local networks are also covered. The authors also discuss challenges for service discovery in wireless and ad-hoc networks, and present existing service discovery mechanisms for these network types. Because the paper is about seven years old and service discovery is a hot research area, it is necessary to also look at more recent solutions; especially in the field of configurationless service discovery and service discovery in the cloud.

P2P Networks. Lua et al. [LCP+05] provide an introduction to P2P networks, introduce a P2P network taxonomy, and give a thorough and detailed survey on various structured and unstructured P2P overlay network architectures, while focusing on different overlay network routing algorithms. Among the surveyed P2P architectures are classical structured approaches like CAN [RFH+01], Chord [SMK+01], Pastry [RD01], Tapestry [ZHS+04], and Viceroy [MNR02]; and classical unstructured approaches like Freenet [CSWH01], Gnutella [Gnu01], Fasttrack [Har04], Overnet [Ove02], and BitTorrent3. Since P2P networks were mainly researched and developed at the beginning of the 21st century, this survey is still relevant. We consider it one of the best resources to get familiar with the area of P2P networks.

Malatras et al. [Mal15] focus on the application of P2P overlays in a pervasive computing environment proposing a taxonomy, analyzing various architectures and classifying existing solutions with respect to the proposed taxonomy. They mostly analyze the aforementioned classical architectures — explained in detail in [LCP+05] — and enhancements making classical architectures suitable for pervasive computing. Among these enhancements are techniques for adapting to dynamic mobile devices and approaches for bridging various overlays.

The following surveys focus on non-functional, particularly security related, aspects of P2P networks. Androutsellis et al. [ATS04] provide a framework for analyzing the affect of different design features on resulting non-functional properties of P2P systems for content distribution and storage. Using the proposed framework, they thoroughly discuss and analyze several existing implementations of P2P systems.

Wallach [Wal03] presents an abstract model of the aforementioned classical, structured P2P overlay routing protocols, describes Pastry [RD01] as a representative and points out the differences to the other classical, structured P2P overlays. The presented abstract model is then used to analyze security, fairness and trust issues that arise when using

3http://www.bittorrent.com
2.2 Related Work

structured P2P overlay networks for file sharing. Countermeasures increasing the level of
security, fairness and trust are also presented.

Amoretti et al. [Amo09] illustrate various aspects of security, trust or efficiency that
classify P2P overlay networks. They point out the influence of different architectural
choices, namely the hybrid model\(^4\), and the decentralized structured and unstructured
models. For some efficiency related problems, they discuss various layered models as a
solution, which can be any combination of the afore mentioned models. Further, they
survey classical overlay network architecture solutions and categorize them with respect to
the described architectural choices and characteristic aspects.

Client-Server-Based Service Discovery. When discussing client-server based service
discovery systems, we do not limit the discussion to classical systems where there is a
single server and multiple clients. We also take distributed systems, both flat-federated
and hierarchical, into consideration. The discriminating factor is the distinction between
the client and the server role. Further, since clients know (or may easily gain the necessary
information) how to contact the servers, the main focus of these systems is efficient selection
of fitting services rather than discovering the servers.

Takada’s book on distributed systems [Tak13] provides an easy-to-understand intro-
duction to modern distributed architectures including service discovery aspects. It details
consensus systems and storage back ends, which may act as building blocks, e.g. as a
service registry, for managed client-server based service discovery systems.

In the last few years the discovery of cloud services became one of the most prominent
representatives of client-server based service discovery. Hoefer et al. [HK10] provide a
taxonomy of cloud services which is a tree, where the top level is comprised of the NIST
classes [MG11]: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and
Software as a Service (SaaS). Since for any client-server based discovery solution the main
issue is to improve the relevance of retrieved service instances such a taxonomy is of the
essence. Cloud service selection approaches, whose goal it is to efficiently and accurately
select services based on specific user requirements, are thoroughly surveyed and classified
by Sun et al. in [SDH14]. The term selection implicitly stresses the fact that these
mechanisms select services among a set of available services rather then discover them.
The authors point out that both standards for specifying cloud services and means for
choosing appropriate services are necessary.

A field that is related to cloud service discovery but should not be confused with it, is
web service discovery. Zunino et al. [ZC12] survey approaches to web service discovery
and classify them according to a set of proposed criteria. Klusch [Klu14] provides a concise
but thorough overview of (web) service discovery, covering all important aspects in an
easy-to-understand way. We strongly recommend this paper for getting familiar with web
service discovery.

Limited Scope. Bettstetter et al. [BR00] provide a comparison of SLP [GPVD99,
Gut02], Jini [ASW99], Salutation [C99b] and UPnP [Um99]. The main focus is on SLP
for which an implementation is presented. Despite being over 15 years old, the paper
is worth reading as it provides anticipatory motivational scenarios for local scale service
discovery and a concise, easy-to-understand explanation of how SLP works. SLP has been
widely replaced by DNS-SD/mDNS, but since it is one of the first well known service

\(^4\)The hybrid model uses a client/server structure for resource publication and discovery, and a P2P
approach for resource consumption. Napster [Nap01] is an example for the Hybrid Model.
discovery solutions for local scale networks, can scale to multi-link enterprise networks, and inspired younger generations of service discovery solutions, we consider it worthwhile to understand the functioning of SLP. Further the paper allows retracing parts of the history of service discovery.

The book on Zero Configuration Networking by Cheshire et al. [SC06] provides an introduction to all stages of Zero Configuration Networking, among them Zero Configuration Service Discovery — DNS-SD over mDNS — which is the service discovery solution we use as basis for realizing the service discovery framework proposed in this thesis. We will discuss DNS-SD over mDNS in Section 2.5 and especially in Chapter 3. An introduction to UPnP from a developer’s point of view is given in [JW03], a book by Jernonimo et al. Obitsching [Obi06] generally discusses configurationless service discovery, lists arising issues, and explains how these issues are addressed by various solutions; among these solutions are DNS-SD/mDNS and UPnP.

Pervasive Computing and Mobile Ad-hoc Networks (MANETs). While the following surveys mostly cover solutions in the field of scope limited discovery, their main focus is in the field of pervasive computing and mobile ad-hoc networks; thus these approaches either especially deal with very limited resources of the involved devices and special routing problems, respectively, or even both.

Edwards [Edw06] surveys and compares several limited scope service discovery solutions, among them SLP, DNS-SD/mDNS and SDP (Bluetooth), in the context of pervasive computing. He points out that these systems were not specifically designed for pervasive computing, and discusses requirements for pervasive computing service discovery systems. Vakkosov et al. [VNP15] survey discovery protocols suited for resource constrained devices, especially focusing on energy efficiency. Raychoudhury et al. [RCKZ13] survey pervasive computing middleware. These middleware solutions often contain a service discovery component. Besides other components, the authors discuss this service discovery component — the service management service — and also classify existing service discovery approaches.

Machine to machine (M2M) communication is a further field in which automatic service discovery is of the essence, as human interaction should be reduced to a minimum. M2M communication often involves resource constrained devices, e.g. sensors. Kim et al. [KLKY14] comprehensively discuss M2M service platforms. Villaverde et al. [VDPAJ+14] especially survey service discovery protocols for M2M communication of resource constrained devices.

Mian et al. [MBB09] provide a comprehensive discussion of discovery protocols in the context of multihop mobile ad-hoc networks. The authors explain components of service discovery frameworks in general, and especially address issues arising when building a service discovery framework on a multihop MANET basis. They further survey existing service discovery solutions for multihop MANETs. Girolami et al. [GCC15] survey service discovery solutions in the context of mobile social networks. Service discovery is a crucial component in distributed social networks as devices have to be able to efficiently and conveniently discover desired and also unsolicited relevant information. The authors discuss the service oriented nature of mobile social networks, some of which also support mobile ad-hoc networks to allow interaction where a network infrastructure is not available.

Semantic Service Discovery and Advanced Queries. We also introduce surveys on sophisticated service selection approaches. Fensel et al. [FFST11] provide a comprehensive
book on various aspects of web services with a special focus on semantics. They point out that semantics are of the essence for efficient, scalable, and convenient discovery of web services among the plethora of offered web services. Klusch et al. [KKS+15] provide a concise survey on the same matter with a similar motivation. Dong et al. [DHC13] survey match-makers for semantic web service discovery with a focus on challenges when designing such match-makers. Wang et al. [WCC09] survey semantic web service discovery approaches in the context of ubiquitous computing.

2.3 Components of Service Discovery Systems

We assume service discovery systems are comprised of four major components: service, client, service provider, and service directory; while services are the objects of discovery, the remaining three components are agents involved in the discovery. These agents are logically separated entities, but they may very well run on the same device or even be combined in software.

Despite the fact that we could use the components described in this subsection and their interactions described in the next subsection to establish a taxonomy by also defining different ways of realizing these entities, we refrain from that because of the following reasons. (1) Various taxonomies for service discovery systems already exist [VDB05, MRPM08, MBB09]; (2) it is (almost) infeasible to create a comprehensive taxonomy, because of the plethora of aspects, certain features condition others, and only a few combinations are really relevant; (3) we want to detail the parts that are in the focus of this thesis; and foremost, (4) we consider it more helpful for designing a discovery protocol to define a set of components and their interactions, rather than defining a comprehensive hierarchy of properties where most of the theoretically possible attribute combinations are not sensible.

Figure 2.2 both illustrates the general service discovery components — service, client, service provider, and service directory — and shows realizing the service provider and the service directory as separate components.

2.3.1 Service

In the context of this thesis, we specify a service as a running application (a process) accessible through a network. Especially, we consider applications that are accessible through the IP network via a specific application layer protocol. At any given time a service is available, it must have certain network parameters that allow addressing it within the network; for IP-based services these network parameters are comprised of the IP address, the port, and the respective application layer protocol. Even a PaaS cloud service can be seen as a network application, as it is most likely accessible via SSH, which is an application layer protocol.

Service Description. Besides the information necessary to address and access a service, a service may be associated with a service description. A service description can e.g. be comprised of key-value pairs as in DNS-SD [CK13b], or use a description language like WSDL [CCM+01]. Further, the service description may contain run-time status variables. We refer to all information related to a service, i.e. the information needed to access the service and the service description, as service-related information.
Chapter 2. Service Discovery Overview

Figure 2.2: Components of a service discovery system. In this example, the service directory is maintained by a dedicated directory server.

**Service Instance.** For clarity’s sake, some service discovery solutions (among them DNS-SD/mDNS) use the term *service instance* instead of the plain *service* to indicate its instantiated and concrete character. This demands the existence of *service types* or *abstract services* that can be instantiated. While our service discovery framework, as well as its basis DNS-SD/mDNS, uses the concept of *service types* and *service instances*, we do not want to restrict our discussion to these concepts and rather consider the *service type* as an optional part of the *service description*. So, in our discussion both *service* and *service instance* refer to the same concept.

**Access Control.** There might be a service specific access control mechanism that determines which users or groups of users are allowed to access certain services. Many service discovery systems (especially configurationless) allow users that are not authorized to access a service the discovery of this service anyway. While access control is out of the scope of a service discovery system and we do not consider it in this thesis, our privacy extension (see Chapter 6) restricts the discoverability of services to services a user is very likely to have access rights for, which increases efficiency.

**Generalization.** While we focus on service discovery, many aspects we introduce and discuss in this thesis are not bound to a service being the discovered entity. Services can
be generalized to arbitrary entities that are discoverable. A less generic generalization of a service is a resource, which — besides a service — could also be a file, computational power, or memory. Vanthrournout et al. [VDB05] provide a taxonomy for resource discovery.

2.3.2 Client

The client is a host using service discovery to discover a service it wants to consume. A host is not limited to only being a client or a server like in classical client-server architectures, but can have both roles.

Connection. After a client has discovered a desired service (we detail the process in the following section) it may establish a connection using the retrieved network parameters. Establishing this connection is not part of the service discovery solution, which has fulfilled its duty after delivering the network parameters.

Cache. Clients may have a cache in which they manage the network parameters of recently discovered services. A cache can greatly reduce network load but may contain stale information if entries are maintained for too long.

2.3.3 Service Provider

Service providers run services and provide them to clients by making them discoverable and accessible. Services are made discoverable by registering service-related information in a service directory.

2.3.4 Service Directory

The service directory manages services available in a discovery system by storing and maintaining service-related data (i.e. both network parameters and descriptions of services).

Most literature distinguishes service discovery systems that have a directory and systems that do not. In systems that do not have a directory, the service information is maintained by the service providers that offer the corresponding services. We, however, consider this as a special case of distributing the directory among nodes and thus act on the assumption that all discovery systems necessarily have a service directory. We refer to this distribution as local distribution because service providers locally maintain the corresponding service information. We consider this advantageous in two ways:

1. We generally think that modeling an exception as a further realization of the class it is an exception of, is preferable as long as it is feasible. In this case, it (1) allows the taxonomy hierarchy to save a level, as every system has a directory and, with respect to the directory, is just distinguished by how the directory is distributed; further it (2) removes a special case from our component diagram as the directory is a (logical) part of all service discovery systems.

2. We introduce service directory discovery as a crucial interaction in service discovery systems. Regarding directory discovery as a separate interaction is beneficial, especially for multicast-based systems where the directory is distributed in such a way that each node maintains the services it provides. Our privacy extension detailed in Chapter 6 is an example.
Chapter 2. Service Discovery Overview

**Topology.** We distinguish the following kinds of topologies for service directories.

*Central:* A single central (logical) entity with fixed network parameters is responsible for providing the service directory, which can be realized either by a single host or by a group of hosts for load balancing proposes. An example is a UDDI registry, which is used to register WSDL [CCM+01] based web service descriptions.

*Hierarchical:* A hierarchical directory is comprised of at least two levels, where the leaf level contains the actual service-related information and the other levels contain pointers to the respective next levels. The omnipresent DNS [Moc87a] uses a hierarchical directory.

*Distributed Hash Table:* A distributed hash table (DHT) distributes the data among nodes by hashing data items and assigning to each node the data items whose hash value is in a certain range of the image space of the hash function. Structured P2P networks often are based on a DHT; examples are Chord [SMK+01] and CAN [RFH+01].

*Local:* Using the local topology, every service provider maintains a directory for the services it offers. DNS-SD over mDNS [CK13a, CK13b] uses the local topology. Also, unstructured P2P networks use this topology. If a service provider leaves the network, the part of the directory which contains its services will become unavailable. This is desired, as a service is a concrete instance and is bound to its provider; when the provider leaves the network, the service will become unavailable anyway. Discovery systems for different resources, e.g. files, need to replicate the resources on other nodes to keep them discoverable even when the original provider leaves the network. Even when discovering services, parts of the directory might get replicated in the caches of clients; to avoid stale information, devices that act as both client and part of the service directory should not feed data accumulated in the cache to the directory.

While the network topology and the construction are separate dimensions (see e.g. [VDB05]), for most applications the topology conditions the construction. For all discussed kinds of central directories we make the common assumption that these systems are manually administered and that clients can access the directory via fixed (or rarely changing) network parameters. For hierarchical directories we also assume manual administration. Further, we assume little to no administration for the DHT and local topologies.

**Storage.** The directory can store the service-related information either on the host it is running on, or just process queries and use independent storage nodes. The storage layer is independent of the directory and can be realized in an arbitrary way as long as it provides the directory with an interface to store and retrieve service-related information.

Large centralized solutions often use a dedicated storage component; e.g. SkyDNS\textsuperscript{5} uses etcd\textsuperscript{6} as distributed key-value store. Hierarchical systems and DHTs may store service-related information on the nodes themselves as the topology itself provides distribution and thus potential load balancing. But especially when using DHTs for storing larger resources, e.g. larger files, the DHT just stores a pointer to the node containing the actual data; this is referred to as indirect storage. For local network service discovery solutions, service-related information is typically stored on the service providers, often in non-persistent memory.

\textsuperscript{5}https://github.com/skynetservices/skydns
\textsuperscript{6}https://github.com/coreos/etcd
2.4 Interaction of Service Discovery Components

We distinguish three crucial interactions between these components: service registration, directory discovery, and service exploration. Each of these interactions might be realized either in a separate protocol, or as logical parts of the same protocol. Figure 2.3 both illustrates the interactions between the service discovery components in general and shows an example of how these interactions could be realized.

2.4.1 Service Registration

Service providers have to register services at the service directory to make them discoverable by clients. To this end, service providers have to discover the part of the service directory that is responsible for the new service (see Subsection 2.4.2). If service providers act as directory for their own services (local topology), directory discovery is trivial and the services can automatically be registered at the local directory. For any other topology,
service providers discover the directory similar to clients that want to consume services. Once the service-related information is handed to the service directory, it is the service directory’s job to store the information appropriately to fulfill the service exploration requirements of the respective service discovery systems.

Registration can be performed either manually or automatically. An example for manual registration is the DNS where retrievable information is stored in zone files that have to be manually edited to add new information. DNS further provides means for automatic service registration known as DNS update [VTRB97]; but still it is not configurationless as service providers both need credentials and have to retrieve the network parameters of the respective DNS server. A truly automatic and configurationless way of service registration is provided by DNS-SD/mDNS, because it is based on the local directory topology and announces services via multicast.

Besides the service providers registering services themselves, registering services can be outsourced to a third party. Similar to a service broker helping during service exploration, third-party registration is mainly used for huge scalable service discovery solutions with a central topology.

Another important point that has to be considered is service deregistration. If services are not deregistered, clients will discover more and more unavailable services. Ideally, a service is deregistered as soon as it becomes unavailable. If a service is gracefully shut down, meaning the service provider itself properly deregisters the service, immediate deregistration is possible. Problems arise in the case of an ungraceful shutdown.

As systems using a central or hierarchical topology are typically administered, service providers are expected to explicitly remove services that are no longer available. If a service provider shuts down ungracefully it is expected to rejoin the network and either restart the service or properly deregister it. DNS, for example, stores resource records indefinitely even if the resource is no longer reachable, leaving deregistration or updating to service providers. Some systems, e.g. Consul\(^7\), provide sophisticated health checks automatically removing unavailable services from the directory.

In structured overlay networks frequent (ungraceful) shutdowns — referred to as high churn rate of nodes — are a big issue and much research has been done in this field. Typically, nodes periodically check whether their neighbors are still reachable, and, if not, take over the job of an unreachable neighbor.

The local topology provides an efficient straightforward solution to deregistration as service providers manage the part of the directory that contains their services themselves. When a service stops, the provider can immediately delete it from the directory, and if the service provider shuts down ungracefully, the corresponding part of the directory becomes unavailable as well. However, stale information might still be in the caches of clients. This is not a problem because the directory does not contain stale information, so the caches are updated when querying the next time.\(^8\)

### 2.4.2 Directory Discovery

In the literature directory discovery is not treated as a separate entity. Typically, what we refer to as directory discovery is modeled as an integrated part of the query routing process. However, as we are in the domain of discovery itself, we prefer to model and describe

---

\(^7\)https://www.consul.io

\(^8\)In a purely push-based system, clients would not learn about directories that become unavailable. Clients should remove a service from the cache if the connection establishment fails a few times.
2.4 Interaction of Service Discovery Components

the process of locating the desired part of the directory as a discovery process. In fact, it can be seen as discovering a special service — we refer to this service as the directory discovery meta service. The service directory itself is accessible through a running network application which is provided as a service.

Directory discovery comprises all the steps necessary to locate (parts of) a service directory which contains the service-related information of desired services. The afore discussed topologies for service directories condition the way the respective directory is discovered.

Central. Discovering a central service directory is trivial if — as we assume — the network parameters of the directory are known to clients. The central service directory can either be manually configured on client devices or be retrieved via DHCP [Dro97]. Thereafter, the client can directly establish a connection to the directory, which already concludes the directory discovery phase. Since directory discovery is straightforward, discovery solutions based on centralized directories focus on improving service selection, efficient fail-safe storage, and health checking, while for certain other topologies directory discovery is one of the main challenges.

Hierarchical. Using a hierarchical system for discovery, we assume that clients manually enter the network parameters of the root node or can retrieve them via DHCP. Clients send a query to the root node that contains at least as much information as necessary for the root node to decide which of the next level’s directories are relevant for the client. After retrieving the network parameters of the relevant next level directories, the client can iterate this process until finally retrieving the network parameters of the relevant leaf level directories, which ends the process of directory discovery. In accordance to the DNS [Moc87a] nomenclature, this way of querying a hierarchical directory is referred to as iterative.

As in the case of DNS, which is the most prominent example of a hierarchical directory, dedicated nodes may act as cache server. Instead of configuring the root as entry point, clients may configure and query the cache server, which iteratively discovers the relevant leaf directories, and directly returns them to the client. From the client’s point of view, it looks like a single query suffices to get the desired answer; this kind of querying is referred to as recursive querying.

Both clients and cache servers may cache information corresponding to arbitrary levels of the hierarchy. The huge advantage of the cache servers is that all clients accessing the same cache server benefit from its cache.

Distributed Hash Table. Besides the aspects necessary for storing data — key space definition and distribution of the key space among the nodes — a structure that allows efficient routing of queries to discover the relevant parts of the directory is necessary. This third aspect is often referred to as overlay routing. To engage in the discovery process, clients need to know the network parameters of at least one of the directory nodes. This information can be configured manually, e.g. by looking it up on a web page, or it can be retrieved leveraging another discovery mechanism, e.g. the DNS. Any query can be sent to any node because the nodes know whom to forward the query to if they cannot answer it themselves. The overlay network is constructed without administration and maintains itself. Efficient DHTs limit the necessary overlay hops a query needs until it reaches the desired node to $O(\log(n))$, where $n$ is the number of nodes in the overlay network.
Well known DHT-based overlay routing algorithms include Chord [SMK+01], CAN [RFH+01], Pastry [RD01], Tapestry [ZHS+04], and Viceroy [MNR02]. A detailed explanation of these DHT architectures can be found in [LCP+05]. Advanced query mechanisms that include sophisticated descriptions\(^9\) of entities are proposed in [BHPW04]. The advantage of being agnostic to the underlying network topology is also a disadvantage with respect to expected message propagation times. Waldvogel et al. [WR03] propose a layer between the transport layer and the overlay network enhancing DHTs, making them topology-aware and more efficient. The routing algorithms for distributed hash tables can also be used as means for overlay multicast [RHKS01]. Pendarakis et al. [PSVW01] present an architecture specifically designed for overlay multicast.

**Local.** Typically, the directory is not explicitly discovered when using the local topology; instead, queries directly asking for certain entities are sent to every (reachable) node in the network, which is referred to as flooding. Flooding performs directory discovery and the discovery of desired entities in a single step. Local service discovery solutions often use flooding realizing it by (IP-based) broadcast or multicast. This solution does not scale, as each query causes an \(O(n)\) increase in network load, where \(n\) is the number of nodes currently in the network. For this reason, multicast is often limited to a single link or completely deactivated.

Unstructured P2P networks also often use flooding realizing it — oversimplified — by sending queries to every known online node, which checks whether it can answer the query, and, if not, in turn relays it to all its known hosts. This corresponds to a breadth first search in the overlay graph. It can be optimized by adding query IDs allowing to avoid relaying the same query twice, by limiting the search depth, or even applying sophisticated algorithms to determine whether a query should be relayed to a certain host. Further, random walks can improve the network load for searching in unstructured overlays, approximately corresponding to a depth first search strategy. Generally, searching in unstructured networks has to find a compromise between network load and recall. Well known unstructured P2P network architectures are Freenet [CSWH01], Gnutella [Gnu01], Fasttrack [Har04], Overnet [Ove02], and BitTorrent\(^10\). A detailed explanation can be found in [LCP+05]. Sophisticated search methods based on semantics in unstructured peer-to-peer networks are addressed in [Cao15].

However, even in an unstructured network, first discovering the directory — via multicast (see Chapter 6) or by other means (see Chapter 7) — is beneficial for both privacy and efficiency if the directory parts contain entities that are related.

**General.** In general, the query for discovering the directory may contain no service-related information at all, a subset, or all service-related information that desired services should have. This also depends on whether the client knows which services or service types it desires during the process of directory discovery. If all desired service-related information is specified during the directory discovery phase, typically, the directory discovery phase is only virtual, or happens seamlessly from a clients point of view. An example for a virtual service directory discovery is DNS-SD over mDNS, where the directory is never explicitly

\(^9\) While sophisticated service descriptions are mainly used during the service selection step of the service exploration phase, we mention this work here as it is in the context of DHT routing, which in turn mostly belongs to directory discovery. Since the existing literature does not explicitly distinguish a directory discovery phase, these two phases are often intertwined, but still logically separable.

\(^10\) [http://www.bittorrent.com](http://www.bittorrent.com)
2.4 Interaction of Service Discovery Components

queried. Standard DNS uses an integrated directory discovery process, as clients query for the resource they desire, and the cache server performs both the directory discovery and the following service exploration. Using queries for NS resource records, DNS also allows explicitly querying any directory level. Our privacy extension proposed in Chapter 6 uses no service-related information during the directory discovery phase. While the discovery daemon — similar to the standard DNS cache server — explicitly performs directory discovery and service exploration, respectively, clients are provided with a seamless user experience.

Modeling directory discovery as a separate step is the more useful the more entities stored in a part of a directory are related to each other. When a client discovers a directory and desires many, if not all, services of this directory, one discovery step has been sufficient to retrieve a lot of entities instead of separately performing a discovery for each resource. For our privacy extension this especially applies as a directory contains services offered by trusted friends, which are very likely to be of interest. The opposite is the case of a DHT that does not use consistent hashing randomly distributing the discoverable entities among nodes; in this case directory discovery is purely virtual.

2.4.3 Service Exploration

After discovering the service directory, clients may explore the services offered by this directory. We further subdivide the service exploration phase into service listing, service selection, and service location. While these sub-phases can always be logically distinguished, they may be intertwined in practical applications. Further, some of these service exploration sub-phases may even be intertwined with the directory discovery phase. Figure 2.4 illustrates the sub-phases of service exploration.

Service Listing. During service listing a subset of the services maintained in directory parts discovered during the directory discovery phase is chosen and presented as a list to the entity responsible for the service selection sub-phase. Depending on the listing query mechanism, this service list might be comprised of services of certain service types or be a result of applying more sophisticated filters. The listed services might be represented by simple service names, by parts of their descriptions, or by their whole descriptions.

It is advantageous to realize a dedicated service listing sub-phase if service selection is not performed on the directory node but either on the client or on a third-party entity, e.g. a service broker. In DNS-SD/mDNS, service listing is realized by browsing for a certain service type. We discuss DNS-SD/mDNS in Chapter 3.

Service Selection. During service selection, relevant services are selected among a subset of the services maintained in a directory. The selection can be performed either manually, or by a service match-maker, or by a service broker. Match-maker or broker can be realized as a component of the client or as a dedicated entity. While a match-maker automatically selects relevant services, a service broker additionally handles connecting to desired services without further interaction. The more service-related information is provided by the service listing sub-phase, the more accurate the service selection mechanism.

\(^{11}\) A further phase is service adaptation, which is out of focus of this thesis because it is not an integral part of the discovery process. Service adaptation allows adapting the service behavior to a client’s needs (see e.g. Da Capo++ [SCW+99]).
Chapter 2. Service Discovery Overview

Figure 2.4: Sub-phases of service exploration. In this example, all the sub-phases are dedicated; the service description is provided during service location.

can operate; but sending a lot of information also affects bandwidth, which we discuss later. DNS-SD/mDNS for example leaves service selection to the users of client devices.

Service Location. The goal of service location is retrieving the network parameters of selected services by querying the service directory. If the service description is not delivered during service listing, it is delivered alongside the network parameters during the location sub-phase. DNS-SD/mDNS locates services by querying for SRV resource records of selected service instances; service descriptions are provided via TXT resource records.

Intertwining Phases. In practice some — if not all — of these phases are intertwined. If a query is specific enough, service selection may be performed directly on the directory nodes. In this case there is no explicit listing sub-phase; it is integrated in the selection sub-

phase. Listing might also be intertwined with location, providing the network parameters at once when listing service instances.

Centralized solutions might integrate all service exploration sub-phases using queries that directly specify desired services and are answered with the corresponding network parameters. Larger centralized solutions might offload selection to a dedicated server, which could even be maintained by a third party. Combination of listing and location is advantageous where network bandwidth is not the problem and saving an extra round-trip time is more important. However, for multicast-based local discovery systems this is different. Multicast-based discovery systems (e.g. DNS-SD/mDNS) typically intertwine directory discovery with service listing and thus have a low precision with respect to retrieving relevant service instances; further, multicast consumes a lot of bandwidth compared to unicast (see Chapter 4). So, separating listing and location — effectively listing just names of services, selecting a few and then asking for further information about these services — is beneficial for this application area. DHT based P2P networks even intertwine all phases; the queries are routed to relevant parts of the directory and the nodes responsible for the corresponding directory parts directly answer with the desired data. In the case of indirect storage the client is provided with means to address the node actually storing the desired data, which can be seen as a separate location sub-phase.

Protocol. The protocol clients and service directories use for service exploration can be based on push, pull, or a mix thereof.

**Pull:** Clients send queries covering every sub-phase of service exploration; either separately or in an intertwined way as described above. Service directories only send service information after being queried.

**Push:** The service directory unsolicitedly pushes service information to clients. These pushes can either be performed in certain intervals — short intervals may cause a huge network load, while long intervals may cause stale information — or they can be performed when new services are added to the directory. Both methods should be used, so that clients joining the network eventually get a complete list of available service instances and new services are announced soon after registration. This method scales badly as service announcements have to be broadcast; but it is beneficial in small networks with weak clients, as clients just have to listen to the announcement messages. Both directory discovery and service listing are performed by the directory pushing the respective information. Either directories might push the location information alongside the listing in a purely push manner, or clients might query for location information after selecting desired services among the ones received in a push message.

**Mix:** Clients may query in a pull manner and may additionally register to receive push messages, yielding a publish & subscribe protocol. Clients might just query for a service listing when joining a network and register to receive push messages whenever service information becomes available, becomes unavailable, or changes. Clients might also use push registration in a more fine grained way, registering for updates on certain service types or even updates for services matching a certain description, while still actively querying for certain other service types.

For our service discovery framework, we use this technique, allowing clients to register for push of certain service types while still allowing pull queries. This is both efficient in terms of network bandwidth and avoids stale information. Push is used for service
types whose list should be up-to-date, e.g. a contact list in a chat application, while pull is used for service types that are used at a specific point of time, e.g. a data synchronization service. It might be tempting to register for the whole directory to always be up-to-date, but this leads to a high network load and scales badly.

2.5 Configurationless Service Discovery

Since configurationless service discovery is the focus of this thesis, we detail this area of service discovery discussing solutions that are widely used in today’s networks. We further argue why we chose DNS-SD over multicast DNS among these solutions as a basis for realizing our privacy-preserving service discovery framework. Most of the discussed solutions are part of a protocol stack; we will not only discuss the discovery part alone but the whole protocol stack as it gives insights to why certain design choices were made.

Solutions for configurationless service discovery in local networks are typically built on multicast and use the local topology. They do not distinguish between a directory discovery and a service exploration phase, and directly query services. Service selection is typically performed manually by the users of client devices. Basically, solutions for configurationless service discovery, e.g. DNS-SD/mDNS and UPnP, gain their configurationlessness by sending both discovery and announcement messages to every device on the same network link.

Summarizing, these protocols provide configurationlessness and a recall of 100% (theoretically, not taking transmission problems on the network layer into consideration) at the cost of a high network load and a possibly very low precision. We specifically cover DNS-SD/mDNS [CK13b, CK13a], UPnP [PFKL08], and Bluetooth [Blu14], because they are widely used and the most prominent representatives for the classes of horizontal, vendor independent automatic service discovery, vertical automatic device management, and personal area network device discovery, respectively. Other multicast-based service discovery solutions, e.g. NetBIOS [Gro87a, Gro87b], as well as multicast-based name resolution systems, e.g. Link-Local Multicast Name Resolution (LLMNR) [ATE07], suffer from similar shortcomings.

2.5.1 Zeroconf and DNS-SD/mDNS

The focus of the thesis is on DNS-SD/mDNS, which is the configurationless service discovery solution of the Zeroconf stack. We shortly discuss the whole Zeroconf protocol stack for comparison to other protocol stacks (we detail mDNS and DNS-SD in Chapter 3). The Zeroconf stack provides configurationless means for all of addressing, name resolution, and service discovery (see Figure 2.5). A huge advantage is that these layers are independent of each other. The name resolution mechanism — multicast DNS (mDNS) — works with automatic [CAG05] or static address configuration as well as with DHCP. The service discovery layer — DNS service discovery (DNS-SD) — works with standard DNS as well as mDNS. In the future, other realizations of these layers could be developed and integrated seamlessly.
A configurationless user experience at a certain layer is only granted if the deeper layers do not need configuration; thus DNS-SD over standard DNS will require the configuration that DNS demands, either via DNS update [VTRB97] or manually editing zone files.

**Addressing:** Local address auto configuration [CAG05] allows automatically establishing an IP-based network without the need of statically configured IP addresses or a DHCP server. Basically, hosts randomly choose an IP address from the IPv4 link-local range 169.254.0.0/16 (excluding the first and last 256 addresses as they are reserved for future use) and check whether the chosen address causes conflicts. The general problem of solving address conflicts using ARP probes is addressed in [Che08]. For IPv6, local address auto-configuration is integrated and handled by the neighbor discovery protocol (NDP) [NNSS07].

**Name Resolution:** With mDNS [CK13a], the Zeroconf stack provides a means for configurationless name resolution. Like the standard DNS, mDNS allows querying for DNS resource records, e.g. for A resource records which map names to IP addresses. Instead of querying a DNS server, the query is sent via multicast to everyone on the same network link, and is in turn answered via multicast by devices that provide desired resource records. mDNS reserves the .local top level domain for exclusive usage in local networks. Each device supporting mDNS announces an A resource record that maps its host name (or any user-chosen name) to its IP address; when receiving a multicast query for this name, the device answers via multicast.

**Service Discovery:** DNS Service Discovery (DNS-SD) is a means for general purpose service discovery leveraging DNS. While DNS offers MX records for discovering the mail server of a given domain, DNS-SD allows discovering arbitrary services. Means for both service listing and service location are provided. Service listing is realized by a special usage of PTR resource records, which are normally used for reverse address mapping. Each service instance is associated with a DNS-SD PTR record which maps the service type to the service instance name. When querying for PTR resource records corresponding to a specific service type, all matching PTR records are returned, effectively listing service instances of a certain type. It is also possible to query for a super type that lists all available types, which in turn allows listing all available service instances. Typically, users select desired service instances among the listed ones, but an external automatic means for service selection could also be utilized. After selecting service instances, queries for SRV and TXT records corresponding to the desired service instances are sent, which provides service location. The SRV

---

12UPnP [PFKL08] queries for both devices (which implicitly maintain the directory) and services via multicast at the same time.

13Special means discussed in Chapter 3 reduce the number of necessary messages.
resource record contains the host name of the service provider and the port the
service instance listens on; the TXT record contains a service description in form of
key-value pairs. After retrieving the host name of the service provider, this host
name is resolved by querying for the corresponding A resource records, which yields
the IP address. Using the IP address and the port, a connection to the service
instance can be established. DNS-SD works with standard DNS as well as with
mDNS as a means for querying resource records.

Related Research. The Zeroconf stack itself is also a basis for research. We already
addressed related surveys in Section 2.2.

While typical applications for DNS-SD/mDNS are document sharing, synchronization,
device sharing, discovering printers, and inter-user communication, possible application
are not limited as every application that needs means for discovering services in the local
network may benefit. An example for an interesting uncommon usage are multi-screen
applications [BTS+13]. Choi et al. [CKK+14] use Zeroconf for device collaboration.
A discussion of Zeroconf with respect to implementations, performance and security is
provided in [SZKF12].

Various research has been done in the field of adapting DNS-SD/mDNS to the Internet
of Things (IoT), mainly making it applicable for resource constrained devices and mobile
ad-hoc networks. Krebs et al. [KKK08a, KKK08b] define a DNS-SD/mDNS based service
discovery architecture for wireless ad-hoc networks. They encapsulate mDNS messages in
OLSR (optimized link-state routing) [CJ03] messages. The architecture also allows non-
routing clients to engage in service discovery by connecting to a service proxy. Macker et al.
[MDTH10] provide a platform (and an implementation of their platform) for researching
service discovery paradigms, especially in dynamic mobile networks. Further, the authors
discuss various service discovery paradigms. The authors incorporate mDNS in their
platform in [MT11]. Klauck et al. [KK12] provide a case study of DNS-SD/mDNS for the
IoT and a DNS-SD/mDNS implementation for the IoT operating system contiki15. The
authors further propose message optimizations to adapt DNS-SD/mDNS for 6LowWPAN
networks [KK13]. Jara et al. [JMJS12] adapt DNS-SD/mDNS for web service discovery
in the IoT. Stolik et al. [SVCL14] deploy DNS-SD/mDNS proxies that handle queries on
behalf of resource constrained devices. Jeong et al. [LJP16] propose an IoT IPv6 name
resolution architecture based on DNS and mDNS. The architecture supports both local
and global name resolution. The authors provide a scheme for automatically generating
domain names for IoT devices; they also published an IETF Internet draft specifying
their naming scheme [JLP15]. Local names are provided via mDNS, while global names
are transmitted to a proxy (via mDNS) which then registers the respective name at an
authoritative name server via DNS update [VTRB97]. The authors further discuss the
application of their architecture in the context of smart cars and smart homes.

2.5.2 UPnP and SSDP

Like Zeroconf, UPnP [PFKL08] is a protocol stack where discovery — realized by the
simple service discovery protocol (SSDP) — is one of the layers. While UPnP covers
discovery, it focuses on the definition of interactions between clients and pre-specified
services. UPnP is based on HTML and SOAP [GHM+07], rendering it eminently verbose,

---

14By the time of writing this thesis, the current version of OLSR is defined in [CDJH14].
15http://www.contiki-os.org
and uses IP multicast as the enabling technique for configurationless discovery. Figure 2.6 (which is taken from the UPnP specification [PFKL08]) illustrates the layers of UPnP.

UPnP distinguishes between control points and (controlled) devices, which correspond to our clients and service providers, respectively. A single physical device can run both multiple logical devices and logical control points. A logical device may embed further logical devices. A logical device that is not embedded in another logical device is referred to as root device; logical devices that are embedded are referred to as embedded devices.

UPnP Steps. We shortly cover each step of UPnP, and then discuss the discovery step, i.e. SSDP, in a more detailed way.

Addressing: UPnP does not provide a means for address configuration. Devices and control points may either use a DHCP address or an automatic IP address using the Zeroconf stack’s technique [Sar04].

Discovery: The simple service discovery protocol (SSDP) allows devices to make root devices, embedded devices, and services discoverable for control points. The discoverable entities are advertised via IP multicast. Control points typically query via multicast, but may query via unicast, if they know the network parameters of the respective device. Answers are always sent via unicast. All of announcements, queries, and answers are realized as HTML messages that do not contain a body.

Description: Among other information, discovery messages contain a URL pointing to a description. In the description step, control points may send a HTTP GET request to retrieve this description. Each discoverable entity, root devices, embedded devices, and services, have such a description. A description is written by a UPnP vendor and templates are defined by the OCF (Open Connectivity Foundation, which swallowed the UPnP Forum, see below). These descriptions contain detailed information about discoverable entities. A device description contains, among other information, the manufacturer, the model’s name, the model’s serial number, and further, detailed information about the services offered by the device in form of a service list. The service list contains URLs for the next steps control and eventing, and further a list of state variables that represent the runtime state of a service.

Control: The interaction between devices and services is realized by SOAP [GHM+07], which allows accessing an API via the network in a thoroughly defined way. UPnP’s control can be seen as remote method invocation, which goes beyond the typical tasks of a service discovery system.

Eventing: From a layer perspective, the control and eventing steps are two mechanisms that work on the same layer. An event is triggered each time a runtime state variable,
which is managed in the service description, changes. The eventing mechanism allows control points to subscribe for receiving updates if certain variables change; further, devices may send multicast updates for certain variable changes. Subscribe and unsubscribe request messages are, like discovery messages, realized as body-less HTML messages. The Simple Network Management Protocol [CMRW93, Dab11] (SNMP) offers similar functionality and may be used by devices that implement the Zeroconf stack.

**Presentation:** Devices may provide a presentation URL in their descriptions. This URL may provide users with a web interface for control, and display the status of devices and their respective services.

UPnP does not provide means for name resolution. If users want to access devices via a non-UPnP service, they have to directly use the provided IP addresses. The UPnP specification [PFKL08] suggests registering names in the DNS if devices want to provide names.

**Simple Service Discovery Protocol (SSDP).** As service discovery is the focus of this thesis, we especially want to discuss the service discovery layer, which is handled by SSDP. Besides being specified in the UPnP specification [PFKL08], there has been an IETF Internet draft [ALG+99] proposing SSDP.

When a device joins a network, it announces its root devices, embedded devices, and services in HTTP header messages via IP multicast. These messages are referred to as discovery messages. Discovery messages use the special HTTP method NOTIFY. Each discovery message must contain (1) a notification type (NT), (2) a unique service name (USN), (3) a URL for further information, and (4) a duration for which the advertisement is valid. For each of these discoverable entities, dedicated discovery messages are published. A root device publishes three discovery messages: the first for announcing its root status, the second for announcing its UUID, and the third for announcing its uniform resource name (URN). Figure 2.7 shows example discovery messages announced by the root device of a miniupnpd16 test installation. Besides the messages from the root device, each embedded device publishes two discovery messages containing its UUID and its URN, respectively. Additionally, each service publishes one discovery message containing its URN.

To discover these entities, a control point can either passively listen to the announcement messages, or actively query by multicasting an HTTP message using the special method M-SEARCH. Figure 2.8 shows example queries. These messages can query for all discoverable UPnP entities, or for entities matching certain criteria. Devices answer these queries via unicast with HTTP OK messages whose content is almost the same as the content of NOTIFY messages17. Despite the name, M-SEARCH messages can also be sent via unicast if the control point already knows the network parameters of the desired device.

Generally, the protocol transmits a lot of information necessary to fulfill the specification and is very verbose. This is not a problem in small home networks, but makes UPnP scale very badly. Both devices and services are pre-specified by device vendors, which supports an easy integration but is inflexible. Figure 2.9 (cp. [PFKL08]) illustrates the interactions of SSDP.

16http://miniupnp.free.fr
17The only difference is that a ST (search target) header field is substituted for the NT header field.
2.5 Configurationless Service Discovery

HOST: 239.255.255.250:1900
CACHE-CONTROL: max-age=120
LOCATION: http://192.0.2.1:37659/rootDesc.xml
SERVER: Arch/rolling UPnP/1.1 MiniUPnPd/2.0
NT: upnp:rootdevice
USN: uuid:00000000-0000-0000-0000-000000000000::upnp:rootdevice
NTS: ssdp:alive
OPT: "http://schemas.upnp.org/upnp/1/0/"; ns=01
01-NLS: 1478533022
BOOTID.UPNP.ORG: 1478533022
CONFIGID.UPNP.ORG: 1337

(a) Root announcement.

HOST: 239.255.255.250:1900
CACHE-CONTROL: max-age=120
LOCATION: http://192.0.2.1:37659/rootDesc.xml
SERVER: Arch/rolling UPnP/1.1 MiniUPnPd/2.0
NT: urn:schemas-upnp-org:device:InternetGatewayDevice:1
USN: uuid:00000000-0000-0000-0000-000000000000::urn:schemas-upnp-org:device:InternetGatewayDevice:1
NTS: ssdp:alive
OPT: "http://schemas.upnp.org/upnp/1/0/"; ns=01
01-NLS: 1478533022
BOOTID.UPNP.ORG: 1478533022
CONFIGID.UPNP.ORG: 1337

(b) UUID announcement.

HOST: 239.255.255.250:1900
ST: ssdp:all
MAN: "ssdp:discover"
MX: 2

(a) Query for all discoverable UPnP entities.

HOST: 239.255.255.250:1900
ST: upnp:rootdevice
MAN: "ssdp:discover"
MX: 2

(b) Query for all root devices.

HOST: 239.255.255.250:1900
ST: urn:schemas-upnp-org:device:InternetGatewayDevice:1
MAN: "ssdp:discover"
MX: 2

(c) Query for all devices based on the Internet Gateway Device (IGD) template.

Figure 2.7: Announcement discovery messages published by the root device of a test miniupnpd installation.

Figure 2.8: UPnP example queries.
The UPnP Industry Consortium. Since the association of UPnP to its organization and to related projects might be a little confusing, we give a short summary. UPnP has been developed by the UPnP forum, which was swallowed by the OIC (Open Interconnect Consortium) in 2015. In February 2016, the OIC changed its name to OCF (Open Connectivity Foundation); at the same time, further big companies joined the consortium. The OCF develops standards for the Internet of Things (IoT) and uses the Constrained Application Protocol (CoAP, RFC 7252 [SHB14]) — an efficient application layer protocol for resource constrained devices on top of TCP/IP — as a basis. The OCF further continues the development of UPnP.

IoTivity is an open source framework for seamless IoT interconnectivity hosted by the Linux Foundation. It implements OCF standards and is sponsored by the OCF, but generally independent. IoTivity utilizes CoAP [SHB14].

AllSeen, an industry consortium which provided the AllJoyn open source IoT framework that addresses similar problems as UPnP, has been swallowed by the OCF in October 2016.

Related Work. UPnP is widely used in smart home systems. Innovative smart home network approaches utilizing UPnP include smart home service robots [BDLPÁM13] and brain computer interface integration [LLLC14].

Further, many attacks on home networks involving UPnP have been published. These attacks mainly rely on the control step of UPnP and are not an inherent problem of UPnP service discovery via SSDP. Among the most powerful of these attacks are attacks on the Internet Gateway Device (IGD) class of UPnP devices. These attacks involve unauthorized configuration changes [Hem06]. The author of this work also maintains

Figure 2.9: SSDP interactions (cp. [PFKL08]).
2.5 Configurationless Service Discovery

a webpage\(^{18}\) where he proposes and discusses UPnP hacks. Baugher and Lortz [BL11] discuss general risks and attacks in home networks with a special focus on UPnP. Kavalaris et al. [KS14, KS15] discuss vulnerabilities of multimedia solutions for Small Office Home Office (SOHO) environments. They focus on devices manufactured by Sonos, which use UPnP.

2.5.3 Comparing Zeroconf and UPnP

Both Zeroconf and UPnP are protocol stacks whose goal is providing a basis for user-friendly configurationless inter-device communication in local networks. Both stacks provide a configurationless service discovery method, focus on IP networks, and use IP multicast as enabling technique for achieving independence of configuration.

However, Zeroconf and UPnP pursue their basic goal quite differently; in fact, one could say they are orthogonal.\(^ {19}\) We discuss the differences and argue why we chose Zeroconf (DNS-SD/mDNS) as a basis for our service discovery framework.

**Complexity — Simple versus Complex.** The main reason for choosing Zeroconf over UPnP is its simplicity and efficiency — as far as multicast-based solutions can be considered efficient. Compared to UPnP, DNS-SD/mDNS announcement messages are simpler and less verbose. Most of the following arguments are linked to simplicity.

**Focus — Discovery versus Control.** While Zeroconf focuses on discovery, UPnP focuses on the interaction of clients (control points) with discovered services. The main contribution of this thesis — making configurationless service discovery privacy-preserving and applicable to multi-link networks — influences the actual discovery, which is fully covered by DNS-SD/mDNS. Interaction of clients with services is beyond the tasks of a discovery system and is orthogonal to our research.

**Service Application Layer Protocol — Agnostic versus SOAP.** For realizing control, UPnP uses SOAP. Every service discovered via UPnP is a service that has to be accessed via SOAP. Zeroconf can be used independent of the application layer protocol the discovered service uses. It can also be used for discovering SOAP services transmitting the control URL in the TXT record. Thus services discovered via Zeroconf may also benefit from using SOAP; we prefer to let application developers choose on a per service basis if these benefits justify the complexity and verbosity of SOAP. We strongly favor the flexible service protocol agnostic way.

**Discovery Protocol — DNS versus HTML.** While DNS-SD/mDNS is based on DNS, UPnP is based on HTTP. We prefer DNS because it is used as the main look-up service of the Internet. DNS-SD can also use the Internet’s DNS infrastructure, which allows for global discovery and helps the seamless integration of our solution in Chapter 7. Our solution would also allow leveraging DNS for scaling UPnP; but this approach does not allow seamless integration. Further, DNS is an actual resolution protocol while the original purpose of HTTP is transferring hyper text.

\(^{18}\)http://www.upnp-hacks.org

\(^{19}\)http://zeroconf.org/ZeroconfAndUPnP.html
Vendor Agnostic versus Vendor Oriented. While DNS-SD/mDNS is vendor and device agnostic, UPnP is not. UPnP vendors define devices and services instantiated from abstract device templates and service templates, respectively, which are defined by the OCF. UPnP’s focus on vendors and detailed pre-specification allows the seamless integration of pre-configured devices in home networks, but comes at the cost of high complexity, verbose, and inflexibility. The detailed specification further causes a pre-defined coupling between devices and control points, which again helps easy integration but also increases inflexibility. UPnP’s thorough service definitions might be feasible for vendors that design a product and then sell it in high numbers, but is not feasible for flexibly adding own services. Again our argument in favor of DNS-SD/mDNS is our focus on the discovery process instead of the description and selection. Further, our approach is user-centric, i.e. it focuses on users as acting entities instead of devices.

Inter-User versus Intra-User. UPnP mainly focuses on home networks and environments where the involved devices belong to the same user. This is not in small part due to the fact that UPnP strictly separates devices and control points, which fits the scenario of a single user controlling his devices. On the contrary, DNS-SD/mDNS is also widely used for offering services to other users. This is confirmed by a multicast protocol measurement study in a campus network [HSS09], where mDNS usage was significantly higher than UPnP usage. Since we do not only focus on home networks but also scale configurationless service discovery to multi-link networks, e.g. campus networks, we prefer DNS-SD/mDNS also in this point.

Deployment. Both DNS-SD/mDNS and UPnP are widely deployed and used. However, DNS-DS/mDNS is pre-installed on most of today’s smart systems, especially Android (NSD) and iOS (Bonjour), which are by far the most prominent systems for smart devices. While there are applications adding UPnP support to both Android and iOS, it is not supported out of the box.

Standard Setting Body — Truly Open versus Industry Consortium. All parts of the Zeroconf stack are defined in RFCs provided by the IETF (Internet Engineering Taskforce), which is independent of companies, has no form of membership, and everyone can participate by joining their meetings or the mailing list. UPnP is maintained by the OCF (Open Connectivity Foundation), which is an industry consortium.

### 2.5.4 Bluetooth and Bluetooth SDP

Bluetooth also provides a widely used service discovery protocol, which is referred to as Bluetooth service discovery protocol (Bluetooth SDP) [Blu14]. It is based on the Bluetooth protocol stack and thus is not IP-based. Being based on Bluetooth and not working in IP networks already makes it infeasible as basis for our service discovery framework; but, as it is widely used, we want to discuss it. Bluetooth SDP is a very simple (at least from a discovery point of view) request-response protocol, whose simplicity is sufficient because of the simple network topology. For this reason, we cover the Bluetooth connection establishment and the network topology before we discuss SDP.

**Connection Establishment.** Zeroconf and UPnP are based on UDP(TCP)/IP and provide no means for setting up the link and physical layers of the network, i.e. they
require devices to be attached to an existing network link. Bluetooth comprises all network layers and devices therefore first have to establish links. In analogy to IP-based solutions, the Bluetooth link establishment step can be compared to a WiFi device discovering and connecting to an access point.

Bluetooth devices may either be non-discoverable, or in one of the discoverable modes limited discoverable and general discoverable. While the first of the discoverable modes makes the device discoverable in a limited time frame or for a specific condition, the latter makes a device continuously discoverable.

A device may send an inquiry request, which makes this device a master device, and causes devices that receive this inquiry request to answer with their Bluetooth addresses. The Bluetooth address is an IEEE EUI-48 address [Ass], i.e. a unique 48-bit value whose first 24 bits represent a vendor. Users might further provide a name for a Bluetooth device, which is also transmitted. The master device selects an address among the received ones, which might be chosen by the master device’s user by selecting the associated name, and then engages in a paging phase during which the link is established. The roles of master and slave may be exchanged after connection establishment. Devices that are connected can perform a device pairing, which allows both secure communication (depending on the pairing method) and interaction-less re-connection at a later point in time. Secure pairing mechanisms involve further user interaction. We discuss the security aspects of establishing a Bluetooth pairing in Chapter 5. A thorough specification can be found in [Blu14].

**Link Layer.** Bluetooth devices establish links in a point-to-point or point-to-multipoint manner. The resulting network is referred to as piconet, comprised of a master device that initiated the connection and up to seven slave devices. Additionally, there can be up to 255 parked slaves which are slaves that are connected to the master but are not active on the channel. The master controls channel access. Active devices have a per-piconet logical 3-bit address. There are no collisions, as a master only talks to a single slave in a piconet at a given point in time. While masters can only be master in a single piconet, devices may be slave in several piconets using time-division multiplexing. Such a network topology is referred to as scatternet. Figure 2.10 illustrates the Bluetooth network topologies.
Since the goal of Bluetooth is setting up a wireless personal area network (WPAN) in a configurationless way (it was first designed as short range cable replacement), means for internetwork routing are not necessary; thus there is no layer that corresponds to the Internet’s IP layer. SDP works on top of the link layer, which multiplexes connections between two endpoints and further provides flow control and a reliable transport\textsuperscript{20}.

**Service Discovery.** While SDP would run over any reliable transport, it specifically addresses service discovery in the Bluetooth environment. Like Zeroconf, the goal of SDP is allowing clients to discover services offered by service providers, and not defining how clients utilize or select services.

SDP servers maintain service information, which is a list of service attributes, in service records. An SDP server can be seen as a daemon running on a Bluetooth enabled device; multiple applications on this device may utilize the SDP server. SDP clients communicate to the SDP server via a reserved channel; communication with the service is then transmitted via another channel. Bluetooth devices may implement both an SDP server and an SDP client. Further, both the *master* and the *slaves* can be both SDP server and SDP client simultaneously.

Since piconets have a simple star topology and may comprise 8 participants at most, service listing can be performed efficiently by just asking all available nodes. This corresponds to sending a unicast query to all available parts of the directory asking them for all their services, which is not feasible in larger networks. It is also possible to ask for certain services specified by a UUID. In a sense, the service directory has already been discovered during the link establishment phase.

### 2.5.5 Privacy Considerations

Zeroconf (DNS-SD/mDNS), UPnP (SSDP), and Bluetooth all breach the users’ privacy as they all publish private data. While both DNS-SD/mDNS and SSDP publish data via link-local IP multicast to all peers on the same network link, Bluetooth publishes data to all devices that are in close proximity\textsuperscript{21} and listen to Bluetooth traffic.

DNS-SD/mDNS publishes the host name, a descriptive service instance name, and possibly additional service information. We detail the privacy problems of DNS-SD/mDNS in Subsection 3.5.1.

The privacy problems of UPnP are very similar. UPnP neither (necessarily) multicasts a host name — name resolution is not a part of UPnP — or descriptive service instance names, but it publishes a lot of information about offered devices and services which renders tacking devices and inferring the device owners an easy task. The above discussed Figure 2.7 shows data multicast to all peers. In addition to the multicast data, UPnP provides a verbose description for each of its discoverable entities via the description URL which is also accessible by every peer in the network. An example of such a verbose description may be found in the UPnP specification [PFKL08] in Sections 2.3 and 2.5 for a device description and a service description, respectively.

Typically, Bluetooth transmits service related data via an encrypted connection. However, in many cases this connection does not provide real security (see Chapter 5) and may easily be overheard by malicious parties. Further, before establishing such a connection,\textsuperscript{20}There is also an unreliable transport mode of operation.\textsuperscript{21}The transmission range of Bluetooth for the typical class 2 devices is up to 10 meters [Blu14]. Class 3 devices may have a range of up to 100 meters when using the strongest permitted power.
Bluetooth devices publish their identifiers which makes them easily traceable, and further, may allow inferring and tracing the devices’ owners.

### 2.5.6 Multi-Link Considerations

The limited discoverability range of configurationless service discovery solutions is a disadvantage. While the previously discussed service discovery systems work well in single-link home networks, they do not scale very well to larger multi-link networks, e.g. a campus network. This is because these solutions are based on link-local IP multicast.

There are solutions that use link-local IP multicast when it is feasible, but provide means for scaling to larger networks. Typically, this scaling is realized by introducing a central directory [Gut99, ASW+99]; but this comes at the cost of losing the configurationless property. In this subsection we shortly cover three representatives of such service discovery systems.

In Chapter 7, we propose a method that allows scaling scopes while maintaining a configurationless mode of operation. While we propose our solutions as an extension to DNS-SD/mDNS, they are not dependent on the discovery system and work with any of the systems discussed in this chapter.

**SLP (Service Location Protocol).** SLP [VGPK97, GPVD99, GPK99, Gut02] is one of the first service discovery frameworks featuring configurationless service discovery in local networks. It scales to larger networks with an additional central service directory. While today Zeroconf, UPnP, and Bluetooth provide the mainly used service discovery protocols, we cover SLP as it not only provides a configurationless mode of operation, but also scales and introduces the concept of scopes.

SLP defines three roles for devices, user agents, server agents, and directory agents. A single device can have any subset of these roles. If there is no directory agent present, user agents query services via multicast and server agents that offer appropriate services directly answer via unicast. For local networks with a single multicast range, directory agents are not necessary. In larger networks comprised of several network links where multicast cannot cross, directory agents maintain service information. Service agents register services at directory agents, and user agents first discover discovery agents and then engage in service exploration. Thus, SLP features a dedicated directory discovery phase if directory agents are involved. The directory discovery phase may either be realized as manual configuration, or in a configurationless manner if the directory agent is reachable via multicast.

Discoverable services are described by a URL that contains a service type, the application layer protocol the service uses, and the host name of the service provider. Further, the description contains key-value pairs providing additional information about the service. SLP allows scopes that represent organizations, buildings, geographical areas or purposes. A set of services can be assigned to a scope.

We did not choose SLP as a basis for our solution, as it has been largely replaced by Zeroconf, and it depends on a central directory for scaling, which cannot be automatically discovered if it is not in multicast range. Further, the arguments in favor of DNS-SD/mDNS we stated above add to our decision of not choosing SLP.

**mDNS Hybrid Proxy.** Hybrid proxy [Che15] is an extension for DNS-SD/mDNS that allows scaling mDNS to multi-link networks without client-side configuration. A DNS
domain is assigned to each link of a network, which is delegated to a hybrid proxy that
is attached to the corresponding network link. A client can now query for services in
arbitrary links; this query process comprises (1) the querying for a link’s DNS domain, (2)
the delegation of this query to the link’s hybrid proxy, (3) the hybrid proxy’s multicasting
the query on its local link, and (4) forwarding the answer to the querying client via unicast.

This allows discovery of services via DNS-SD/mDNS that are offered in a network link
which would otherwise be unreachable via multicast. This approach is different from the
other approaches discussed in this subsection, as the entity introduced for crossing link
boundaries is not a directory node, but rather relays queries (the proxy may, however,
cache entries). The directory is still distributed among the clients and clients are agnostic
to the existing of the proxy; this maintains the configurationless user-experience for clients,
but both DNS entries and the proxies have to be set up.

UPnP entities that are in separate home networks via the Internet. All home networks
involved need a Remote Access Server (RAS) that is connected to the other network’s RAS
via a secure transport protocol, e.g. TLS. Establishing these tunnels is handled by the
Remote Access Transport Agent (RATA) component of the RAS. The network parameters
of RASes can be maintained in an Remote Access Application Server (RA AS).

Discovery across home network boundaries is handled by the Remote Access Discovery
Agent (RADA) component, which has both listener and relay functionalities. While the
listener is realized as a UPnP control point monitoring all SSDP messages in its local
network, the relay announces SSDP messages from the other network, which it receives
via a synchronization service that is fed by the listener of the other network. The RADA
relay will also locally answer SSDP queries for remote entities. Network administrators
can configure filters in the RASes to prevent certain entities from being discoverable in
select remote networks.

While UPnP RA works transparently for both devices and control points, the RASes
have to be set up. If an RA AS is used, it also has to be set up and maintained. Further,
in contrast to mDNS hybrid proxy, information about remote entities is pushed, which
increases the network load.

Jini. Jini [ASW+99] (now also known as Apache River22) is a Java based service discovery
system. Services are realized as Java objects, and clients can interact with services via
Java RMI (remote method invocation). Thus, like UPnP, the Jini framework does not
only specify service discovery but also client-service interaction.

The Jini directory is managed by a look-up service running on dedicated directory
nodes. Jini supports a central topology, but as directories might proxy requests to other
directory nodes, more complex topologies can be built. Setup and maintenance of these
directory nodes demands administration. Jini features a separate directory discovery
phase, which can either be realized by manually configuring the directory on clients, or
via multicast. Multicast discovery provides a configurationless experience for clients. Via
multicast, only directory nodes in the same multicast range may be discovered. But these
directory nodes can relay queries similar to DNS-SD/mDNS hybrid proxy.

We did not choose Jini as a basis as it depends on a single platform, Java, and needs
configuration for each directory node.

22http://river.apache.org
2.5 Configurationless Service Discovery

**Multicast Routing.** As mentioned before, link-local IP multicast, which is the basis for the afore discussed service discovery solutions, is confined to a single network link. Packets with a destination address in the link-local multicast range, $224 \cdot 0 \cdot 0 \cdot 0/24$ [CVM10] and $ffx2::/16$ [HD06, Dro14] for IPv4 and IPv6, respectively, must not be routed.

IP multicast [Dee89] per se is not limited to a single link. The link-local address range is only a small portion of the whole multicast IP range which is $224 \cdot 0 \cdot 0 \cdot 0/4$ [CVM10] (former Class D) and $ff00::/8$ [HD06] for IPv4 and IPv6, respectively.

Besides the link-local address ranges, there are further ranges, which, amongst others, determine routing and multicast IP address allocation. For IPv4, these ranges are defined in RFC 5771 [CVM10]. The definition of the IPv6 address ranges and a description of the integrated multicast scoping method may be found in RFC 4291 [HD06].

Within a subnet, hosts join a multicast group, which is represented by a multicast IP address, using IGMP [HCH06] and MLD [VC04] (a sub-protocol of ICMPv6 [BGTP07]) for IPv4 and IPv6, respectively. Hosts that have joined a multicast group receive packets sent to the corresponding multicast IP address; the mDNS multicast group, for instance, is represented by $224 \cdot 0 \cdot 0 \cdot 251$. For delivering multicast packets in a subnet — instead of using ARP [Pos83] — a multicast link-layer address is derived dynamically from the multicast IP address as described in RFC 1112 [Dee89] Section 6.4.

For efficiently propagating multicast packets between routers, the PIM (Protocol Independent Multicast) multicast routing protocol family has been developed. While PIM dense mode (PIM-DM) uses flooding and subsequent pruning, PIM spare mode (PIM-SM) reduces network load by building a multicast tree in bottom-up manner, and only forwarding forwarding multicast packets to subtrees accommodating interested hosts. This tree is built by propagating multicast group membership information from local routers (which learned this information via IGMP or MLD), to a rendezvous point, which is a chosen router, associated with the multicast group. The rendezvous point acts as root for distributing multicast packets associated with its group. Routing from a rendezvous point to subnets containing interested hosts is referred to as reverse path forwarding, because packets are routed from a source rather then towards a certain destination.

Mbone (Multicast Backbone) [Eri94] realizes global scale IP multicast by using IP multicast within routing domains and tunneling via unicast for inter-domain transmission of multicast packets.\(^{23}\) RFC 7450 [Bum15] defines a protocol for automatic multicast tunneling (AMT).

Nevertheless, using multicast-based service discovery solutions over multi-link multicast or global scale multicast is not feasible as (almost) every network participant is expected to be in the service discovery multicast group and acting as both multicaster and receiver of all other multicasts. While each host had to be a member of the service discovery multicast group for receiving information about services it is interested in, the vast majority of packets transmitted in this group would contain information the host is not interested in. This scenario is comparable — both with respect to usability and privacy — to a global telephone network were every participant hears everybody else talking, but wants to talk to a few specific persons. Direct connections, which would mitigate the problem — as in a real world telephone network — are not possible because in configurationless service discovery hosts do not have prior knowledge about network parameters of service providers (in the telephone example such prior knowledge would be the phone number). Multicast-based

\(^{23}\)The name *mboned* refers to both the classical Mbone daemon which acts as tunnel entry and exit point, and the IETF multicast backbone development working group ([https://datatracker.ietf.org/wg/mboned/about](https://datatracker.ietf.org/wg/mboned/about)), in which many of the IP multicast related RFCs were developed.
service discovery in multi-link networks would both cause an unmanageable network load and yield a huge list of service instances (most of them irrelevant) a user would have to choose from. Further, multicast is especially inefficient if the access networks are wireless, e.g. IEEE 802.11 [80212], and if there is a large number of access networks that are part of the same discovery scope, which we discuss and analyze in Chapter 4 and Chapter 6.

Multi-link IP multicast is mainly used and most efficient for applications such as multimedia streaming and software updates, where a few sources send packets to a large number of interested hosts.

Another approach is scaling the concept of subnets to multi-link networks; arising issues have been discussed in [Tha07].

2.5.7 Interoperability and Bridging

Further open problems exist in the field of service discovery solution interoperability. Mechanisms that bridge between two or more service discovery systems have been researched but are not (yet) widely used. There is a US patent [Eyt10] describing a solution to bridge DNS-SD and UPnP. Allard et al. [ACGRI03] propose an architecture for bridging Jini and UPnP.

A framework that allows interoperability of several service discovery protocols — DNS-SD/mdDNS, UPnP, SDP, and Jini — and allows plugging in further protocols via an interface is proposed by Verslype et al. [VNV+09]. The framework uses a generalized service description. It also allows scaling these solutions beyond the range of multicast by leveraging UPnP-RA.

Nelis et al. [NVVD12] discuss service discovery protocols and interoperability issues. They further provide a middleware that provides interoperability of arbitrary service discovery protocols by unifying the model used by various service discovery protocols and providing similar services with the same semantics across these protocols. The generalization provided by their middleware is geared to UPnP (e.g. defining generic service types).

2.6 Conclusion and Prospects

Service discovery plays a crucial role in today’s networks as it allows for efficient and dynamic connection establishment. It provides efficient means for discovering service directories and exploring services. We gave a broad overview over different areas of service discovery discussing surveys. Further, we discussed the components that comprise service discovery systems as well as their interactions. We dived deeper into the area of configurationless service discovery solutions for local networks, which as of yet do not protect the user’s privacy and, if even scalable to multi-link networks, lose their configurationless property — the very reason these solutions are so comfortable to use.

We pointed out to the fact that both privacy and configurationless multi-link scalability should be addressed. While the solutions this thesis presents may be integrated in any service discovery solution, it is important to us to thoroughly design an integration into one of the widely used service discovery solutions for both showing the feasibility of the solution and directly providing its benefits to users in an easy-to-deploy way.

We chose DNS service discovery over multicast DNS (DNS-SD/mdDNS), which is part of the Zeroconf stack, for the aforementioned reasons. DNS-SD/mdDNS provides a general and simple means for configurationless service discovery in local networks, while UPnP —
which often is wrongly considered a competing technique — provides a set of thoroughly
defined vertical solutions for specific purposes, and further is complex and verbose.
DNS Service Discovery

3

Contents

3.1 Introduction ......................................................... 50
3.2 Related Work ....................................................... 51
3.3 DNS ................................................................. 51
3.4 Multicast DNS (mDNS) ............................................. 64
3.5 DNS Service Discovery over Multicast DNS .................. 69
3.6 Conclusion and Prospects ......................................... 73

DNS Service Discovery over Multicast DNS (DNS-SD/mDNS), made fashionable through Apple’s Bonjour, is a prevalent technique for configurationless service discovery in local networks. Possible application areas are device synchronization, instant messaging, VoIP, collaboration, file and screen sharing. It is very convenient for users, because they can connect to and offer services when they enter a network without any manual configuration. But DNS-SD/mDNS suffers from problems inherent in multicast-based service discovery solutions: insufficient privacy, confinement to a single network link, and a significant network load.

This chapter (1) discusses DNS-SD/mDNS, (2) details its enabling techniques DNS and mDNS, and (3) argues why we chose DNS-SD/mDNS as a basis for the realization of our service discovery framework (the concepts of our solution are independent of DNS-SD/mDNS). Since the realization of our service discovery framework is based on DNS-SD/mDNS, we want to properly introduce this technique bestowing it a dedicated chapter in this thesis. The chapter further (4) discusses the afore mentioned problems using DNS-SD/mDNS as an example, not because it is especially weak, but rather because we consider it worthwhile mending its problems.
3.1 Introduction

DNS Service Discovery over Multicast DNS (DNS-SD/mDNS) is a prevalent technique widely used for offering and requesting services in local networks without configuration. Using the upper two layers of the Zeroconf stack [SC06], namely DNS Service Discovery [CK13a] and Multicast DNS [CK13b], it provides a great user experience. For example it allows a student who enters her campus network to automatically connect her mobile devices (smartphone, tablet, notebook) to each other, allowing file sharing and synchronization; it further allows automatically connecting to friend’s devices on campus, communicating or sharing data, and connecting to infrastructure devices like printers. The discovery works without user interaction and regardless of the IP addresses and ports the corresponding services use.

We chose DNS-SD/mDNS as a basis for realizing our service discovery framework for the following reasons.

- It is relatively simple yet still efficient — as far as multicast-based solutions can be referred to as being efficient.
- DNS-SD/mDNS is widely used. It runs on Linux (Avahi), Windows (Avahi, Bonjour), MacOS (Bonjour), Android (NSD), and iOS (Bonjour). With the exception of Windows, all systems come with a DNS-SD/mDNS implementation out of the box; from a user’s perspective it just works. Implementations for Internet of Things (IoT) operating systems, such as contiki\(^1\), also exist.
- DNS-SD/mDNS is based on DNS which is the lookup service in the Internet and thus predestined for service discovery.
- DNS lookup functionality is ubiquitous. Libraries providing DNS lookup functionality exist for almost every language. Almost every device has a client for DNS lookups.
- Cutting edge service discovery solutions offer DNS syntax for service discovery, e.g. docker\(^2\), skydns\(^3\), and Amazon Route53\(^4\).
- It is defined in IETF RFCs which are open standards; development is not directly dependent on industry consortia, and everybody can contribute and join the meetings.

We provided a comparison of DNS-SD/mDNS and UPnP, often mistaken as direct competitors, in Chapter 2. Even though we use DNS-SD/mDNS as basis for our service discovery framework, it does not depend on DNS-SD/mDNS and can be adapted to work with other techniques (see Chapter 4).

Since Zeroconf distributes information via cleartext link-local multicast messages, it compromises the users’ privacy and does not work across multicast boundaries. We generally discussed problems inherent in multicast-based service discovery solutions in Chapter 1; we discuss these problems specifically with respect to DNS-SD/mDNS in this chapter.

In the following sections we

- provide an overview over DNS,
- explain mDNS and DNS-SD,
- show how these techniques work together,
- discuss the inherent problems (privacy and being confined to a single network link),
- and discuss requirements that have to be met for mending these problems.

\(^1\)http://www.contiki-os.org \(^2\)https://www.docker.com \(^3\)https://github.com/skynetservices/skydns \(^4\)https://aws.amazon.com/route53
3.2 Related Work

The main references for this chapter are the RFCs specifying DNS [PT87, Moc87b], DNS-SD [CK13a] and mDNS [CK13b]. There are many more RFCs specifying further important aspects of these techniques, which we introduce in the course of this chapter. Besides the specifying RFCs, there is research work analyzing and adapting these techniques. We addressed this research in Chapter 2, specifically in Section 2.2 and Subsection 2.5.1.

3.3 DNS

The Domain Name System (DNS) [Moc87a, Moc87b] is one of the most crucial parts of the infrastructure of today’s Internet. Besides its main task of mapping domain names to IP addresses, the DNS allows mapping domain names to various kinds of data, providing means for various other services, among them host aliasing and service discovery. Since DNS service discovery is an important basis for the realization of our service discovery framework, this section explains the basics of DNS and further parts of DNS that are important for this thesis. Since privacy is a crucial property of our service discovery framework, we also discuss security and privacy features of DNS. Not only our privacy extension builds on DNS techniques, but also our means for multi-link service discovery (see Chapter 7 and Chapter 8).

Basically, the DNS is a hierarchical lookup system that allows querying for complex data types associated with a domain name. These data types are referred to as resource records. The most common resource record is the A resource record which maps a domain name to an IP address. For example, the query example.net A asks the DNS server for the A resource record associated with the domain name example.net, which might yield the answer example.net A 192.0.2.137.

Before we go into more detail, we want to discuss the components DNS is comprised of. While typically the components of DNS are seen as the dedicated entities of a DNS implementation, client, authoritative server (master and slaves), and recursive server, we first list the logical components and discuss them subsequently.

**Domain Name:** A domain name is a hierarchical name for an entity, which can e.g. be a host or a service. For example, the domain name of a server might be example.net, while the name of the web server running on this server might be www.example.net.

Looking at the DNS as a key-value store, domain names correspond to the keys.

**Resource Record:** A resource record (RR) is data associated with a type and a domain name. Looking at the DNS as a key-value store, resource records correspond to the values.

**Protocol:** The DNS protocol is an application layer protocol running on UDP/IP. It provides a means for the communication between a client and a DNS server.

**Database:** The hierarchical database maintains resource records and can be queried for these using the DNS protocol. The entity that is responsible for one or several hierarchy levels of one or several domain names is referred to as authoritative server.

---

5While the DNS has been originally defined in RFC 1034 and RFC 1035 (older documents exist but are obsolete), many more RFCs exist that update and extend the original definition. When explaining certain aspects of DNS, we cite the corresponding RFCs. An up-to-date comprehensive list of DNS-related RFCs is provided on http://www.statdns.com/rfc.

6RFC 2181 [EB97] Section 11 clarifies and explicitly states that DNS cannot only be used as a mapping system for IP addresses but also in a more general way.
While in the classical DNS these components are inseparable, these components do not necessarily have to be used in combination. mDNS, which we also discuss in this section, does not use the hierarchical DNS database (the database is maintained by the service providers themselves) and defines a multicast-based protocol that is similar to the standard one, for querying DNS resource records. However, many aspects in all the components have been specifically made to support the hierarchical unicast-based structure.

### 3.3.1 Domain Names

Domain names identify entities in the DNS hierarchy. Each such entity can be seen as a node in a tree data structure and is associated with a label (e.g. example). The (fully qualified) domain name of a node is comprised of this node’s label concatenated with this node’s ancestors’ labels (from left to right); the labels are separated by a dot (‘.’). The example www.example.net specifies an entity whose label is www, whose parent’s label is example, whose parent’s label in turn is net. The parent of net is ‘.’ which represents the root of the DNS hierarchy. Since the root is unique, in most cases the dot is not necessary when referring to a domain name; it is sometimes necessary to avoid confusions with relative domain names, i.e. domain names that are relative to a certain origin. For example, a node with the relative domain name test.www and the origin example.net, is associated with the fully qualified domain name test.www.example.net. The Bind9\(^7\) zone file syntax uses relative names and origin names; if an origin is specified, domain names may be abbreviated substituting @ for the origin. Analogous to Bind9, we substitute @ for the origin if the origin is obvious in the respective context. Domain names that are a direct child of the root are referred to as top level domains (TLDs). Figure 3.1 illustrates the DNS name hierarchy. In this thesis we use fully qualified domain names unless explicitly introducing relative domain names, which allows us not writing the final root-representing dot in the remainder of this thesis avoiding confusion with other punctuation. RFC 2181 [EB97], which removes restrictions from the original definition in RFC 1035 [Moc87b] yielding the following definition.

- A domain name is comprised of one or several labels separated by a dot (‘.’).

\(^7\)Bind9 is the most commonly used DNS server implementation. [http://www.bind9.org](http://www.bind9.org)
3.3 DNS

![Diagram of DNS hierarchy]

**Figure 3.2:** Example domain name hierarchy including local domains. While each local network shares the same global name space, they each have a dedicated local name space. `myprinter.local` in one local network will very likely be resolved to an IP address belonging to another host in another local network.

- A label may contain up to 63 octets of arbitrary data (an octet is comprised of 8 bits).
- A domain name consists of a maximum of 255 octets including the separating dot.

The format of domain names specified in RFC 1035 [Moc87b] restricted the octets to case-insensitive alphanumeric strings including hyphens (and the dot separating labels). While most of today’s domain names still heed these restrictions, they are not necessary.

**Local and Special Use Domain Names.** Both the afore described logical components specifying an architecture and the single instance of the architecture that is deployed in the Internet are referred to as DNS. Deploying a separate architecture instance with its own domain name space and root is possible. Domain names of the resulting local DNS name space are local, but what is typically referred to as local domain names, are domain names of the Internet’s domain space, which are designated for local use. Despite the fact that such domain names are in the global hierarchy, resolving them yields results dependent on the current local network and may be different in another local network. Such a local-use domain is `.local`. There are further domain names that are reserved for special use which are defined in RFC 2606 [EP99] and RFC 6761 [CK13c].

**Global Domain Names.** Names that are part of the Internet’s global name space are referred to as global domain names. Similar to the global IP addresses, global domain names are a limited resource and have to be registered with an authority. Top level domains are registered at the root; child domains of TLDs, which make up the highest hierarchy level that typically may be registered by end users, are registered at the respective TLD’s authority. Global TLDs are e.g. `org`, `com`, `net`, and the country specific TLDs, e.g. `de`, `ch`. Figure 3.2 illustrates global and local domain names.
3.3.2 Resource Records

The data items that the DNS associates with domain names are referred to as resource records (RR).

Generic Data Fields. Before discussing the most common resource record types we show the general structure of DNS resource records, which is specified in RFC 1035 [Moc87b].

- **NAME**: the fully qualified domain name of the entity the resource record belongs to. It is sometimes referred to as owner name.
- **TYPE**: a two octets wide identifier of the records type (e.g. 1 for A).
- **CLASS**: a two octets wide class identifier. The class specifies for which kind of network the corresponding record is meant. We will only focus on the most common class IN, which is the Internet class. When we do not specify the class, we always assume the IN class.
- **TTL**: the time to life, i.e. the time for which the resource record is valid. The TTL determines how long resource records should be cached. A TTL of 0 prevents caching.
- **RDLENGTH**: a 2 octets field specifying the length of the following RDATA field.
- **RDATA**: the actual resource record specific data.

Common Resource Record Types. In the following we provide a short description of the most common resource records, and the resource records we use in this thesis; a comprehensive description of the RDATA fields may be found in the RFCs we cite along the RR descriptions or in RFC 1035 [Moc87b] if there is no specific RFC cited. We discuss additional records for DNS security in the corresponding subsection.

- **A**: as already shown in the example, it maps a domain name to an IPv4 address.
- **AAAA**: maps a domain name to an IPv6 address.
- **SOA**: specifies the start of authority for a DNS zone, meaning it specifies who is responsible for a certain subtree of the DNS hierarchy. The node associated with the NAME field of this RR is the root of the corresponding subtree. For specifying the responsibility, the SOA RR contains the domain name of the primary name server for this zone, and the email address of the responsible person (which might be an organization). The SOA record also contains a zone version number, which is incremented each time the zone is updated. It contains further fields specific to zone transfers, namely the time interval before which the zone should be refreshed, the time interval that should be waited before retrying to refresh after a failed refresh attempt, and the maximum time until expiry. Within a zone there can be further SOA records specifying sub-zones.
- **NS**: maps a domain name to a name server that is responsible for that domain name.
- **CNAME**: maps a domain name to a canonical domain name, which then in turn can be queried. This is mainly used for domain name aliasing.
- **DNAME**: maps all child domain names of its owner name to child domains with corresponding names of a target domain name. It might e.g. map *example.net to *example.xy. This RR is specified in RFC 6672 [RW12].
3.3 DNS

**MX:** specifies the *domain name* of the mail exchange server associated with the *owner name*.

**PTR:** maps a *domain name* to another *domain name* indicating the fact that the first *domain name* points to the latter. Typically, it is used for specifying a reverse mapping from an IP address to a *domain name*, where the *owner name* of the PTR RR is a special domain name encoding the IP address as labels under the special domains `in-addr.arpa` and `ip6.arpa` for IPv4 and IPv6, respectively. DNS-SD uses PTR resource records for the purpose of service listing, which we explain in Subsection 3.3.6.

**SRV:** allows general purpose service location and is not bound to a specific protocol and does not assume specific ports (unlike e.g. the MX resource record, which is specific for the mail exchange service). It is defined in RFC 2782 [GVE00] and is utilized by DNS-SD (see Subsection 3.3.6).

**TXT:** contains character strings. It is e.g. used for DNS-SD to provide additional information about service instances in form of key-value pairs.

All of CNAME, DNAME, and PTR contain the exact same RDATA — a single *domain name*. They differ just in the record type which indicates the semantic meaning of the RDATA.

**Resource Record Sets.** A set of resource records that have a common *owner name*, a common *class*, and a common *RR type* is referred to as resource record set (RR set). RR sets are defined in RFC 2181 [EB97].

### 3.3.3 The DNS Protocol

The DNS protocol specifies how a DNS client can query a DNS resolver for resource records of specified types corresponding to a specific *owner name*. It is specified in RFC 1035 [Moc87b]. A single type of message is responsible for all involved communication. Typically, messages from the client to the server are referred to as *queries* and messages from the server to the client are referred to as *answers*. A message is comprised of the five following sections.

**Header:** This section contains a *message ID* for correlating associated query and answer messages, the number of items in the following sections, and further header values that we discuss when needed alongside descriptions in this chapter.

**Question:** This section contains the query, which comprises the *domain name* (QNAME), the type (QTYPE), and the class (QCLASS) of the desired resource records. A response message contains the same question section as the corresponding query. The type can be any resource records type or one of the special types ANY, which yields all\(^8\) RRs associated with the queried *domain name*, AXFR which queries for a whole zone, and IXFR which queries for an incremental zone transfer, i.e. for the delta between RRs associated with the last retrieved zone’s version number (see SOA record) and RRs associated with the current zone’s version number.

**Answer:** This section contains a list of resource records that match the query.

** Authority:** This section contains NS resource records of name servers that are authoritative for this query.

---

\(^8\)In practice, an ANY query may only retrieve a subset of matching RRs. Often, this subset comprises the matching RRs which the queried server has in its cache.
Additional: This section contains resource records that might help, but do not directly answer the question. Typically, it contains the A resource records corresponding to the name servers contained in the authority section, which are commonly referred to as glue records.

As the DNS protocol and the hierarchical DNS architecture are intertwined in a way that certain flags only make sense if a corresponding architecture component is present, we will detail these parts of the protocol alongside the explanation in the next subsection. A list of all protocol parameters including a comprehensive list of resource records is provided by the IANA\textsuperscript{9}. IANA considerations concerning DNS are presented in RFC 6195\textsuperscript{[3rd11]}.

### 3.3.4 The Hierarchical DNS Architecture

While the afore discussed components of DNS would also work with a single central DNS server,\textsuperscript{10} a centralized approach would not scale. In this subsection we give an overview over the hierarchical DNS architecture, as it is used in the Internet. We detail aspects of the hierarchical DNS database which are necessary for our Stateless DNS Technique in Chapter 8.

**Authoritative Server.** An authoritative server is a DNS server that has the authority to answer DNS queries for resource records associated with domain names in it’s zone, i.e. in the subtree of the DNS hierarchy it is responsible for. It must be specified in the zone’s SOA record and maintains a database containing the resource records associated with it’s zone. Typically, this database is simply realized as a text file (or a set of text files), which is what the most commonly used DNS server implementation Bind9\textsuperscript{11} does.

For load balancing and availability reasons, there might be several authoritative servers for a single zone; only one of these is specified in the SOA record. Typically, the authoritative server specified in the SOA record acts as master server and the others as slave servers; the zone is administered at the master server and the slave servers automatically transfer the zone from the master server using the afore mentioned special query type IXFR.

When sending an authoritative answer, an authoritative server sets the AA (authoritative answer) flag in the answer message. More specifically, the AA flag in a message indicates that the answering server is authoritative for entries in the answer section whose owner name corresponds to the query domain name, or for the first entry.

**Resolver.** The client\textsuperscript{12} in the DNS infrastructure is referred to as resolver. A resolver sends queries to a DNS server and receives the corresponding answer. There are two kinds of queries, iterative and recursive; we explain the former here and the latter when explaining the recursive server. With just a resolver and a hierarchy of authoritative servers, only iterative queries are possible.

For retrieving resource records associated with a certain domain name, the resolver first queries one of the root name servers. The IP addresses of these root name servers have to be provided to the resolver, e.g. in form of a text file, to make the resolver capable

\textsuperscript{9}http://www.iana.org/assignments/dns-parameters
\textsuperscript{10}In this case, SOA and NS resource records would be superfluous.
\textsuperscript{11}http://www.bind9.org
\textsuperscript{12}DNS servers may also act as clients, either when asking for a zone transfer, or when executing a recursive query.
of performing iterative queries. This root server information is typically referred to as root hints. When confronted with a query for a domain name referring to a node that is deeper in the hierarchy, e.g. a query for an A resource record associated with www.example.net, the root name server will most likely not know the answer. If it knew all answers, the hierarchy would be superfluous. Instead of a direct answer, the root server will provide the resolver with an NS resource record associated with the domain name, which e.g. contains ns1.net\(^{13}\). Such an NS answer is referred to as delegation because answering the query is delegated to a name server further down in the hierarchy. The answer should also contain a glue record (in the additional section of the message), which is an A resource record that provides the IP address of the name server. A delegation without a glue is sometimes referred to as a glueless delegation, which demands the resolver to explicitly query for this A resource record. If the answer to this query again just contains the same NS delegation, the desired domain name is rendered unresolvable. When receiving the A resource records of the next name server in the hierarchy, the process recurs as the resolver sends the same query to this name server and might either get the desired answer, or a further delegation. In our example, the answer might be a further NS resource record containing ns1.example.net, which, when asked the same query, might finally yield the desired A resource record for www.example.net. While a single query is not iterative per se, these queries are referred to as iterative as an iterative process is involved for retrieving resource records associated with a domain name. Figure 3.3 illustrates the iterative query process.

The resolver might retrieve several A resource records. In this case client software should use them starting with the first in the list as this is a means for load balancing: the DNS server may return a list of A resource records in random order to balance the load between the servers that have the corresponding IP addresses.

Recursive Server. A recursive server may take over the iterative querying. Utilizing a recursive server, a resolver sends query messages to the recursive server instead of to the root server. The recursive server iteratively queries the DNS hierarchy as described above and returns the desired answer to the resolver. From the resolvers perspective it looks like a single query was sufficient for retrieving the desired information (see Figure 3.4).

\(^{13}\)The actual domain names of the name servers for the net zone are [a-z].gtld-servers.net. We chose this example name for simplicity’s sake.
A resolver requests recursive handling of a query by setting the RD (recursion desired) flag in the query. A DNS server sets the RA (recursion available) flag in the answer if it supports recursive querying. The recursive server functionality may be implemented separately or integrated in an authoritative server. Such an integrated server will also set the RA flag in its answers; it will only set the AA flag if the answer originates from its own zone database.

**DNS Cache.** While the DNS would theoretically work using the components described so far, it would be very inefficient, as every query would be sent to one of the root servers; either directly by the resolver, or indirectly by a recursive server. To overcome these problems, both resolver and recursive server may implement a DNS cache. The DNS cache can also be realized as a separate component (DNS cache server) and be asked like any other DNS server; it will replay matching cached answers and appropriate cached delegations. Especially the cache of recursive servers may save a lot of upstream messages as all hosts using the same recursive server benefit from this cache. Depending on the caching rules, the cache of a recursive server might store arbitrary resource records including the delegations it learned about while executing recursive queries. If our example query for an A resource record for www.example.net is followed by a query for a TXT resource record for info.example.net, the DNS cache might already have the corresponding delegation to ns1.example.net including the glue record, and thus does not have to ask the root server and then the name server for example.net, but can directly ask ns1.example.net.
Figure 3.6: Query process involving a forwarder and caches. The figure further shows an example of where these components might be maintained.

Figure 3.5 illustrates a query process involving a DNS cache. However, caching every received record is a security risk. The caching rules are not strictly defined in an RFC and depend on the implementation; we discuss these rules in Chapter 8 as they directly affect our Stateless DNS technique.

Stub Resolver. A resolver that only sends recursive queries is referred to as a stub resolver. In case of a non-validating stub resolver, it further does not perform DNSSEC validation, which we address in Subsection 3.3.7. Instead, it treats answers that have the AD (authenticated answer) flag set as authenticated; thus entrusting the recursive server with the validation. Stub resolvers are used as DNS resolver daemons by end user operating systems.

Forwarding Server. A forwarding server is a very simple DNS server, that just forwards queries to an recursive server. Forwarding servers may be used in home routers; they forward queries originating in the home network to the ISP’s recursive servers. They can be combined with a DNS cache for saving the exchange of messages with the respective recursive server. Figure 3.6 illustrates involving a forwarder and caches in the query process and further shows an example of where the components might be maintained.

3.3.5 EDNS

The extension mechanisms for DNS (EDNS), defined in RFC 6891 [DGV13], allow adding additional flags and header items to the DNS while being fully backwards compatible. It works by introducing a pseudo resource record type OPT, which in both query and answer is transmitted in the additional section of the DNS message. OPT resource records must not be defined in the resource record data base of authoritative DNS servers. Further, they do not have an owner name. Their RDATA field contains an option code, a field specifying the option data length and the option data field; the option data field is a bit field containing the additional flags the corresponding extension defines.

The most prominent use case of EDNS is DNSSEC, which we address is Subsection 3.3.7.
3.3.6 DNS Service Discovery (DNS-SD)

DNS Service Discovery, defined in RFC 6763 [CK13a], allows discovery of arbitrary services using the DNS. Often, the DNS is used to resolve the domain name of a web server, typically indicated by www as the most specific label of the domain name, e.g. www.example.net. Because a web server is expected to listen on port 80 (for HTTP), a client may connect to the retrieved IP address on port 80. A similar expectation about the ports and the application layer protocol is involved when using MX resource records for resolving mail exchange servers. A means for discovering arbitrary services is provided by the SRV record [GVE00], which contains both domain name of the service provider (the host that offers the service) and the port on which the service listens. DNS-SD utilizes the existing resource record types PTR, SRV, and TXT, for the purpose of service listing, service location, and service description, respectively.

Domain Names. DNS-SD uses the concept of service types and service instances, which correspond to the concept of classes and instances in object oriented programming. A service type is either associated with TCP, if its instances run over the TCP transport layer protocol [Pos81], or associated with UDP, if its instances run over any other transport layer protocol, e.g. UDP [Pos80] or SCTP [Ste07].

DNS-SD defines structured domain names for service instances. Each node in the DNS tree may spawn two subtrees containing service instances; one for service instances associated with TCP, and one for service instances associated with UDP. Each service type is realized as a child of either the TCP or the UDP node, and each service instance is in turn a child node of its service type’s node. For example, if a host wants to provide an ssh service with the name "example-ssh" under the domain name example.net, the domain name of this service instance would be example-ssh._ssh._tcp.example.net.

The <Instance> portion of the service instance name, which is the left-most label of the domain name, is a DNS label containing arbitrary Net-Unicode [KP08]; the service type’s name is preceded by an underscore and the protocol specifying label is either _tcp or _udp. The service type’s name and the protocol part of the domain name are referred to as <Service> portion of the service instance name; the remaining labels on the right are referred to as the <Domain> portion. This yields the definition (RFC 6763 [CK13a] Section 4.1):

Service Instance Name = <Instance> . <Service> . <Domain>

Resource Records. For service listing, DNS-SD uses PTR records that point to instance names of available service instances of a service type. The PTR record’s owner name corresponds to a service type node in the DNS. In our example this is _ssh._tcp.example.net. The PTR record

_ssh._tcp.example.net IN PTR example-ssh._ssh._tcp.example.net

conveys the information that there is a service instance of the name "example-ssh" (in the Internet class) of the type _ssh._tcp which belongs to example.net. The domain example.net might offer several service instances of the type _ssh._tcp, which would be represented by PTR records with the same owner name. When a resolver issues the query

14The classical SRV record definition does not mention the use of the concept of service instances, and thus does not integrate them in example domain names.
3.3 DNS

PTR _ssh._tcp.example.net the responsible authoritative name server will reply with all associated PTR resource records effectively listing all available service instances of the desired type. Further, DNS-SD specifies a special meta domain name whose PTR resource records point to all available service types under a name: _services._dns-sd._udp.<domain>. For our example, the answer would be

_services._dns-sd._udp.example.net IN PTR _ssh._tcp.example.net.

For service location, which is performed during the service resolution phase, DNS-SD uses SRV resource records (RFC 2782 [GVE00]). These SRV records map a service instance domain name to the domain name of the service provider offering the corresponding service instance and the port this service instance is listening on. The SRV record associated with our example service might be

example-ssh._ssh._tcp.example.net IN SRV 0 0 13337 somehost.example.net.

This conveys the information that the ssh service instance with the name "example-ssh" runs on a host with the domain name somehost.example.net using the port 13337 (DNS-SD allows automatically configuring non-standard ports on the client side). The two zeros are for specifying priorities (similar to the priority fields in MX records) if there were several service instances with the same name, e.g. for load balancing, which is very rarely the case.

For a service description during the service resolution phase DNS TXT resource records are used; these describing TXT resource records must be provided and should contain a single zero byte if the corresponding service needs no further description. DNS-SD thoroughly specifies the format of these TXT records. Generally, TXT records as defined in RFC 1035 [Moc87b] allow arbitrary character strings of 255 bytes preceded by a one-byte length field. Simplifying, the definition of DNS-SD TXT resource records boils down to a list of strings (compliant with RFC 1035) containing either key-value pairs of the form "<key>=<value>", or just "<key>"; the latter is interpreted as a boolean in a sense that a certain attribute is true, e.g. "needs password" might state the fact that a certain service needs a password. The TXT record associated with our example service might look as follows.

example-ssh._ssh._tcp.example.net IN TXT
ForwardX11=no
PasswordAuthentication=no

This TXT record tells clients that both X11 forwarding and password authentication are inactive. This record just serves as an example and is not necessary; the service provider may freely change the TXT record (as long as it heeds the format rules).

Providing and Querying for Services. To provide a service via DNS-SD, a host has to provide the corresponding PTR, SRV, and TXT service records; e.g. by adding them to a zone file of an authoritative name server. When asked for an SRV record, an authoritative server should provide the A resource record corresponding to the host name in the SRV record in the additional section of an answer message (similar to the afore mentioned glue records for NS records).

The query process can be subdivided into three phases, service browsing, service resolution, and name resolution. During service browsing, clients ask for the PTR records
Figure 3.7: Discovering a service using DNS-SD, assuming the A resource record is not already known and not sent alongside the SRV record. The first pair of query and answer shows the process of service browsing using PTR records. The second pair of query and answer represents service resolving using SRV and TXT records.

3.3.7 DNS Security

The original DNS specification does not provide any security mechanisms; thus DNS yields several security and privacy related problems [Bor15]. Especially prominent is the privacy
problem resulting from all DNS messages being transmitted in clear text, which allows for pervasive monitoring [FT14]. In this subsection we explain various techniques addressing DNS security problems.

**DNSSEC.** The domain name system security extensions (DNSSEC) provide a means for resolvers to verify the authenticity of received DNS records. DNSSEC does not provide confidentiality or privacy. The core of DNSSEC is defined in RFC 4033 [AAL^05a], RFC 4034 [AAL^05c], and RFC 4035 [AAL^05b]; there are further RFCs specifying different aspects of DNSSEC.

DNSSEC provides signatures for resource records, which are part of a trust chain from the root to the respective node in the DNS. Simplified, the root servers sign keys for the TLD servers which the TLD servers use to sign their authoritative RR sets; the TLD servers further use this key to sign a key for the next level of servers in the hierarchy; these servers again use the keys for signing their authoritative RR sets and for signing the keys for the next level. To ease key updates, the key that is signed by the parent zone is not actually used to sign the resource records but to sign a further key; these keys are referred to as key signing key and zone signing key, respectively.

We give a short overview over the most important resource records introduced by DNSSEC; a thorough definition of the respective RDATA is provided in RFC 4034 [AAL^05c].

**DNSKEY:** contains the public key that shall be used for verifying further resource records in the same zone.

**RRSIG:** records are associated with each resource record whose authenticity should be verifiable. They contain the signatures.

**DS:** refers to a DNSKEY record in a child zone, and contains a hash of the child zone’s public key. The DS (delegation signer) is the means for establishing the transitive trust relationship. The key authentication process is described thoroughly in RFC 4035 [AAL^05b].

**NSEC:** contains the next owner name of an authoritative RR set in the current zone in the canonical zone ordering, and the (other) types of resource records that are associated with its owner name. The canonical ordering of a zone means ordering all domain names that are part of the zone in alphabetic order regardless of their actual level in the hierarchy (see RFC 4034 [AAL^05c] Section 6.2 for the exact definition). The main purpose of the NSEC records is allowing authenticating the non-existence of a resource record.

**NSEC3:** serves the same purpose as the NSEC resource record but aims to deny zone enumeration. To this end it uses hashes of the owner names instead of just the owner names. Attacks that still allow (partial) zone enumeration exist.

The DNSKEY and DS resource records are associated with a zone (like the SOA resource record), which might be comprised of several levels in the DNS hierarchy.

**DANE.** DNS-based Authentication of Named Entities (DANE) leverages DNSSEC providing a means for authenticating the association of a domain name with X.509 certificates [CSF^08]. This allows using the DNSSEC trust chain for authenticating the certificate instead of using a certificate authority (CA).

DANE is specified in RFC 6698 [HS12], which proposes DANE in conjunction with the application of authenticating certificates for TLS [DR08], which is referred to as
DANE introduces the new resource record type TLSA which contains certificate authentication data. The RRSIG record associated with the TLSA records allows verifying the authenticity of the certificate authentication data, and thus the authenticity of the certificate.

DANE can also be used for verifying connection endpoints specified in SRV resource records by providing means for discovering the related TLSA records, which is defined in RFC 7673 [FMSA15].

DNS also allows storing X.509 certificates, which is defined in RFC 4398 [Jos06]. The introduced CERT resource records are different from TLSA resource records in that they provide whole certificates instead of data which identifies a certificate. Further, the purpose of CERT records is providing certificates via the DNS, while DANE allows authenticating certificates that were retrieved by a different means.

**DNS Privacy.** DNS does not provide confidentiality and thus no privacy, even when using DNSSEC. A solution for the privacy problem of transmitting all messages in clear text is provided by DNS over TLS, specified in RFC 7858 [HZH+16] (DNS message transmission over DTLS is described in [RWP16]). While DNS over TLS mitigates pervasive monitoring of the network traffic, a recursive server, e.g. the recursive resolver of an ISP, can still monitor all queries it receives.

While DNS over TLS leverages the transport layer security, DNScurve\(^{15}\) [Dem10] directly integrates confidentiality into the DNS protocol. The public keys of authoritative servers, which are prefixed by a magic string, are encoded in name server names, i.e. in the target of NS resource records. Resolvers send their public keys in query messages. All DNS messages are encrypted using the respective keys. DNScurve can work together with DNSSEC. Like DNS over TLS, DNScurve protects messages between a resolver and server, which mitigates pervasive monitoring but still allows recursive servers and caches to monitor queries.

### 3.4 Multicast DNS (mDNS)

Multicast DNS (mDNS), defined in RFC 6762 [CK13b] allows the configurationless distribution of DNS resource records in local networks via link-local multicast. Resource records that can be offered and queried via mDNS have to be under the special-use top level domain .local. Each host may claim a subdomain of .local on a first-come-first-serve basis without having to register the domain. All mDNS messages are sent via multicast to the multicast address 224.0.0.251 (or to its IPv6 equivalent FF02::FB, respectively). Hosts that receive a multicast query for a resource they provide send the corresponding answers to the same multicast IP address. In contrast to the standard DNS where names are globally available, subdomains of .local are only unique within the current subnet. As link-local multicast is the enabling technique for communicating the resource records, the scope in which link-local multicast messages are propagated determines the range in which these names are resolvable. Typically the range is a network link; it might, however, be confined to an access network of an access point, or even non existent as multicast is sometimes deactivated due to efficiency issues. Figure 3.8 illustrates the mDNS query process.

\(^{15}\)https://dnscurve.org
3.4 Multicast DNS (mDNS)

As IPv4 and IPv6 links are independent, they also have their separate .local domains. Dual-stack hosts may be part of both of these .local domains.

3.4.1 Distinction from DNS

Multicast DNS differs from the standard DNS in several points. Before we discuss specific mechanisms of the mDNS protocol, we will shortly cover the peculiarities of mDNS.

- Hosts provide their resource records themselves (see local topology in Chapter 2).
- While the mDNS name space is still hierarchical, the database is not. Since there is no DNS server hierarchy involved, there are neither NS resource records nor SOA resource records.
- Query messages might have a non-empty answer section, which contains answers to the query the querying host already knows.
- Query messages, if carrying a probe query, have an authority section, which is used for probe tiebreaking (see Section 8.2 of RFC 6762 [CK13b]).
- Query messages use one bit of the QCLASS field to signal the wish for a unicast response.
- Answer messages do not have to repeat the query in the query section, which is due to the fact that mDNS does not correlate queries and answers. For the same reason, the ID header field is empty.
• Answer messages for queries with the type ANY must contain all matching resource records.\textsuperscript{16}

• Answer messages may use one bit of RRCLASS field for signaling that the corresponding resource record should be used to overwrite a possible existing one in the same RR set from the caches, which is referred to as cache flush. A cache flush should only be performed for unique records, which are resource records whose RR set only contains this single record.

• Answer messages use a zero TTL to trigger the deletion of the corresponding resource records from the caches. This is the means that mDNS uses for deregistering resource records associated with mDNS domain names.

• Answer messages contain AAAA and A resource records in their additional sections when queried for A and AAAA records, respectively (if available).

Further (minor and technical) differences are listed in a comprehensive list of differences in RFC 6762 [CK13b] Section 19. We will discuss further differences while explaining specific features of mDNS.

### 3.4.2 Providing Resource Records

Since there are no DNS servers involved in mDNS, and the knowledge about the local zone is distributed among peers, hosts that wish to provide resource records for subdomains of the .local have to do so themselves. Instead of registering a domain and providing associated resource records via a DNS server, hosts provide new resource records via multicast. The process of providing resource records consists of the two phases probing and announcing, which hosts perform whenever they either join a network (also when waking from sleep mode) or change their network parameters.

**Probing.** During the probing phase, a host checks whether the resource records for its desired domain already exist. This is done by multicasting a query for the desired domain name, which is referred to as probe query. Since the underlying transport layer protocol is not reliable (UDP) and it is essential for avoiding collisions that all peers receive the probe message, the probe query is send thrice after 250 ms intervals. To avoid bursts, e.g. after a network outage, hosts wait a randomly chosen time between 0 ms and 250 ms before sending the first probe query. If a peer already claimed the desired domain name it will defend it by announcing it after receiving the probe query; the defending announcement is sent immediately. Probe queries should be sent with the QU bit set asking for a unicast answer. If the probing host does not receive a message within 250 ms after the last probe query, it proceeds to the announcing phase. The probe query should use the qtype ANY for claiming exclusive ownership for a domain name. Handling of special cases where two peers start probing for the same domain name at the same time (probe tiebreaking), or where the defending announcement is not received in time, is proposed in Section 8.2 of RFC 6762 [CK13b]. To allow probe tiebreaking, probe queries list unique resource records the host wishes to announce in the additional section of the query message.

\textsuperscript{16}Many recursive DNS servers return just a subset of the matching RRs comprised of the RRs the queried server has in its cache.
3.4 Multicast DNS (mDNS)

![Diagram of mDNS cache state]

**Figure 3.9:** Cache state after the query process illustrated in Figure 3.8. Each host caches received multicast answers, even if a peer sent the corresponding query.

**Announcing.** When no conflicts were detected during probing, the host *announces* the corresponding resource records by multicasting an unsolicited answer message that contains the desired resource records in the answer section. For *unique* resource records the aforementioned cache-flush bit must be set. Like queries, answers should be delayed to avoid bursts blocking the network; the timings are detailed in Section 6 of RFC 6762 [CK13b].

### 3.4.3 Query Messages

The mDNS query process distinguishes between *one-shot queries* and *continuous queries*. The former is multicast once, and answers that are received in a certain time frame after sending the query are considered by the querier. The latter seems like a subscribe mechanism from a users point of view, as the query is multicast in exponentially growing time intervals,\(^{17}\) which provides a relatively up-to-date list of available resource records matching the query. Further, query messages are sent for resource records whose TTLs are about to run out. The exact timings for sending these queries can be found in RFC 6762 [CK13b] Section 5.2.

\(^{17}\)The growth stops at a maximum time interval of one hour.
Receiving multicast answers when sending multicast questions has the benefit of updating the caches of peers and keeping an up-to-date state of knowledge over the .local domain (see Figure 3.9). However, in certain situations it is better to provide the querier with a unicast answer for saving network bandwidth,\(^{18}\) which can be requested by setting the QU bit in the query message. RFC 6762 [CK13b] suggests to utilize this for the queries a host sends when joining the network; for any subsequent queries the QU bit should not be set because known-answer suppression, which we explain in the following, kicks in. For special purposes, direct unicast queries might be sent to port 5353, which must be answered via unicast; the responder must check that the query came from the same network link. Rules for the caching of received answers are discussed and proposed in Section 10 of RFC 6762 [CK13b].

3.4.4 Answer Messages

Answer messages are sent upon receiving a query by hosts that are authoritative for the requested resource records. In the context of mDNS a host is authoritative for RR sets it claimed during the above described process of probing and announcing. Hosts must not answer messages using RR sets from their caches. When answering queries with query type ANY, all resource records matching the query have to included in the answer. Sensible TTL values are discussed and proposed in Section 10 of RFC 6762 [CK13b].

3.4.5 Mechanisms for Reducing the Network Load

Since multicast has a significant impact on the network load, mDNS proposes several mechanisms of reducing the number and size of multicast messages.

Known Answer Suppression demands that queriers include already known answers in the answer sections of their query messages. This only applies for non-unique RRs, as in the case of a unique RR the query would be unnecessary if the answer is already known. Records whose remaining TTL is less than 50% of the original value should not be included as the responder would answer anyway.

Duplicate Question Suppression prevents a host from sending a query that has just been multicast by a peer if the answer section contains only RRs the host already knows and thus would also suppress.

Duplicate Answer Suppression prevents responders from multicasting answers that just have been multicast by a peer. This might happen in the presence of proxies.

Besides these mechanisms, queriers should send multiple queries in a single message, and responders should send as many answers in one message as possible; to this end, queries and answers should be queued for a short time frame and all queries and answers should then be sent in one message, respectively. Queries should stop querying in intervals for unique records upon receiving them until the remaining TTL drops below 50% of its original value.

\(^{18}\)We argue that providing service information, which we discuss in the next section, only to hosts that are authorized to use the respective service increases both privacy and efficiency. We address this matter in Chapter 6.
3.5 DNS Service Discovery over Multicast DNS

While DNS-SD (RFC 6763 [CK13a]) and mDNS (RFC 6762 [CK13b]) are fully independent techniques, they are companion techniques mainly utilized in combination for realizing configurationless service discovery in local networks. These techniques work seamlessly together. DNS-SD over mDNS (DNS-SD/mDNS) brings great user experience as services can be discovered automatically both via sending multicast queries and listening to announcements.

Since mDNS mainly uses multicast for transmitting messages, privacy and network load problems arise. When using DNS-SD over multicast, both problems are significantly amplified as DNS-SD resource records contain much more private data and the number of multicast messages increases for communicating the additional records. For this reason and because mDNS is mainly used in conjunction with DNS-SD, we discuss both the privacy and efficiency problems in this section.

3.5.1 Query Process and Privacy Considerations

In this subsection we describe the process of a host querying for a desired service type. We point out the privacy problems arising in each phase of DNS-SD querying and illustrate these problems using the example of Alice querying for the _presence service type, which is a configurationless chat service used e.g. by iChat and Pidgin\(^{19}\).

In many cases a separate name resolution phase is not necessary as the querying host might have received the record during the announcement phase of the corresponding peers, or during a previous query process. We address the phases separately to clearly point out which problems arise during which phase, and which problems arise when using mDNS stand-alone for name resolution. Figure 3.10 illustrates the messages sent during the query process described in this section.

When a host learns about a certain service instance by passively listening to the announcements or responses for other queries, its privacy with respect to this particular service is not breached, meaning its interest in a certain service instance is not made public. The private information leaking on the service provider side is independent of how peers learn about the service related information.

**Service Browsing.** During the service browsing phase, hosts ask for PTR records, which are used as indicator for the existence of an instance of a certain service type (see first pair of query and response in Figure 3.10).

When Alice is asking for the service type _presence, she sends a PTR query with the label _presence._tcp.local. All hosts providing a service instance of type _presence multicast a response containing information that Figure 3.11(a) shows.\(^{20}\)

There are two privacy problems arising during the browsing phase. Firstly, the name and type of the service instance can be seen by anyone in the same local network. Secondly, any peer may browse for a service type and cause service instance information to be multicast, even if the host is not authorized to connect to these service instances.

\(^{19}\)https://pidgin.im
\(^{20}\)The figures containing resource records show the subset of the information which is relevant for the purpose of illustrating the privacy problems.
Figure 3.10: Discovering a service using DNS-SD/mDNS assuming the A resource record is not already known and not sent alongside the SRV record (cp. Figure 3.7).

In our example, anyone using the _presence chat service sees Alice coming online. This is not a constructed example; most operating systems choose a default host name which contains the username, and chat applications using the _presence service use the pattern `<username>@<hostname>` as chat alias. This problem is more severe than it might look at first glance; everyone in the same network can see everyone else coming online, even users that are not in the buddy list. It is not even necessary to use sniffing tools like Wireshark\(^\text{21}\); using a simple service browser or a chat client like Pidgin suffices.

**Service Resolution.** During the *service resolution* phase, hosts retrieve the service location information and the service description, by querying for SRV and TXT, respectively (see second pair of query and response in Figure 3.10).

A host who has browsed for existing services can use the instance names gained from the PTR records to ask for the corresponding SRV and TXT records. The peer in the network who offers the requested service instance will answer by multicasting the requested SRV and TXT records.

\(^{21}\text{https://www.wireshark.org}\)
3.5 DNS Service Discovery over Multicast DNS

 présence._tcp.local: type PTR,
 alice@Alice’s Notebook._presence._tcp.local
(a) PTR record containing the service instance name. In the case of the _presence service type, it typically contains the user name and the device’s host name.

 alice@Alice’s Notebook._presence._tcp.local: type SRV,
 port 5298, target Alice’s Notebook.local
(b) SRV record showing the port and the host name which by default often contains the real name of the user as operating systems ask for it during setup.

 alice@Alice’s Notebook._presence._tcp.local: type TXT,
 vc=! ver=2.10.6 node=libpurple
 port.p2pj=5298 txtvers=1
 status=gaming
 last=Wonderland
 1st=Alice
(c) TXT record containing several privacy breaching key-value pairs like the first and last name of the user, the chat status and the version of the service.

 Alice’s Notebook.local: type A, addr 192.0.2.10
(d) A record presenting a mapping of the host name to the IP address.

 Alice’s Notebook.local: type HINFO
 CPU X86_64, OS LINUX
(e) HINFO record providing information about the host device.

Figure 3.11: Resource records multicast when using a chat application that is based on the _presence service. Each of these resource records violates privacy.

Service resolving yields another privacy problem. The SRV record publishes both the host name of the service provider and the port number the provided service instance uses. Since the port number is often a synonym for a service type, e.g. 22 for ssh, the port number is critical as well. The published port number may also yield a security problem; though not relevant for our chat example, it is a problem for protected services, allowing attackers to attack the service without the need of a port scan, which would render them suspicious on a network monitoring tool. The TXT resource record contains further information about the offered service instance in form of arbitrary key-value pairs. These may contain a significant amount of private information accessible by anyone in the same network. Further, since the owner name of both the SRV and the TXT records matches the PTR record’s target, the information published during browsing — service instance name and service type — is published again.

Example answers for the instance name

 alice@Alice’s Notebook._presence._tcp.local

are shown in Figures 3.11(b) and 3.11(c), respectively. The _presence service transmits first and last name in the TXT record, if they were entered during setup. A further problem
arises due to the version number sent in the TXT record, allowing identifying hosts running a vulnerable version of a particular service in order to attack them.

**Name Resolution.** During the *name resolution* phase, which is not specific to DNS-SD, hosts ask for A and AAAA records, respectively, to retrieve the IP address corresponding to the host name gained from the SRV record (see third pair of query and response in Figure 3.10).

Once again the host name is publicly announced. Due to publishing the IP address, it is possible to get a mapping from host name to IP address allowing inferring further information about a host. Even without offering any service instances the Zeroconf daemon immediately offers an A (and AAAA) resource record when joining a network. Depending on the configuration, there is also an HINFO (host information) resource record published, which also contains the host name plus information about the host’s CPU and operating system. Figures 3.11(d) and 3.11(e) show an example answer for the types A and HINFO, respectively.

This thesis addresses the privacy problems arising when using mDNS and DNS-SD/mDNS proposing a privacy extension in Chapter 6.

### 3.5.2 Multi-Link Considerations

Multicast DNS (and thus DNS-SD/mDNS) messages do not cross network link boundaries which confines the discoverability of .local domains to a single network link. This is very well suited for configurationless service discovery; however, it renders configurationless service discovery infeasible in multi-link networks, e.g. in a campus network or in the network of a larger conference.

A simple solution that just forwards mDNS messages across links would be infeasible as the number of multicast messages would bring the network to a halt (see Chapter 4). Besides the network load problem, a .local domain containing a very large number of service records would render manual selection of desired results infeasible. Either a means for automatic service selection or a means for structuring the search space — which can very well be based on the hierarchical structure of DNS names — is necessary. Further, the afore discussed privacy problems become more severe as private information is published in a wider scope.

Introducing an appropriately designed *discovery scope* concept to mDNS would provide a solution for multi-link mDNS and helps in mitigating these problems. Confining discovery to a scope which is defined by a domain label instead of by a link of the physical network is much more flexible and less restrictive for users. Depending on the distribution of scopes over the multi-link network, multicast messages will be reduced. Where appropriate, scopes can be defined to cover a single link, e.g. the network link of a working group. Scopes also confine the range in which private information is published. Further, scopes structure the search space and thus help the manual selection of service instances.

However, introducing such scopes to mDNS while maintaining a configurationless mode of operation and further being privacy-preserving is a challenge. We provide a scope extension for DNS-SD/mDNS that provides a configurationless mode of operation in Chapter 7; additionally, our privacy extension (Chapter 6) can be used in conjunction with the scope extension for providing private service instances in multi-link networks.

RFC 7558 [LCBM15] discusses the problems arising when scaling DNS-SD/mDNS to multi-link networks and states requirements a feasible solution should meet. The Internet
3.6 Conclusion and Prospects

draft [Che15] proposes DNS-SD hybrid proxies, a solution that relies on registering name server delegations at authoritative DNS servers. We discussed DNS-SD hybrid proxies in Chapter 2.

3.5.3 Network Load Considerations

Compared to unicast, multicast consumes more overall network bandwidth because the messages are delivered to all hosts in the network. Within a single link, multicast messages are also delivered to hosts that are not part of the corresponding multicast group.\footnote{Switches featuring IGMP snooping [CKS06] only forward multicast messages on ports with hosts that are members of the corresponding multicast group.} these hosts then filter the messages at their network interfaces. Especially in half-duplex shared media such as WiFi, the impact of multicast on the network load is severe as multicasts have to be sent with the lowest supported transmission rate so that devices only supporting the lowest transmission rate also receive the multicasts. Further, multicasts block the medium of all access networks of a network link \( \text{e.g. the access networks of WiFi access points that belong to the same network link)} \). Hong et al. [HSS09] performed an empirical analysis at a campus network which showed that 13\% of the network bandwidth is used by DNS-SD/mDNS. We compare the network utilization of multicast and unicast in half-duplex shared media in Chapter 4, which illustrates that multicast-based solutions do not scale.

Analysis of the Network Impact on Small Networks. The impact of DNS-SD over mDNS on WiFi networks with up to 25 hosts has been simulated and analyzed in detail in a Master thesis supervised by the author of this thesis [Rai15]. The analysis shows that while DNS-SD/mDNS does not scale to large networks, it is perfectly feasible for small networks as the impact on small networks is negligible. In the context of this Master thesis, INET/OMNeT++ models for DNS and mDNS were developed [RKW15].

3.6 Conclusion and Prospects

DNS is the resolution and look-up service in the Internet and as such is omnipresent. It does not only support name resolution but also various other resolution mechanisms, among them service discovery. In addition to the global resolution functionality provided by DNS, mDNS — which is also widely used — allows for configurationless resolution in local networks. Combining DNS-SD and mDNS, DNS-SD/mDNS allows for user-friendly configurationless service discovery in local networks.

However, DNS-SD is neither privacy-preserving, nor does it work in multi-link networks. We address these problems in an abstract and general way — not only DNS-SD/mDNS suffers from these problems as they are inherent in multicast-based service discovery solutions — in the remainder of this thesis. Since new mechanisms profit from the wide deployment of DNS if they can be seamlessly integrated with DNS, and further, DNS is a mature and stable system, we realize our solutions as seamlessly integrated extensions for DNS-SD/mDNS.
A Secure, Scalable, and User-Friendly Service Discovery Framework

Contents

4.1 Introduction ................................................. 76
4.2 Related Work ............................................... 77
4.3 Abstract Concepts .......................................... 77
4.4 Integration and Architecture ............................... 82
4.5 User Interface .............................................. 84
4.6 Implementation ............................................ 92
4.7 Network Impact ............................................ 92
4.8 Conclusion and Future Work ............................... 94

Configurationless service discovery based on link-local multicast offers great user experience. However, it comes at the price of publishing private data, being confined to a single network link, and causing considerable network load.

We provide a practicable and easy-to-use service discovery framework addressing these three disadvantages. While we apply the realization of our enhancements on DNS-SD, they can be applied to any service discovery system.

This chapter (1) discusses the abstract concepts for enhancing configurationless service discovery and shows how our ideas can be be applied to arbitrary service discovery systems. We (2) provide an overview over the components of our service discovery framework and show how these components can be integrated into a framework providing a seamless user experience. We (3) describe our service discovery framework from a user’s point of view illustrated by screenshots, and show that although the underlying techniques were challenging to develop, from a user’s perspective the framework is very simple. Further, we (4) analyze the effects of multicast on the network load and show how our components reduce the network load by avoiding multicast where possible.
4.1 Introduction

While existing solutions for configurationless multicast-based service discovery are convenient for users, these solutions neither protect the users’ privacy nor work beyond the restrictions of multicast. As we motivated in the introduction of this thesis, these problems are severe; providing solutions raises configurationless service discovery from a convenient easy-to-use technique to a technique that can be recommended with a quiet conscience and even works in large institutional networks where typically multicast would restrict its applicability.

The solution has to cover each of the components identified in Chapter 1 while being user-friendly. In this chapter we give an overview over the components of our solution and discuss their seamless integration. Our service discovery framework protects the user’s privacy, allows service discovery in arbitrary scopes which may comprise multiple network links, and provides an application that returns control to the user. While sensible defaults allow configurationless privacy-preserving service discovery, users are both free to choose whom to share services with and free to choose in which scope services are offered and requested. It allows selectively offering services to chosen friends, chosen groups, or everyone in a scope.

Figure 4.1 classifies service discovery techniques with respect to the area a scope may cover and the privacy they offer. We combined existing techniques but also developed new techniques for providing solutions for the yet missing parts. We payed special attention to user-friendliness which manifests mainly in providing a configurationless mode of operation where feasible. Solutions for single-link local networks and the whole Internet are given by DNS-SD/mDNS and DNS-SD/DNS, respectively. These techniques combined do not cover the whole desired area as they neither provide privacy nor provide configurationless operation on multi-link networks. Providing a configurationless solution for multi-link...
networks, we developed DNS-SD over Stateless DNS, which we detail in Chapter 7 and Chapter 8. Providing privacy protection we developed both a pairing component and a privacy-preserving service discovery component that leverages the pairing component for establishing secure authenticated connections.

This chapter

- discusses abstract concepts for enhancing configurationless service discovery and points out how the ideas of our solutions can be be applied to arbitrary service discovery systems;
- provides an overview over the components of our service discovery framework, which are detailed in Chapter 5, Chapter 6, and Chapter 7;
- shows how these components can be integrated into a framework providing a seamless user experience;
- illustrates the workflow of using our service discovery framework from a users point of view;
- analyzes the effects of multicast on the network load and points out how our solutions reduce network load by avoiding multicast where possible; and
- provides an overview over our implementations.

All those concepts work together seamlessly and are transparent to both applications and the network.

4.2 Related Work

We address most of the related work either in one of the following chapters if it is especially related to one of the components of our framework, or in Chapter 2 if it is related to the general area of configurationless service discovery. As most of the service discovery framework related work is also relevant for a general overview over service discovery, we cover it in Chapter 2.

Directly related to this chapter is the work by Mian et al. [MBB09], which discusses service discovery frameworks in the context of multi-hop mobile ad-hoc networks. But the discussed frameworks do not feature a privacy component, which is the focus of our work.

Another research topic that is directly related to this chapter is efficiency and bandwidth usage of multicast in general and especially in 802.11 wireless networks. The efficiency problem of multicast in wireless networks is explained in [VTLAY14]. Chow et al. [CM05] introduce mechanisms to mitigate the problem. Hong et al. [HSS09] measure the impact of multicast DNS on a campus network.

4.3 Abstract Concepts

We present a general overview over the components of our privacy-preserving service discovery framework. The overview is general in a sense that it is not dependent on a concrete service discovery solution. It both provides readers with the basic concepts and shows that our components can be incorporated in any service discovery framework. It further shows that our main components — pairing, privacy-preserving service discovery, and multi-link & scopes — are independent and could each be separately replaced by another realization. Further, each of these components is again divided in independent layers, which also can be separately replaced by other realizations.
Figure 4.2: Layers of our device pairing solution. (1) The devices that are about to be paired discover each other and establish a connection. A user interaction is necessary for privacy protection. (2) The devices exchange the pairing data in a confidential way. (3) The users of the devices manually authenticate the pairing data, e.g. by comparing a short authentication string displayed on both devices.

4.3.1 Pairing

Any privacy-preserving communication demands that the communicating parties can exchange messages in a way that grants confidentiality, integrity, and authenticity. This, in turn, demands that the communicating parties have prior knowledge about each other because authenticated key material is necessary. A privacy-preserving service discovery solution must protect messages containing information about users, devices, or services, granting all of confidentiality, integrity and authenticity. Parties not knowing each other can protect the confidentiality by using opportunistic encryption (see e.g. [Duk14]), but authenticity is not granted. A means for establishing a relationship between devices by exchanging authenticated key material is a must for privacy-preserving service discovery. While this key material could be provided by a central service, e.g. a certificate authority, this would be incompatible with our goals as it would demand configuration.

For our service discovery framework, we establish the relationship necessary for privacy-preserving communication via manually authenticated device pairing. While this demands minimal user interaction, it does not require maintaining a central service. Further, we provide a purely configurationless mode of operation in Chapter 5. Our solutions received the consensus of the IETF dnssd working group, also with regard to the fact that a
4.3 Abstract Concepts

minimal user interaction is acceptable for the sake of privacy, as our Internet drafts on privacy-preserving DNS-SD/mDNS and on a manually authenticated device pairing mechanism were adopted ([HK16b, HK16a]).

Our device pairing component establishes a relationship between devices by agreeing on a mutually authenticated shared secret. Since the pairing step has to be performed only once at an arbitrary point in time before the first discovery engagement, and since the only requirement the other components have on the pairing component is the establishment of the mutually authenticated shared secret, the device pairing component is inherently independent of the service discovery related components; thus it automatically works with arbitrary pairing solutions. Our pairing component is subdivided in discovery, pairing data exchange, and manual authentication; Figure 4.2 illustrates these layers.

Discovery. Before starting the pairing process, devices have to establish a connection, which in turn demands that they discover each other. Mainly, we use DNS-SD/mDNS or QR codes for the discovery phase of pairing. Both discovery methods need a user interaction. For DNS-SD/mDNS we need the user interaction to allow privacy-preserving discovery without prior knowledge (see Chapter 5). Utilizing DNS-SD/mDNS for the discovery phase of pairing does not make paired devices dependent on DNS-SD/mDNS for privacy-preserving service discovery at a later point in time. Further, we support connecting both devices via a cable and via Bluetooth as discovery methods. The goal of the discovery phase is establishing a connection between devices that are about to be paired.

Pairing Data Exchange. During the pairing data exchange phase, devices agree on a shared secret and may exchange further pairing related data. The secret is agreed upon via a Diffie-Hellman key exchange [DH76]. The protocol we use is based on the improved Hoepman protocol [NR11b], which both protects against offline attacks and generates a short authentication string (SAS), which is handed to the verification phase. Besides a theoretical protocol, we provide a protocol that can be efficiently and securely reduced to practice as we heavily rely on TLS [Res16].

Verification. During the verification phase, the users of the pairing devices check whether the SASes generated on the devices match. There are several methods for realizing this verification, which we discuss in Chapter 5. Given a secure pairing data exchange protocol like the improved Hoepman protocol, the manual SAS verification provides authentication of the exchanged data.

4.3.2 Privacy-Preserving Service Discovery

As a basis for efficient privacy-preserving service discovery we divide service discovery in two stages, namely directory discovery and service exploration (see Figure 4.3). Both stages demand an existing device pairing, which can be realized as described above and is detailed in Chapter 5. Further, our solution requires a local service directory topology, meaning each service provider maintains the part of the service directory that contains information about its own services.
Figure 4.3: Layers of our privacy-preserving service discovery solution. Neither directory discovery nor service exploration involve user interaction, allowing for a fully automatic operation. During directory discovery directories maintained by paired devices are discovered. The necessary identifiers are derived from the authenticated keys exchanged during pairing. These keys are also used to establish a secure and authenticated connection, which is utilized for transmitting all service exploration messages protecting the privacy of users.

Directory Discovery. During directory discovery service consumers (clients) discover parts of the directory that might contain information about desired services. As our solution demands that each service provider maintains the information about its own services, directory discovery corresponds to discovering relevant service providers. While hosts do indeed know service providers they are paired with and thus have a common secret, they do not know which of these service providers are online and available in the current network. To discover available known service providers, hosts derive time-frame identifiers from the shared secrets and a timestamp. This identifier, which does not leak private information to third parties, is then made available in a certain scope and paired service providers will initiate an authenticated connection to corresponding hosts. Alternatively, service providers can also announce themselves to paired clients using the same mechanism for creating time-frame identifiers, and clients then initiate the authenticated connection to the corresponding service providers. Publishing the time-frame identifiers can either be realized via multicast or via means we shortly discuss in the next subsection and detail in Chapter 7. The authenticated connection is established using a Diffie-Hellman key exchange which is authenticated via the shared secret exchanged during pairing. Using the pairing secret only for authenticating subsequent connections and not for encrypting payload grants perfect forward secrecy. Again, to be able to efficiently and securely reduce this phase to practice, we provide a protocol relying on TLS, leveraging TLS pre-shared keys (PSK) [Res16]. We detail this phase in Chapter 6.

Service Exploration. The service exploration can now be performed leveraging authenticated connections to service providers. This makes service discovery more efficient
because means for making information available to the public, which are needed for
discovery but cause a lot of network load, are only used for discovering the directory, while
the exploration of available services can be efficiently performed via unicast. We further
subdivide *service exploration* into three sub-phases, *listing*, *selection*, and *location* (see
Chapter 2). Our solution demands that the communication for all these sub-phases is
performed using the authenticated connection between client and service provider. We
do not outsource any of these sub-phases to a third party, e.g. to a match-maker or to a
service broker.

**Public Service Discovery.** Using our privacy extension does not block the typical
public service discovery. Certain services can be publicly announced and discovered without
the need for a device pairing; this is convenient for services that should be accessible by
anyone in a certain scope, e.g. a printer in a working group network. Our user interface
(see Section 4.5) allows making select service types public; it alternatively allows setting
public service discovery as a default and activating privacy protection only for select
service types. It is also possible to provide a public directory, which allows public *directory
discovery* followed by unicast *service exploration* over a connection that provides integrity
via opportunistic encryption, but does not offer authenticity.

### 4.3.3 Multi-Link and Scopes

Our multi-link solution is also based on separating the *directory discovery* and *service
exploration* phases of service discovery. In principle, it works by making queries for
directories and announcements of directories available in a larger network crossing multicast
boundaries. The *scopes* are created by *directories*; each directory is responsible for its own
scope. This means each service provider that offers private services establishes its own
private scope. For public services we introduce *scope name servers* (SNS), which run on
select hosts that maintain the service directory for their respective scope (see Chapter 7).
Privacy and scoping work seamlessly together, especially when leveraging DNS as we do
for our service discovery framework (see Section 4.4). While the Zeroconf hybrid proxy
[Che15] establishes one scope per network link, our solution supports arbitrary scopes that
are not dependent on the network topology; it supports per-link scopes, but also private
scopes, and scopes that are comprised of multiple links.

For making directories configurationlessly discoverable in multi-link networks, we
leverage our Stateless DNS technique proposed in Chapter 8. We detail Stateless DNS
based multi-link service discovery in Chapter 7. This technique relies on the widely
deployed DNS infrastructure. Instead of transmitting the desired resource records via
multicast, it stores these records in the cache of a local DNS server with the help of our
stateless reflector, which allows programming the cache via queries that have a special
syntax but still are valid standard DNS queries. Figure 4.4 illustrates *directory discovery*
leveraging Stateless DNS.

Typically, service discovery solutions are made scalable by introducing a central
directory that has to be configured on clients. Instead of using this central directory, the
directory can be distributed among peers, or kept locally, and can be made discoverable
across multicast boundaries fulfilling the same requirements as the central component.
Existing solutions can substitute our Stateless DNS based *directory discovery* method
for the central directory, and still use their native way of service exploration when communicating with the directory or when discovering services in the local link.

### 4.4 Integration and Architecture

In this section we describe how our solutions for device pairing, privacy-preserving service discovery, and multi-link scaling can be integrated into an existing DNS-SD environment. While we have proof of concept implementations for our solutions, as of yet we do not have an implementation of the full framework (see Section 4.6).

To ease the integration of alternative ways of transmitting DNS resource records for service discovery, and to allow the integration of further service discovery daemons, we propose a service discovery daemon (SDD) that is responsible for demultiplexing client requests to different resolvers. While we use DNS-SD, the SDD could also act as a bridge and integrate other service discovery protocols. Leveraging the proposed service discovery daemon, client software can use a unified interface for service discovery. Figure 4.5 illustrates our proposed architecture.

As for some platforms integrating a daemon might be uncomfortable, we also plan to provide libraries that allow integrating our solutions in client software.
### 4.4 Integration and Architecture

#### Service Discovery

**Avahi**

**DNS-SD/mDNS**

**Client Control Module**

**Pairing Module**

**DNS-SD/DNS**

**Privacy Module**

**IPC or REST**

**DNS-SD/sDNS**

**Figure 4.5:** Service discovery daemon (SDD) architecture. The SDD offers a unified interface to client software and demultiplexes client requests to different means of resource record distribution. Backwards compatibility is granted by the legacy interfaces. Yellow components are provided by existing specifications and software while we provide and discuss solutions for the green and blue components.

#### 4.4.1 Client Interaction

Applications (client software) that want to either offer or request services ask the SDD which then handles the request. This corresponds to applications asking the DNS resolver for resolving a domain name. The SDD supports both a D-BUS and a REST interface.

**D-BUS.** Via the D-BUS interface, the communication with the SDD is very similar to the communication with the Avahi\(^1\) Zeroconf daemon. A seamless and transparent integration can be achieved if the SDD is configured to listen to the D-BUS `org.freedesktop.Avahi` destination. This setup causes client requests to be sent to the SDD without the client software noticing. It either requires changing the Avahi D-BUS destination or deactivating D-BUS for Avahi and integrating the Avahi core library in our SDD.

We also provide an augmented D-BUS interface for client software that explicitly uses the SDD; it allows e.g. to directly set the scope in which services should be requested and offered. All requests made via the D-BUS interface may be overridden due to privacy settings of the privacy module.

**REST.** To achieve platform independence, the SDD also offers a REST interface. In an IoT scenario, the REST interface allows using a stronger device as service discovery server which allows weak devices to outsource the service discovery functionality.

**Legacy.** If the SDD does not override the D-BUS destination, client software may also directly query existing service discovery daemons. This is important for granting backwards compatibility. Our privacy extension for the Avahi daemon (see Chapter 6) provides privacy even when directly requesting the Avahi daemon.

\(^1\)[http://avahi.org](http://avahi.org)
4.4.2 Components of the Framework

Our framework comprises the components shown in Figure 4.5. We cover their basic functionality and their interactions in the following; we provide the details in the remaining chapters of this thesis.

**Service Discovery Daemon.** The SDD receives requests from applications and forwards them to one of the components handling the resource record distribution according to rules set by the control module. The SSD directly handles the meta services necessary for directory discovery, while the meta services necessary for automatic pairing and pairing data synchronization are handled by the pairing module.

**Control Module.** The SDD can be controlled by users via our privacy enhanced phone book which provides the user interface for establishing pairings with contacts. It further allows altering privacy settings and discovery scopes for single service instances, certain service types, or all services. We discuss the user interface in Section 4.5.

**Pairing Module.** The pairing module establishes and manages pairings. It provides the SDD with an interface for retrieving authenticated keys allowing setting up secure authenticated connections. We detail our pairing module in Chapter 5.

**Means of Resource Record Distribution.** Based on sensible defaults or decisions made by the user overriding the defaults, the SDD decides which means of resource record distribution has to be used. Public services for the local scope are handed to a Zeroconf daemon (e.g. Avahi), private services are handed to the privacy module, requests for larger scopes are handed to our DNS-SD/sDNS component, and queries in the Internet scope are forwarded to the local DNS resolver.

4.5 User Interface

The pairing and service discovery daemons alone are not sufficient without a user-friendly application providing users with an interface for straightforward control over both privacy and scopes. Our application looks and feels like a typical contact application but offers enhanced features for tuning privacy and shows the online status of friends and other devices of the user; thus we refer to it as privacy enhanced phone book. Daemons and privacy enhanced phone book empower the users with privacy and transparency when using configurationless service discovery.

Our proposed user interface allows tuning privacy settings of service discovery in a fine grained way. Further it informs the user when a (newly installed) application wants to offer or request a service which is not configured yet, instead of publishing the information without the user’s awareness. Sensible defaults allow configurationless usage, and further ease tuning the privacy properties because default settings that are applied to each new user and each new service can be altered. The interface further provides the functionality of an enhanced service browser; besides listing available service instances it shows the privacy status of these services and allows checking with whom the respective services are currently shared.

In the following we introduce our interface along with screenshots. These screenshots are simplified and do not show the final product. The purpose of covering the user interface
4.5 User Interface

(a) Main page of the user interface. The main page shows the contact list; it further shows which contacts are online (indicated by the green circle) and for which contacts a pairing exists.

(b) The groups view shows available groups and service types shared with members of the respective groups.

Figure 4.6: Contacts and Groups views of our user interface.

is two-fold; (1) foremost we wish to describe the functioning of our solution not only from the research and the technical points of view but also from a user’s perspective; further, (2) this thesis provides a self contained service discovery framework for which we consider the specification of a user interface as important.

We first discuss the main views of the interface and then cover both setting up a device pairing and managing privacy and scopes for service discovery. While the design and development of the underlying mechanisms was challenging, the application is simple from a users perspective as the following description of workflows illustrates.

4.5.1 Main Views

As our interface is intended to be used as a replacement for a typical contact application, its main view is a list of contacts very similar to a typical contact application on today’s smartphones as shown in Figure 4.6(a). For example, Android allows easily exchanging
Chapter 4. A Secure, Scalable, and User-Friendly SD Framework

(a) Contact page. This page shows the pairing status, the groups a contact is part of, and the service types that are shared with a contact.

(b) Group page. This page shows the group’s members and the service types shared with the group’s members.

Figure 4.7: Contact and Group pages, respectively. Pressing a field opens a window that allows altering the corresponding value.

the contact application and the substitute can easily import existing contacts. While the view may show further information, the additions we make are the online status which shows whether the respective contact is reachable (i.e. the respective contact currently shares at least one discovery scope) and a field showing a checkmark if a pairing with the corresponding contact is established. The second main view allows the maintaining of groups; it is shown in Figure 4.6(b). There is a pre-configured default group which all new users are assigned to. In this view users can create new groups pressing the plus button, and open a group context menu by pressing the group name. The context menu allows adding contacts to groups and adding private service types (or even service instances) which are then shared with contacts in this group. Unpaired users may be added to groups but they cannot discover the private services the group has access to until a pairing is established.

When pressing a contact’s field, the contact page opens as shown in Figure 4.7(a). Besides typical contact information, it shows the pairing status, groups the contact is part
4.5 User Interface

(a) Interface for setting the defaults for new users and services.

(b) Settings for a user’s own profile.

Figure 4.8: Settings.

of, and the service types shared with this contact. Pressing one of these fields opens a window that allows altering the corresponding values. Pressing a group’s field in the group view opens the group’s page as shown in Figure 4.7(b). This window shows the group’s members and shared service types, and allows altering these values.

4.5.2 Menu

The "hamburger" menu in the top right corner allows choosing the following views.

Default Settings. The default settings (see Figure 4.8(a)) determine the values that are set for new contacts and services. It (1) allows setting the groups to which new users are automatically added, (2) choosing whether new services are only offered to paired contacts or publicly, (3) choosing in which scopes new service types are offered and queried, and (4) choosing whether new services — both offered by the own device or discovered — should cause a notification.
**Own Profile.** The *My Profile* page allows altering a user’s own profile; it is shown in Figure 4.8(b). The user name is transmitted during pairing and allows for an automatic setup of an entry in the paired device’s phone book. While the screenshot only shows the users name, further information, e.g. the phone number and email address, can also be provided. The domain name field allows choosing the device’s domain name that is announced via mDNS. The `<scope>` portion of this domain name is fixed. If domain name privacy is chosen, the domain name is only announced to paired devices. Further, via this view the user’s own devices may be managed and paired; we propose and discuss the utilized *intra-user* pairing method and the *pairing data synchronization service* in Chapter 5.

**Automatic Pairing.** The automatic pairing view allows activating the automatic pairing mode proposed and described in Chapter 5.

**Manage Service Types.** The manage service types view (see Figure 4.11(a)) allows managing privacy and scope settings of service types. We discuss this view subsequently.

**Service Browser.** The service browser view shows an *enhanced service browser* that allows browsing for services in available scopes. Besides typical DNS-SD/mDNS service information it also shows the privacy status, the friends, and the groups associated with each service. It further allows editing the privacy and scope settings for services by pressing a service’s name.

### 4.5.3 Pairing

Our service discovery framework requires a single pairing per pair of users. In this subsection we describe the involved procedure from a user’s perspective which illustrates its simplicity. We detail the research aspects and technical aspects of our device pairing methods in Chapter 5. For the *discovery* phase of pairing, we propose both a DNS-SD/mDNS and a QR code based method; in the following description we use the QR code method as it is the preferred method when both devices have both a display and a camera, which is the case for our smartphone example.

A user may either initiate a pairing with a peer that is already in its contact book, or initiate the pairing during adding a new entry, or just initiate a pairing request and use the contact information received from the peer to automatically generate a new entry. We illustrate the process of adding pairing information to an existing entry.

Assume Alice wants to establish a pairing with Hina. The screenshots illustrate Alice’s perspective. Both Alice and Hina either press the unchecked pairing field of the respective contact in the main view (Figure 4.6(a)), or select the respective contact’s page by pressing the contact’s name. In case of pressing the unchecked field, a pairing dialog appears giving the option to either display a QR code or scan a QR code; this dialog is shown in Figure 4.9(a). The same choices can be made in an unpaired contact’s page shown in Figure 4.9(b).

Let’s assume Alice chooses *Display QR*, which makes her contact application display a QR code as shown in Figure 4.10(a). In this case, Hina has to choose *Scan QR* which opens a QR code scanner integrated in the contact application. Hina will now scan the
4.5 User Interface

(a) Pairing dialog popping up when pressing an unchecked pairing field.

(b) A contact’s page also allows initiating the pairing.

Figure 4.9: Ways of starting the pairing process.
Chapter 4. A Secure, Scalable, and User-Friendly SD Framework

(a) QR code displayed by the device whose user chose to display the QR code.

(b) QR code scanner running on the device whose user chose to scan the QR code.

Figure 4.10: Pairing via the QR code method described in Chapter 5.
4.5 User Interface

(a) Overview over the maintained service types.  
(b) Settings for the presence service type.

Figure 4.11: Managing service types.

QR code displayed on Alice’s device. During this scanning process, the QR code on Alice’s device will change, and Hina will also scan this second QR code. She can scan both QR codes without any interaction by just holding her phone in such a way that her scanning application may scan the QR codes displayed on Alice’s device. The rationale behind using two QR codes is provided in Chapter 5. If the pairing is successful, Hina’s device shows a message as illustrated in Figure 4.10(b). Hina then informs Alice about the successful pairing (by either showing her the message or telling her), and, thereafter, Alice must press the confirm button on her device (see Figure 4.10(a)), which concludes the pairing process.

Pairings of devices of the same user can be managed via the My Profile page (see Figure 4.8(b)) either utilizing automatic pairing, which can be activated in the menu, or via manual pairing as described above.

4.5.4 Privacy and Scope Settings for Service Discovery

Via the menu in the top right corner, users can select the service type management page shown in Figure 4.11(a). This page shows service types that either have been automatically
added as they have been discovered or published before, or manually added using the plus button. Automatically added services have the default properties as defined in the default settings page (Figure 4.8(a)). Pressing a service type opens this service type page (see Figure 4.11(b)).

This page allows selecting the privacy and scope settings for the corresponding service type. If privacy is deactivated, the service is published in a classical DNS-SD way so that every device that shares a scope with the offering device may discover it; if privacy is activated, the service is only offered to select users that either have been directly specified or are part of a group with which this service is shared.

The service page further allows specifying with which groups the service type is shared, with which users (regardless of their groups) the service type is shared, and in which scopes the service type is discovered and announced. While the settings related to groups and users only affect private services, the settings related to scopes also affect public services.

4.6 Implementation

As listed in Chapter 1, we have prototypical implementations for our pairing, privacy, and multi-link scaling components. We discuss these implementations in Chapter 5, Chapter 6, and Chapter 7, respectively. The implementation of our pairing daemon also provides the privacy enhanced phone book functionality. As of yet, we do not have an implementation of our service discovery daemon (SDD).

4.7 Network Impact

Users do not want to pay for privacy with complicated configuration, but they also do not want to pay with performance loss. One of the main goals of our privacy extension is to be at least as efficient as standard DNS-SD/mDNS in terms of both network and host performance. Our privacy and scope solutions do not increase the network load; in fact they reduce the network load by using mostly unicast instead of multicast. Further the computational overhead on the host devices is imperceptible for the user, both in terms of responsiveness of the system and battery life.

Our privacy extension uses multicast only for directory discovery, and our scope extension does not use multicast at all. When service providers privately offer their services across multicast boundaries, they also use the scope extension. Our service discovery framework performs service exploration via unicast. In the following analysis we generally compare substituting a few unicasts for multicast, as this is done by both our privacy and scope extensions. We specifically analyze the network load impacts of these extensions in Chapter 6 and Chapter 7, respectively.

If a desired service type is offered by several directories, a unicast query is sent to each of these directories instead of sending a single multicast, which would reach all directories. Similarly, if multiple clients have registered for push updates of a certain service, the service provider sends several unicast messages in case of a push event instead of a single multicast, which would reach all clients. This might look like a performance problem, but in small networks, the influence of mDNS packets on the network load is negligible (see discussion below), whereas in huge networks sending a few unicasts has a lower impact on the whole network than sending a single multicast which is received by multitudes of peers increasing the load on each of these peers.
4.7 Network Impact

(a) Network utilization time frames used in the whole local network.

(b) Network utilization time frames used on average per access network.

Figure 4.12: Comparison of network utilization time frames needed by 10 unicasts and a single multicast, respectively, with respect to the number of access networks. We assume that each access network has at least one associated host.

Assuming each device wants to transmit one service discovery packet per time unit, using multicast, devices have to transmit one packet but receive as many packets as there are users in the multicast group (which roughly corresponds to all devices in the network for the mDNS multicast group); using unicast, devices have to send and receive at most as many messages as there are relevant directories or registered clients. Since there are many services that are only relevant for a few peers, e.g., device synchronization, we assume sending each packet to 1% of the devices in a network as a default, which results in substituting 10 unicasts for a single multicast in our default network of 1000 hosts.

The influence of many multicasts on the network load is especially severe in huge WiFi networks, because multicasts are transmitted using a very low transmission rate so that older devices not supporting higher transmission rates can receive the multicasts as well [VTLAY14]. Hong et al. [HSS09] show that 13% of their campus network bandwidth is used by DNS-SD/mDNS; the authors also describe formulae for generally analyzing the bandwidth utilization caused by mDNS in 802.11 WiFi networks [80212].

Abstract Comparison of the Unicast and Multicast Network Utilization. To illustrate the general affect of both a few unicasts and a single multicast we abstract from specific WiFi standards, e.g. IEEE 802.11[80212]. While the different WiFi standards have quite a few specific properties, they share properties that are crucial for this comparison. The following properties and definitions are relevant for our abstract comparison.

- Hosts connect to an Access Point (AP) and send every message to this AP, which in turn forwards it either to a host associated to itself or to its up-link.
- All hosts associated with an AP (and this AP itself) share the same medium, which allows half duplex transmission. This means, only one device can send at a certain point in time.\footnote{We do not consider multiple channels per AP and MIMO because each channel can be modeled as a virtual AP in our analysis.} We refer to the network of hosts associated with the AP (including the AP) as access network.
- While a unicast can be sent using the maximum transmission rate both the AP and the respective host support, multicasts have to be sent at the lowest transmission rate so that older hosts also receive the packet.

Abstract Comparison of the Unicast and Multicast Network Utilization. To illustrate the general affect of both a few unicasts and a single multicast we abstract from specific WiFi standards, e.g. IEEE 802.11[80212]. While the different WiFi standards have quite a few specific properties, they share properties that are crucial for this comparison. The following properties and definitions are relevant for our abstract comparison.

- Hosts connect to an Access Point (AP) and send every message to this AP, which in turn forwards it either to a host associated to itself or to its up-link.
- All hosts associated with an AP (and this AP itself) share the same medium, which allows half duplex transmission. This means, only one device can send at a certain point in time.\footnote{We do not consider multiple channels per AP and MIMO because each channel can be modeled as a virtual AP in our analysis.} We refer to the network of hosts associated with the AP (including the AP) as access network.
- While a unicast can be sent using the maximum transmission rate both the AP and the respective host support, multicasts have to be sent at the lowest transmission rate so that older hosts also receive the packet.
• A local network can be comprised of several access networks. The more access networks are available the more overall network bandwidth does the local network provide.
• When hosts in the local network communicate with each other, a unicast affects the access networks the sending and receiving hosts are associated with, respectively, while a multicast affects all access networks.

We abstract from physical and link layer overhead, from the actual transmission speed, and from the size of the average transmitted packet. We introduce a unit for the time frame that is needed for transmitting an average sized packet at the highest unicast transmission rate, and refer to it as a network utilization time frame (NUTF). We further do not distinguish between upstream (to the AP) transmission time and downstream (from the AP) transmission time. While a multicast transmission needs one NUTF on the upstream as the involved communication is just between the sender and its associated AP, we chose a default cost of 20 NUTFs on the downstream, which roughly corresponds to the ratio between unicast and multicast when using 802.11n [80212] with a transmission rate of 300 Mbit/s and setting the multicast transmission rate to 11 Mbit/s. The ratio is even more in favor of unicast when choosing a faster standard while maintaining backwards compatibility. Using these defaults, a unicast needs 2 NUTFs, while a multicast needs $20 \cdot k + 1$, where $k$ is the number of APs in the local network. This might lead to the wrong assumption that increasing the number of access points in a local network decreases multicast performance; this is not the case as each new access point increases the overall number of available NUTFs in the network — but multicast ignores the benefit of these additional NUTFs. Figure 4.12 illustrates the comparison of the network impact of a few unicasts and a single multicast in terms of NUTFs with respect to the number of APs; we assume that each AP in a local network has at least a single associated host.

### 4.8 Conclusion and Future Work

We gave an overview over the service discovery framework proposed in this thesis, which solves the main problems current configurationless service discovery solutions suffer from — privacy breaches and being confined to a single network link — by

- providing user-friendly means for device pairing, which allows establishing secure authenticated connections at any later point in time;
- discovering service directory parts (devices) instead of services, which reduces the number of necessary multicasts as the number of entities that is discovered via multicast is reduced;
- using unlinkable time-frame identifiers as the only information in multicast messages granting privacy during directory discovery.
- establishing a secure and authenticated connection to paired devices for service exploration, which grants the privacy-preserving transmission of service related information.
- leveraging our Stateless DNS technique for scaling configurationless service discovery to multi-link networks.

We further described and illustrated our service discovery framework from a user’s point of view. Our framework neither outsources any of the arising problems to third parties
nor depends on poorly sketched components; the design for all necessary components is proposed and detailed in this thesis.

We work on an implementation of our service discovery daemon. Further, we plan to augment our framework providing a component for more sophisticated service selection, which will help scaling the solution to even larger networks.
Privacy preserving communication is a crucial component for every privacy-preserving network application. To engage in privacy-preserving communication, devices need an authenticated pre-established relationship.

This chapter proposes a device pairing mechanism that establishes a relationship between two devices by agreeing on a secret and manually verifying the secret’s authenticity. Pairing has to be performed only once per pair of devices, as for a re-discovery at any later point in time, the exchanged secret can be used for mutual authentication. The proposed pairing method is suited for each application area where human operated devices need to establish a relationship that allows configurationless and privacy-preserving re-discovery at any later point in time. For the necessary human interaction, we especially care about usability. This chapter further proposes a pairing daemon that manages pairings for arbitrary applications, providing convenience for both end users and application programmers.

We published an IETF Internet Draft on a pairing mechanism for DNS-SD over mDNS, which is based on the solution presented in this chapter. Our Internet draft has been adopted by the IETF dnssd working group, rendering our solution a potential standard for privacy-preserving DNS-SD device pairing.
I would like to thank Christian Huitema (Microsoft) for publishing the IETF Internet Draft on device pairing [HK16a] together with me, and for all the helpful discussions while working on the draft.
5.1 Introduction

Communication — the exchange of information — is the main reason for networking. Today’s devices are almost non-stop attached to networks and communicate with other devices. This might be for the synchronization of data among the devices themselves or a means for users of devices to communicate with each other. We argue that in general, it is preferable that only the peers devices intend to communicate with, are able to extract the contained information from the corresponding messages. It would be desirable for all device communication to be privacy-preserving. While privacy may not be necessary for certain applications, it is not a disadvantage per se. Analog real world examples include post transfer, where a sender puts a letter in an envelope instead of using a post card so that only the desired recipient might read the letter. For non-private mails, where a sender deliberately chooses a post card, the sender does not care that the postman can also read it, but being readable to third parties is not a desired requirement. A main reason for the existence of post cards also aligns to device communication: transmission cost. To make private communication ubiquitous, means that are not more expensive in terms of both efficiency and user-friendliness are required. The main part of these potential costs, the costs that users have to pay in terms of interaction or configuration, is not payed during communication but rather before the first engagement.

To engage in secure and privacy-preserving communication, hosts need to differentiate between authorized peers, which must both know about the host’s presence and be able to decrypt messages sent by the host, and other peers, which must not be able to decrypt the host’s messages and ideally should not be aware of the host’s presence. The necessary relationship between host and peer is typically established by a centralized service, e.g. a certificate authority, or, more rarely, by a web of trust, e.g. PGP.

However, these solutions come with inherent disadvantages. A centralized certificate authority must be trusted, it must be available, and foremost, users must obtain certificates, which is complicated, tedious, and might be expensive. Further, setting up certificates involves uncomfortable configuration. A web of trust mitigates the central trust problem, but it is still dependent on key servers, and demands configuration, which should not be underestimated. For applications that mainly act and communicate in the edge of the

Figure 5.1: Service discovery framework architecture proposed in Chapter 4. In this chapter, we present device pairing.
Chapter 5. Device Pairing

Internet, it would be nice to establish the necessary relationship between devices without having to rely on a central component.

Direct device pairing in the edge network without any involvement of a third party can be achieved by manually authenticated pairing mechanisms. Thereby, the users’ devices automatically exchange pairing data, and then users manually verify this pairing information. Well known techniques like Bluetooth [Blu14] and WPS [All14] establish the desired relationship via manually authenticated pairing.

However, existing solutions have at least one of the following undesired properties. (1) They are not user-friendly, demanding e.g. the comparison of two different, unnecessarily long authentication strings; (2) they might be user-friendly but insecure, providing no protection against man-in-the-middle attacks; (3) they do not provide privacy as they publish device information in the discovery phase; and (4) they might be limited to specific application areas, e.g. by demanding a fixed protocol stack. Further, existing solutions do not provide a seamless experience and demand a new manual pairing for each application, sometimes even for each connection establishment.

In this chapter we propose device pairing mechanisms providing human operated devices with pairwise authenticated secrets, allowing mutual automatic re-discovery at any later point in time along with mutual private authentication. We especially care about privacy and user-friendliness.

The proposed pairing solution consists of three steps needed to establish a relationship between a host and a peer:

1. Discovery of the peer device. The host needs a means to discover network parameters necessary to establish a connection to the peer. During this discovery process, neither the host nor the peer discloses its presence.
2. Secure pairing data exchange. The devices agree on pairing data that can be used by both parties at any later point in time to generate identifiers for re-discovery and to prove the authenticity of the pairing. The pairing data can e.g. be a shared secret agreed upon via a Diffie-Hellman key exchange.
3. Manual authentication of the pairing data. Since in most cases the messages necessary to agree upon pairing data are sent over an insecure channel, means that guarantee the authenticity of these messages are necessary; otherwise the pairing data is in turn not suited as a means for a later proof of authenticity. For the proposed pairing mechanism we use manual interaction involving an SAS (short authentication string) to prove the authenticity of the pairing data.

We further provide a pairing daemon that seamlessly integrates all stages of pairing and manages pairing for arbitrary applications. Encapsulating device pairing in a dedicated component provides benefits for both users, making a single pairing per "friend" sufficient, and application developers, allowing utilizing the easy-to-use pairing daemon interface instead of implementing a separate, most likely insecure, pairing solution for each application. Figure 5.1 illustrates the integration of the pairing component in our privacy-preserving service discovery framework that we proposed in Chapter 4.

In this chapter, we provide the following contributions.

1. We provide an extensive discussion of related work surveying all relevant components of manually authenticated pairing methods in a structured way, and further covering related areas for the sake of delineation.
2. We explain the details of constructing a secure manually authenticated pairing protocol, covering several important design choices.
3. We propose an attack on a recent pairing protocol claimed to be the most efficient, and show that a minimal number of three messages is necessary.

4. We propose efficient methods for all three stages of pairing, discovery, secure pairing data exchange, and manual authentication. Besides security, we especially care about user-friendliness. We take both theoretical aspects as well as reducing these methods to practice into consideration.

5. We propose an efficient and flexible pairing daemon, which seamlessly handles all stages of privacy-preserving device pairing. The pairing daemon provides convenience for both end users and application developers.

5.2 Related Work

Since device pairing is a necessary basis for most secure communication methods, much research has been done in this field. We focus on mechanisms that are independent of central services, and relay on manual verification of data transmitted via an out-of-band (OOB) channel.

First, we present survey papers — both general and more specific ones — for readers who want a broad but detailed overview over device pairing. As further comparative work, we also cover user studies regarding the usability of pairing protocols and the influence of users on the actual security.

Starting the discussion of milestones in the area of pairing protocols, we discuss work on authenticating the Diffie-Hellman key exchange [DH76] via a PKI and pre-shared keys, respectively, as these ways of authentication were developed first and have influenced the class of pairing mechanisms we are interested in. We then present select OOB channel pairing protocols that we consider most useful for reenacting the design of this class of protocols, and which also act as a basis for the techniques presented in this chapter. Besides these protocols, we also present ways of how the human interaction can be leveraged to authenticate pairing related information. To delineate from other classes of pairing mechanisms, we also shortly discuss related work on secure in-band pairing. Further, we discuss widely used practical applications of these protocols.

5.2.1 Surveys

Nguyen et al. [NR11b] provide a thorough survey of out-of-band authentication protocols that use manual transfers of short authentication strings; these protocols can be used for device pairing. The authors analyze existing protocols with respect to the number of necessary messages and the computation time. Further, they improve existing protocols; some of these improvements were published in earlier work [NR08]. They introduce the concept of "commitment before knowledge" and propose separating the two roles of one-way message authentication [NR09b]. These concepts are used by most of the protocols reviewed in [NR11b] and — when properly realized — prevent offline combinatorial search attacks, while allowing authentication via comparing a small authentication string (SAS) because attackers are confined to one-shot online attacks. The authors discuss digest functions and short-output hash functions that can be leveraged to construct efficient and user-friendly pairing protocols in [NR09a, NR11a, NR12]. Group pairing protocols are also discussed in [NR11b].

1 We discuss device pairing mechanisms in Section 5.3.
Mirzadeh et al. [MCT14] survey existing secure pairing methods for pairing with both a single device and groups of devices. Their survey is less formal compared to [NR11b], while still providing the necessary mathematical background. In their discussion, the authors also take the protocols improved in [NR11b] into consideration.

For readers unfamiliar with the matter, [MCT14] is — in our opinion — the easier understandable introduction to the research area of pairing protocols. A further point that makes this paper easier to read is its being closer to practice also discussing realizations of OOB channels. Nonetheless, we consider [NR11b] the best overview for researchers working on improving certain aspects of device pairing or using device pairing as a crucial building block for their research, where a deeper understanding of the pairing mechanisms is of the essence.

5.2.2 Usability Studies

Before discussing related work about select protocols, we want to discuss related work that compares and evaluates the usability aspects of different ways of realizing OOB channels.

Kainda et al. [KFR09] (the work originated in the same research group as [NR11b]) point out the fact that the effective security of pairing protocols does not only rely on provable secure pairing protocols, but also on the involved human interaction, which is mainly dependent on the realization of the OOB channel. They focus on security, conducting a user study to empirically evaluate the effect of different out-of-band channel realizations on the security of pairing protocols, whose theoretical security is based on the afore mentioned 'commitment before knowledge' concept. The study is based on an earlier study conducted by Uzun et al. [UKA07]. The authors further discuss the problems arising from security and usability trade-offs in a more general way in [KFR10].

The papers [UKA07, KSTU09b, KSTU09, USK11] share common authors and are related to each other. Kumar et al. [KSTU09b] conducted a usability study for pairing scenarios where users pair two of their own devices (we will call this kind of pairing intra-user pairing). The authors also give a good overview over different means for realizing OOB channels. The paper enhances a previous paper of the same authors [KSTU09a] by a formal statistical analysis of the user study. Following [KSTU09b] and presented at the same conference as [KFR09], the focus of [KSTU09] is purely on usability and contains a more thorough user study.

With respect to overall usability, which is most interesting for this thesis, all these studies agree on number comparing being the winner.

Uzun et al. [USK11] conducted a user study to analyze the usability aspects of different OOB channels when the two devices to be paired remain in the hands of separate users. This scenario is most relevant for this thesis as it concerns pairing a device with a device of a friend (which we will call inter-user pairing). While performing the pairing tasks of the study, users preferred to not show their devices' display to each other; one reason being, the authors also took pairing of users who do not know each other into consideration. For the two-user (inter-user) scenario, the comparison of real-language-like sentences was the overall winner.

5.2.3 Secure Pairing Protocols using PKI or Passwords

The first protocols for authenticated key exchange either used a public key infrastructure (PKI) or were based on passwords known by both parties; the latter family of protocols is
5.2 Related Work

Known as Password-Authenticated Key Agreement (PAKE). We also want to shortly cover these types of protocols, as they are a basis for manually authenticated pairing protocols.

Diffie et al. [DVOW92] present a protocol for an authenticated key exchange, which uses certificates. The SIGMA family of authenticated key exchange protocols [Kra03] guarantees authenticity leveraging digital signatures and message authentication codes (MAC). These protocols are used in IETF Internet Standards (RFCs) as cryptographic basis for the Internet Key Exchange (IKE) [HC98, Hof05, Kau05], which is used for IPSec [KS05]; one of the authentication methods used for IKE are X.509 certificates [CSF+08, Yee13]. Canetti et al. [CK02] provide an analysis of IKE and the used SIGMA protocols. RFC 4322 [RR05] defines opportunistic encryption using IKE; this provides a means for secure communication without prior knowledge, but also uses a central component: the DNS.

Bellovin et al. [BM92] presented the first secure Password-Authenticated Key Agreement (PAKE) protocol. They call their family of protocols "Encrypted Key Exchange (EKE)". The authors published an augmented version of the EKE [BM93]; augmented with respect to PAKE means that even if the pre-shared data is stolen from one party (the server side), the adversary cannot impersonate an authorized client. Further research on PAKE protocols has been done in [BMP00, BPR00, MPS00, KOY01, Gen08]. EKE modes of operation for TLS [DR08] have been proposed in [SBEW01, ABC+06]. Further, there is an EAP [ASE08] authentication method using EKE [SZTF11].

Zero-knowledge proofs [GMR89] are a basis for PAKE protocols and also connected to authenticated pairing methods. Especially the Socialist Millionaires Problem [FNW96, BST01], which — in a general informal way — describes the problem of two parties that want to verify the matching of two instances of a certain value of which they both hold one, e.g. a password, without disclosing any information about the values. Checking whether passwords match without disclosing any information about the passwords brings important advantages for PAKE protocols; lessons learned from this can also be applied to manually authenticated pairing protocols, which we will discuss later in this chapter. In practice, a Socialist Millionaires Protocol is e.g. used for Off-The-Record (OTR) messaging [BGB04], which is a widely used protocol for end-to-end encrypted instant messaging.

5.2.4 Secure Manual Pairing Protocols

In this subsection we discuss related work on secure pairing protocols relying on authentication via user interaction. The discussed work focuses on the pairing protocol itself and mostly does not specify an implementation of an OOB authentication channel. OOB channels are introduced in an abstract way by assuming a certain set of properties. Some of this work also gives examples of how such channels can be realized, but we focus on that matter in next subsection.

Stajano et al. [Sta99] proposed the Resurrecting Duckling model, which sends secret data over a location-limited out-of-band channel. Devices (ducklings) are paired to a master device (mother duck) using a low-bandwidth OOB contact channel. Except for bringing the devices close to each other, no human interaction is necessary. But the method is limited to a master-slave scenario and further, the OOB channel requires secrecy, which is realized by physical contact. In general, the secrecy requirement either demands special interfaces or plugging in a cable, which again defies the usability aspect.

Balfanz et al. [BSSW02] build on concepts of [Sta99], generalize the mechanism, and present the first pairing protocol using manual comparison of hash values. The security of the protocol depends on the collision resistance of the hash function used to generate the
hash value that is transferred via the OOB channel. Since users have to compare this hash value and today’s collision resistant hash functions (e.g. SHA256) generate 256-bit output, the protocol is not user-friendly. Pasini et al. [PV06] provide an enhancement of the Balfanz protocol preventing chosen-plaintext attacks by using a commitment scheme. Mashatan et al. [MS07] also provide an enhancement to provide the same security enhancement by using hybrid collision-resistant hash functions also presented in their paper. Basically, they combine a seed — which the attacker cannot control — with the input of a cryptographical hash function to avoid combinatorial search. The authors also provide a formal model for general non-interactive\(^2\) manually authenticated pairing protocols.

The first class of non-interactive manually authenticated pairing protocols that significantly reduce the length of the string that users have to compare — a protocol family called MANA — was presented by Gehrmann et al. [GMN04]. These protocols hand a short authentication string (SAS) to the user interface, which users have to verify. The MANA family is an important milestone for the development of manual authenticated pairing protocols, but neither is it optimal with respect to user interaction, nor with respect to security. While the authors use a commitment bit, which violates the non-interactiveness, Vaudney [Vau05] proposes to use a \textit{stall-free} OOB channel instead (we will discuss different types of OOB channels in 5.3). Vaudney [Vau05] also presents a further one-way authentication protocol that is secure and optimal with respect to SAS length, but requires interaction beyond a single-bit commitment as the other party has to generate and send a nonce (via the unsecured network). Nguyen et al. [NR09b] present improved versions of both the original and Vaudney’s MANA, which are secure, optimize the SAS length, and relax OOB channel requirements. A security proof for a Diffie-Hellman version of the improved MANA protocol can be found in [NR09a].

Hoepman et al. [Hoe04, Hoe05] and Wong et al. [WS05, WS07] present interactive SAS based manually authenticated pairing protocols. While the Hoepman protocol uses \textit{direct} binding (defined in [NR11b]), the Wong-Stajano protocol uses \textit{indirect binding}. Since both protocols generate a separate SAS per user, the protocols are not optimal with respect to user interaction as two separate SASes have to be verified. A protocol that is similar to the Wong-Stajano protocol but only needs a single SAS is presented in [CCH06]. A further improved version that also needs a single SAS is proposed in [NR11b]. Nguyen et al. [NR11b] also present an enhanced version of the Hoepman protocol, which is secure and optimal in all of user interaction, messages via the insecure network, and computation time. We use this enhanced version of the Hoepman protocol as basis for our pairing solution.

A further interactive protocol that only uses a single SAS is the basis for Bluetooth Simple Pairing [Blu07b]. While the protocol is optimal with respect to human interaction, it uses unnecessary messages over the insecure network and unnecessary computation time [NR11b]. An improved version using a commitment scheme was proposed by Laur and Nyberg [LN06]. The protocol is optimal in user interaction and messages but needs both more computation time and is more complicated compared to the improved Hoepman protocol.

As we are working on a pairing document for the IETF in the area of DNS-SD, it is especially interesting to look at related work developed within the IETF. RFC6189 [ZJC11], which is on ZRTP, describes a pairing based on a committed Diffie-Hellman key exchange and SASes. We are currently reviving and updating the draft on an SAS

\(^2\)We explain the difference between \textit{non-interactive} and \textit{interactive} protocols in Section 5.3.
5.2 Related Work

based TLS extension [MGR14], as it would be advantageous to leverage an existing TLS mechanism.

5.2.5 Formal Analysis

We also want to briefly take up the formal analysis of manually authenticated pairing protocols. Chang et al. [CS07] provide a formal analysis of Bluetooth pairing leveraging the ProVerif\(^3\) cryptographic protocol verifier.

Nguyen et al. [NL14]\(^4\) formally analyze manually authenticated pairing protocols. For their formal analysis, they augment the Strand Spaces model [FHG98]. The authors also propose a new protocol [NL15] which they claim to be the best; they provide a formal analysis of their new protocol using the same methods. However, we show in Section 5.3 that this protocol actually is not secure.

5.2.6 Authentication Channels

In this subsection we present related work that mainly focuses on the OOB authentication channel, providing methods for both presenting the short authentication string (SAS) and realizing the human interaction necessary to perform the SAS verification. The afore discussed user studies give a good overview of different verification methods. The most common methods, comparing numeric or alphanumeric representations of SASes, are discussed e.g. in [KFR09, KST\(^`\)09, USK11].

Saxena et al. [SEKA06] present methods for visual channel device pairing, which act as basis for the QR code method used in the solution presented in this chapter. Zhang et al. [ZRX\(^+\)16] present a way to realize a secure OOB channel via barcode scanning. Cagalj et al. [CCH06] propose both a committed DH key exchange and three methods to realize the authentication channel: via visual comparison, distance bounding, and integrity codes (see also [MCT14]).

Sethi et al. [SAA14] present a pairing method that is based on simultaneously drawing with two fingers (e.g. thumb and index finger) on the touchscreens of two devices. Mayrhofer et al. [MG09] propose a verification method that requires to shake two devices simultaneously. Studer et al. [SPB11] also provide a shake-based pairing protocol and argue that the bump\(^5\) protocol is not secure. Since for these methods one user has to hold both devices, they are only suited for a single user pairing scenario. Chen et al. [CNR11] show how contextual information can be used for authentication using OOB channels.

Pairing methods for interface constrained devices are proposed in [STU09]. Also suitable for interface constrained device is the method described in [STU08], which realizes a secure OOB channel that is used for transmitting all messages necessary to complete the pairing data exchange.

5.2.7 Secure In-Band Pairing

To delineate from other types of pairing mechanism, we shortly want to take up pairing mechanisms that do not relay on OOB channels but transmit data necessary for authen-

\(^3\)http://prosecco.gforge.inria.fr/personal/bblanche/proverif
\(^4\)Even though the surnames match, the authors of this paper and [NR11b] are different.
\(^5\)http://blog.bu.mp
tication in a sophisticated way via the unsecured network. We do not consider these methods, as they leverage specifics of the physical layer.

Čapkun et al. [ČČR+08] introduce integrity codes that are inserted into messages for protecting the integrity of messages sent over an insecure radio channel. They further leverage their integrity codes to provide means for message authentication, which can also be used for key establishment. Gollakota et al. [GAZK11] extend this work and introduce a communication primitive that in addition to providing integrity, also prevents an attacker from hiding messages. Hou et al. [HLG13] extend secure in-band pairing to groups of devices.

5.2.8 Practical Applications

We further want to discuss widely-used well-known practical applications of pairing protocols and also argue, why we did not base our pairing component on one of these mechanisms.

Bluetooth pairing. Bluetooth pairing is a very good example to demonstrate a history of improvements and security analysis of pairing protocols; not only because it is widely used in practice.

The first versions of Bluetooth pairing [C+99a] transmitted the secret, a PIN, directly over the OOB channel. The improved pairing method — Bluetooth Simple Pairing [Blu07b, Blu07a] — uses an SAS based approach as mentioned before.

Attacks on the original Bluetooth pairing protocol were first presented by Jakobsson et al. [JW01]. Stajano et al. [WSC05] implemented these attacks and propose methods to fix the original Bluetooth pairing.

Attacks on the Simple Pairing Protocol are presented in [HT08, HH08, Lin08]. A general threat and security analysis of Bluetooth 4.0 is provided in [SD12]. Minar et al. [MT12] surveyed Bluetooth security threats and solutions.

Besides specifying the pairing protocol, the Bluetooth core specification [Blu14] specifies several ways of realizing the human assisted verification, namely numeric comparison, passkey entry, just works and OOB. Numeric comparison and passkey entry correspond to the compare & confirm and copy & enter methods, respectively, which we will discuss later in this chapter. The just works method transmits the authentication string via the unsecured network and thus offers no protection against a man in the middle attack. While normally the first two methods are considered to realize OOB channels, the Bluetooth core specification only refers to OOB channels for further channels that are not part of the core specification and use other means of transmitting the SAS. A Bluetooth OOB channel realization is e.g. the NFC method [NFC14].

Bluetooth with low energy (LE) [Blu10] especially suffers from security problems [Rya13]. This is not in small part due to the fact that in contrast to Simple Pairing used in Bluetooth basic rate/enhanced data rate (BR/EDR), Bluetooth LE pairing mechanisms do not leverage the well-established Diffie-Hellman key exchange.

6Simple Pairing is part of the Bluetooth core specification since version 2.1 published in 2007. The current version of the core specification is 4.2 [Blu14].

7Bluetooth LE and its special pairing mechanism are part of the Bluetooth core specification since version 4.0 published in 2010.
5.2 Related Work

**WPS.** Wi-Fi Simple Configuration [All14], also known as Wi-Fi Protected Setup (WPS), is also a wide-spread well-known pairing method, which is used to associate devices with a Wi-Fi access point. Besides Access Point, there are two further roles devices can have: Enrollee and Registrar. Devices that want to join the network (Enrollees) can either be directly paired with an Access Point or with a Registrar, which is a device that has earlier been paired with the Access Point and been chosen as Registrar by the Wi-Fi administrator.

Like Bluetooth Simple Pairing, WPS is based on an authenticated Diffie-Hellman key exchange and offers several ways of realizing the human verification, namely push button, PIN, and NFC.\(^8\) The push button method is similar to the just works Bluetooth method, providing no protection against man-in-the-middle (MitM) attacks at all. With respect to the user interaction, the PIN method is similar to the passkey entry method of Bluetooth; however, the PIN itself is a secret (even if it is generated for devices that have a display) and often fixed for a single device, while the Bluetooth passkey is not an input to the security algorithm and not a secret; thus this method corresponds to the PIN entry method of earlier Bluetooth specifications, e.g. [C+99a]. The NFC method is the only method that does not demand a secret verification method; however, an attacker could steal the RFC tag of a device before the device is paired. The specification refers to these methods as in-band configuration methods. In this case, in-band refers to configuration, which is mainly performed via WLAN, the unsecured in-band channel. Nevertheless, these methods use an OOB channel that requires human interaction for authentication. There is also the out-of-band mode of operation where all the configuration is performed via an OOB channel. Methods include directly providing the WLAN credentials on an OOB medium, e.g. a NFC tag, and performing a Diffie-Hellman key exchange via NFC.

WPS is not in our focus because it is a master-slave method, it mostly requires a secure OOB channel, and even with the most secure in-band configuration NFC method only provides one-way authentication. We further do not focus on methods where the whole exchange is performed via the OOB channel (out-of-band configuration) as this derogates the number of feasible OOB channels and especially renders the most user-friendly verification method [KFR09, USK11] — compare & confirm — infeasible.

Kuo et al. [KWP07] analyze and compare WPS and Bluetooth Simple Pairing. A feasible brute force attack against the WPS PIN method is presented in [Vie11].

**Instant Messaging.** The proliferation of instant messaging applications for smartphones, which act as an enhanced successor of the classical SMS, has at some point lead to popular demand for more privacy. Today, end-to-end encryption is a well-established feature among instant messaging applications. For establishing an end-to-end encrypted communication in a user-friendly way, these applications establish a device pairing.

The big advantage for the used protocols is that they can leverage the phone number as an identifier and classic SMS as a kind of OOB channel\(^9\) for authenticating a user to the provider of such an instant messaging app. Because these protocols rely on central services and SMS, they are not in our focus; nevertheless they are interesting to study, as they are widely used and overlap in the techniques for realizing an authenticated key exchange.

---

\(^8\)Further, there is a deprecated method where users have to transmit pairing data via a USB pen drive.

\(^9\)It is not an OOB channel that is compliant with the definitions used in this thesis. These protocols relay on the SMS being delivered to the rightful owner of the corresponding phone number.
A prominent example of an open source instant messaging application supporting end-to-end encryption is Signal (formerly known as Text Secure). Frosch et al. [FMB+14] describe and analyze the Text Secure (Signal) protocol in detail. Unger et al. [UDB+15] survey secure instant messaging solutions.

5.3 Device Pairing Background

In this section we discuss secure pairing protocols leveraging out-of-band (OOB) channels, as well as related mechanisms and problems relevant for our contributions. We will discuss components necessary for a pairing protocol that is both secure and efficient.

As most of the related work does, we assume we have an untrusted high-bandwidth network based on the Dolev-Yao Model [DY83]; thus an active attacker can delay, drop, and tamper with messages in any way. OOB channels provide authenticity, integrity, and — depending on the type — further properties, but are limited to a very low bandwidth. For most OOB types it is only feasible to transmit a single short string (between 16 and 32 bits) per protocol run. The pairing protocols we take into consideration transmit the data that has to be authenticated via the unsecured high-bandwidth network, while an authentication string — e.g. a digest of the transmitted data — is sent via the low bandwidth OOB channel. One way to realize such an OOB channel is displaying a user-readable representation of the digest on both devices’ displays, and asking the users to press a confirmation button on both devices, if the digests match. In some sense, this establishes a channel between the devices, which transmits a single bit that is 1, if both devices have the same pairing data, and 0 if the pairing data is different (meaning the pairing failed). We also assume that two parties that engage in device pairing trust each other.

In this chapter, we use the following terms to refer to different types of data relevant for pairing protocols.

**Pairing data:** the data the pairing parties agree upon. This data contains a shared secret (e.g. the Diffie-Hellman secret $g^{ab}$) which should never be transmitted; neither via the unsecured network, nor via the OOB channel. The protocol guarantees authenticity of the pairing data.

**Transmitted data:** the data which is transmitted via the unsecured network and acts as means for the parties to agree on pairing data (e.g. the public Diffie-Hellman keys $g^a$ and $g^b$).

**Authentication string:** a digest of either the transmitted data or the pairing data (e.g. $\text{hash}(g^a || g^b)$ and $\text{hash}(g^{ab})$, respectively), which is transmitted via the OOB channel. It is a means for verifying the authenticity of the pairing data.

**Authentication and Pairing.** In the literature, the discussed protocols are often referred to as authentication protocols (instead of pairing protocols). While authentication is the more general problem, we only use these protocols for pairing.

**Authentication** of user A to user B is the process of A proving its identity. For the discussed protocols, this process involves the transmission of data allowing the recipient to cryptographically verify the authenticity of the data.

**Pairing** of user A to user B is the process of a first time authentication in conjunction with storing mutually authenticated data; thus pairing is required to be a stateful process.
5.3 Device Pairing Background

Since all discussed protocols are suited for a first time authentication — they do not rely on any pre-established knowledge — they are suited as pairing protocols. While it is possible to use these protocols for any following authentication, it would come with the cost of unnecessary user interaction. For any further authentication, the pairing data can be leveraged in conjunction with pre-shared key authentication protocols that do not require user interaction. We consider this the far better solution for any application area where holding state (the pairing data) is feasible. This applies to most applications where users can be involved in the authentication process, as — in our opinion — today even very restricted user controlled devices can and should spare a few kilobytes to significantly improve user experience. Thus we limit our discussion to a stateful first time authentication, namely device pairing.

We further argue that the pairing process only has to authenticate transmitted data that is necessary to agree on a shared secret. Further data can be exchanged via a secured channel established via the shared secret. This is desirable as it allows keeping the pairing component simple and allows managing optional data transmission in another layer of a protocol stack.

The mechanism used to agree on a shared secret must guarantee that attackers cannot gain knowledge about the exchanged key by listening to the protocol; meaning the transmitted data must not allow any conclusions to be drawn about the pairing data. The mechanism should allow this using a single (asynchronous) message from each party, respectively. Without loss of generality, we focus on the Diffie-Hellman key exchange [DH76]; meaning we focus on authenticating public Diffie-Hellman keys. The big problem, and also the reason that makes manually authenticated pairing protocols necessary, is the fact that such key exchange mechanisms do not provide authenticity and thus are prone to man in the middle (MitM) attacks.

5.3.1 Diffie-Hellman key exchange

The Diffie-Hellman key exchange allows two parties to exchange a secret key over an unsecured connection. It is based on the hardness of the discrete logarithm problem in a finite cyclic group.

In general, the key exchange works as follows. Before the protocol starts, the two parties — we call them Alice and Bob — agree on a finite cyclic group $G$ and a generator $g$ of this group. For most applications, the key exchange is a part of a protocol — as it is in this thesis — and both $G$ and $g$ are fixed; a potential attacker also knows $G$ and $g$. Basically, $G$ can be any cyclic group in which the discrete logarithm problem is hard. The original proposal was to use a multiplicative group of integers modulo a prime number $p$; but there are also elliptic curves that serve as cyclic groups usable for the Diffie-Hellman key exchange.

Alice and Bob both generate a secret value, $a$ and $b$, respectively. These values are referred to as the Diffie-Hellman private keys. These values can either be long term keys or be generated anew for each protocol run. A protocol demanding new private keys for each run is often referred to as being based on an ephemeral Diffie-Hellman key exchange. Alice and Bob will calculate the Diffie-Hellman public keys as $g^a$ and $g^b$. The exponentiation is the corresponding operation in the chosen cyclic group. As with any secure asymmetric encryption method, it is not possible to calculate the private key given the public key using today’s available hardware. For the Diffie-Hellman keys this property is directly derived from the hardness of the discrete logarithm problem. Protocols using ephemeral
Diffie-Hellman keys might use these public keys as random values instead of generating separate random nonces.

The actual key exchange boils down to Alice and Bob exchanging their public keys. This exchange can be asynchronous, and the order of messages is arbitrary. Figure 5.2 illustrates the messages sent by Alice and Bob.

<table>
<thead>
<tr>
<th>Alice</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g^a$</td>
<td>$g^b$</td>
</tr>
</tbody>
</table>

![Figure 5.2: Diffie-Hellman key exchange. After exchanging $g^a$ and $g^b$, both parties can calculate the shared secret $g^{ab}$.

Upon receiving the keys, Alice and Bob can calculate $g^{ab}$ and $g^{ba}$, respectively, both being equal to $g^{ab}$, which is referred to as Diffie-Hellman secret. An attacker cannot solve the problem of retrieving $g^{ab}$, given the public values $G,g,g^a,g^b$. This problem is referred to as the Diffie-Hellman Problem. As of today, the best way to solve the Diffie-Hellman Problem is to solve the discrete logarithm problem.

The Diffie-Hellman secret can now be used by Alice and Bob as key material for symmetric encryption. Since the Diffie-Hellman secret is an element of $G$ and thus might be longer than typical keys and may not provide enough randomness if truncated, a key derivation function should be used to generate a symmetric key. When we use a Diffie-Hellman secret as key material for symmetric encryption, we always assume that a secure key derivation function was used. We use HKDF (HMAC Key Derivation Function) described in [KE10].

### 5.3.2 The Man in the Middle (MitM) Problem

The essential problem when using the Diffie-Hellman key exchange is the man in the middle attack shown in Figure 5.3.

<table>
<thead>
<tr>
<th>Alice</th>
<th>Eve</th>
<th>Bob</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g^a$</td>
<td>$g^{a'}$</td>
<td>$g^b$</td>
</tr>
</tbody>
</table>

![Figure 5.3: Diffie-Hellman MitM attack. While Alice and Bob think they exchange $g^a$ and $g^b$ with each other, they actually send $g^a$ and $g^b$ to Eve and receive $g^{a'}$ and $g^{b'}$, respectively, which were created by Eve. There are now two secrets, $g^{ab'}$, which Eve shares with Alice, and $g^{a'b}$, which Eve shares with Bob.

A Diffie-Hellman key exchange and subsequent encryption of the communication using the Diffie-Hellman secret protects against passive attackers. The passive attacker can
neither calculate the Diffie-Hellman secret — see the discussion above — nor can she decrypt the communication, if a secure cryptographic algorithm is used, e.g. AES. But an active attacker (Eve) can prevent Alice and Bob from establishing the secret; instead of Alice and Bob establishing a shared secret, Eve establishes substitute secrets with both Alice and Bob. This attack is possible in unsecured networks where an attacker can alter transmitted messages, e.g. a network based on the Dolev Yao Model [DY83]. In a Dolev Yao network neither Alice nor Bob know whether the received public Diffie-Hellman keys are actually the keys they intended to exchange.

5.3.3 Towards MitM Protection using OOB channels

To protect against a man in the middle attack, means for authentication are necessary. Alice needs be able to tell whether $g^b$ was actually sent by Bob or by a MitM attacker. The first methods to solve this problem were either based on pre-shared knowledge [BM92] or on a central service, e.g. a PKI [DVOW92]. But for our service discovery framework, we are interested in pairing mechanisms that do not need any prior knowledge or central entities that have to be set up and administrated. Pairing without the need for prior knowledge can be achieved using OOB channels that are authenticated by the user.

The Resurrecting Duckling approach proposed by Stajano et al. [Sta99] realizes a secure OOB channel\footnote{Different types of OOB channels are discussed in the following.} by physical contact; while this is appropriate for application areas where devices have special interfaces and preferably belong to the same user, relying on a secure OOB channel is to restrictive for our pairing component.

The Balfanz protocol [BSSW02] extends on ideas from [Sta99] and is the ancestor of the class of manual authentication protocols we are interested in. Protocols of this class create an authentication string that must be cryptographically bound (e.g. by a cryptographic hash function) to the data exchanged during pairing, but must not allow drawing conclusions about the exchanged data. Using a secure OOB channel, these requirements for the authentication string do not have to be fulfilled as — per definition — an attacker cannot access it. We use the Balfanz protocol, shown in Figure 5.4, as a basic example for manually authenticated pairing mechanisms.

![Figure 5.4: Authenticating a DH key exchange leveraging the Balfanz protocol [BSSW02]. The data transmitted via the unsecured network, in this case the two public Diffie-Hellman keys, is authenticated by users via the comparison of hash values. This protocol is called non-interactive as in order to authenticate data sent by a certain user only this user has to send messages.](image-url)
Non-Interactive and Interactive Authentication. The Balfanz protocol is a non-interactive authentication protocol, because for authenticating the transmission of user A, only messages sent by user A are involved. An authentication protocol is called non-interactive iff messages sent by the parties are completely independent. The main application area for non-interactive protocols is authenticating unidirectional messages, because as soon as mutual agreement is desired, the non-interactive condition poses an unnecessary overhead.

Figure 5.5 shows an interactive variant of the Balfanz protocol. For our application area, namely a pairing solution for privacy-preserving service discovery, interactive protocols are better suited because pairing is mutual and interactive protocols can be realized more efficiently. Interactive protocols are at least as efficient in terms of necessary messages compared to non-interactive protocols, because they may combine messages necessary for authentication. We will discuss the security of these protocols in the following subsections.

First, we want to discuss the concept of manually authenticated pairing protocols and OOB channels. In the figures of this chapter, we denote messages that are transmitted via the OOB channel with "OOB:". These messages are authenticated by the user. Our preferred way of realizing the OOB channel is displaying the hash value on the devices’ displays and asking the user to press a button if the hash values match; we will provide the rationale for this preference later in this section.

OOB Channel Types. Out-of-band (OOB) channels and their properties are crucial for manually authenticated pairing protocols. OOB channels used for these pairing protocols are assumed to offer both authenticity and integrity; thus messages cannot be forged (see e.g. [NR11b]). In the literature, OOB channels are classified by the properties they offer beyond authenticity and integrity. Certain pairing protocols may have certain requirements for these properties of OOB channels. In this thesis we distinguish weak, public, standard, stall-free, and secure OOB channels, which are defined as follows. The OOB channel types are listed in ascending order of strength; each channel type also inherits the protection properties of the previously described OOB channel types.
weak: There are no properties except authenticity and integrity. Attackers can delay, drop, and even replay messages. An OOB channel of this type could e.g. be realized by voice messages\textsuperscript{11}.
public: Attackers can delay and drop, but not replay messages.
standard: OOB channels of this type further prevent the attacker from delaying messages to another protocol session. We call this type standard as we assume it when not further specifying the type. An example would be the afore mentioned comparison of hash values that are shown on the devices’ displays.
stall-free: Messages cannot be dropped, delayed or replayed, but an attacker can ‘overhear’ the transmitted messages. An example is a face-to-face communication.
secure: In addition to being stall-free, this type of OOB channel also provides confidentiality. An example of this kind of OOB channel is connecting a cable between two devices.\textsuperscript{12}

Sometimes, channels that allow delaying messages for a fixed limited amount of time are defined (e.g. in [NR11b]).

Choosing an OOB channel type for a pairing solution involves a trade-off between usability and security. The OOB channel realizations that are most user-friendly (see Subsection 5.3.6) construct an OOB channel of the standard type. Thus to improve usability for pairing scenarios where users actually meet, choosing a cryptographic protocol that supports weak OOB channels is not a requirement. We consider cryptographic protocols that achieve security even using the weak OOB channel type as only suited for our application if the additional overhead is little to none. Stall-free or even secure channels impose a burden on users, which makes choosing a cryptographic protocol that does not require these types of OOB channels the better choice. For these reasons, we focus on standard OOB channels as building block for the pairing component of our privacy-preserving service discovery framework.

Most standard OOB channels demand users to be in close proximity to each other, and further cause a pairing to be performed in a limited time frame. Independent of the pairing protocol, this limited time frame also limits the time an attacker has to perform an attack. Some approaches even just rely on limiting attacks to a short time frame, e.g. the Bluetooth just works and the WPS push button methods, and do not use any OOB channel authentication at all. These methods are only suited for private networks. We will consider such methods for intra user pairing in trusted networks (see Subsection 5.4.7).

5.3.4 MiTM attacks against Authenticated Key Exchange

As stated above and shown in Figure 5.3, the MitM attacker tries to substitute new Diffie-Hellman public keys $g^a$ and $g^b$ for the original ones. When using authentication via manual verification, this becomes much harder for the attacker, because the public keys $g^a$ and $g^b$ indented for substitution must withstand the manual verification. Since the OOB channel provides at least both authentication and integrity, the attacker cannot modify the authentication string. She has to generate the keys $g^a$ and $g^b$ in such a way that the resulting authentication string matches the original one.

Using the Balfanz protocol shown in Figure 5.4 as an example, the function used to create the authentication string is a cryptographic hash function. To be successful, the

\textsuperscript{11} Voice messages offer authenticity and integrity under the assumption that a chosen human voice cannot be synthesized.

\textsuperscript{12} As it would go beyond the scope of this thesis, we do not consider physical layer attacks.
attacker has to find $g^a$ and $g^b$ so that, $\text{hash}(g^a) = \text{hash}(g'^a)$ and $\text{hash}(g^b) = \text{hash}(g'^b)$. This means the attacker has to find a second pre-image under the given cryptographic hash function for both $g^a$ and $g^b$. As long as the hash function is a secure cryptographic hash function, this attack is infeasible. But hash functions that guarantee second pre-image resistance given today’s hardware will produce an output of at least 128 bits, which has to be verified by users\textsuperscript{13}. It is tempting to reduce the output length, but doing so opens a pathway to MitM attacks.

Transmitting Secret Authentication Information. Before we discuss attacks on more sophisticated systems, we want to shortly take up pairing methods using a secret PIN that is instrumental in the cryptographic algorithm. Such methods are used e.g. by WEP [All14] and in Bluetooth prior to version 2.1, e.g. [C’99a]. In its worst installment, a fixed PIN is used. Once an attacker gets hold of this PIN, she can attack all future pairings of the corresponding device as if there was no authentication at all.

Generating a new PIN for each pairing brings some form of forward secrecy but is still an easy to attack method if this PIN is instrumental in the cryptographic algorithm, meaning if it is used to derive a cryptographic key. Most practical OOB channels allow attackers to eavesdrop, which in turn allows the attacker to get hold of the PIN and hijack the pairing. In contrast to the fixed PIN method, the attacker has to retrieve a new PIN for each pairing, which might be a hassle as timing is often of the essence.

As such, PIN methods are easy-to-overcome barriers for attacks that make a MitM attack a little more cumbersome but do not prevent it. Methods depending on secret PINs should be avoided, and we do not consider them any further.

Types of Attacks. We distinguish between offline and online attacks. Offline attacks are attacks that are comprised of first performing calculations on the attackers device — in most cases a combinatorial search — and then sending some result of this calculation over the network. It is called offline because almost all work is done offline on the attackers device. This kind of attack is most powerful as it is not detectable during performing the offline calculation, and the only limit is the attacker’s hardware’s computational power.

Online attacks are attacks that are comprised of a guess on the attackers device — which in most cases is just picking a value randomly — and sending the guessed value to the attackee for evaluation. The main work is done online; most of the time will be used to wait for an evaluation of a sent guessed value. The attacker cannot verify an SAS guess without sending it to the attackee. Online attacks are further subdivided in one-shot and n-shot attacks. A one-shot attack is comprised of a single send-and-wait-for-evaluation cycle while an n-shot attack is comprised of $n$ such cycles. The n-shot attack is more powerful than $n$ one-shot attacks, if the attacker can gain knowledge from one cycle and use it for a following one. A good pairing protocol should guarantee that an n-shot attack does not have a better chance of success than $n$ one-shot attacks.

The afore discussed attack on the Balfanz protocol is an offline attack as the attacker performs a combinatorial search for a second pre-image on her device. This is the reason for using a cryptographical hash function; we have to protect against a potentially very high computational power. If the attacker was only able to perform online one-shot attacks, a much shorter hash value would suffice. While for a short hash value it is easy to find a second pre-image offline (given enough computational power), finding a second

\textsuperscript{13}The users will verify a user-friendly representation of these bits. But a 128 bit hash represented as 32 hex digits is still inconvenient to compare and might stop users from actually comparing these strings.
5.3 Device Pairing Background

pre-image with an online one-shot attack, which is just a single random guess, has a success probability of $2^{-b}$, where $b$ is the bit length of the hash value and thus $2^b$ the size of the hash function’s image space; thus a $b$ as low as 16 already yields a reasonably low success probability. In practice, Alice and Bob will get suspicious after very few tries, which makes online one-shot attacks infeasible. Online one-shot attacks can be compared to trying to guess an 8 digits PIN code, where the device blocks when a wrong PIN was entered thrice. Checking all $8^{10}$ possible PINs would be a matter of milliseconds, but limiting the number of guesses to 3 yields a success probability of only $3 \times 8^{-10}$.

Since an online attack is infeasible, the attacker can only be successful if she can perform an offline attack (which depends on the available computational power). So, to prevent a MitM attack, the calculations needed for an offline attack have to be sufficiently hard, and if SASes are leveraged it is necessary to confine the attacker to online one shot attacks.\footnote{We assume SASes are reasonably long, at least $b = 16$ bit, so online one-shot attacks are infeasible to perform.}

Security Levels of Hash Functions. Before discussing how to design a pairing protocol to prevent MitM attacks, we want to discuss the different security classes of hash functions used by these protocols. It is possible to always use a hash function of the highest security class — a cryptographical hash function that provides all three of pre-image resistance, second pre-image resistance and collision resistance. But this would be disadvantageous for two reasons; first it costs unnecessary computation time, and secondly, the values that have to be compared by users become infeasibly long.

We distinguish three security classes of hash functions, whose names — longhash, hash, and shorthash — are derived from their respective output lengths. For all classes we require that the hash algorithm provides confusion and diffusion; more specifically, is should fulfill the strict avalanche criterion \cite{WT85}, which states that each bit change in the input has a 50% change on each output bit to change it. The practical implications are that (1) finding a second pre-image demands an exhaustive search of the image space, and (2) finding an arbitrary collision requires a combinatorial search that on average approximately needs to look at $\sqrt{H}$ elements, where $H$ is the size of the image space.\footnote{As this attack is based on the birthday paradox, it is also referred to as the birthday attack.} For both (1) and (2), it should not be possible to perform a more sophisticated combinatorial search further pruning the search space. In this chapter we always expect hash functions to have these properties.\footnote{In practice it is hard to mathematically exactly meet the strict avalanche criterion, but the hash function should come close enough so that attacks cannot find second pre-images or collisions significantly easier. An analysis of SHA-1 with respect to the strict avalanche criterion can be found in \cite{MI16}.}

Further, to provide a specific security guarantee for protecting against attackers that are bound to a particular computation power, the output length of a hash function has to be long enough to make an exhaustive search of the image space, or a birthday attack infeasible. A hash function guarantees 128 bits of collision resistance security, if it fulfills the strict avalanche criterion and has an output length of 256 bit.

We define the hash functions classes as follows, along with their respective output length to provide the claimed properties against attacks that are bound to today’s hardware capabilities.

longhash: provides all three of pre-image, second pre-image and collision resistance. Hash functions fulfilling these properties are also known as cryptographic hash functions.

hash:

shorthash:
To provide these properties given today’s hardware, the output of \textit{longhash} is at least 256 bit long. 

\textit{hash}: provides pre-image and second pre-image resistance; sometimes second pre-image resistance is also referred to as weak collision resistance. As of today, the output of \textit{hash} should be 128 bit long to prevent brute force attacks.

\textit{shorthash}: suffices to make online attacks infeasible. An output bit length of at least $b = 16$ bit is sufficient for most applications. Online attacks are independent of the available computational power as they are bound to the same small number of feasible tries.

Typically, \textit{longhash} is realized by a cryptographic hash function like SHA256. For realizing \textit{hash}, SHA256 values could be truncated to 128 bit, or MD5 could be used. The \textit{shorthash} could also be created by truncating the output of a cryptographic hash function. This would still preserve the much more important advantage of \textit{shorthash}, providing short authentication strings for users, but it would demand unnecessary computation time. The function used to realize \textit{shorthash} does not have to be a cryptographic hash function. Universal hash function families \cite{CW77, WC81} that satisfy the strict avalanche criterion are well suited for a realization of \textit{shorthash}. Universal hash function families fulfill the following property: Given an image space size of $2^b$ and two distinct messages $m_1$ and $m_2$, the probability for $\text{shorthash}(m_1) = \text{shorthash}(m_2)$ must be $\leq 2^{-b}$. As it would go beyond the scope of this thesis, we do not discuss different classes of hash functions with respect to the efficiency of calculating the hash values. We refer to Nguyen et al. \cite{NR12}, who discuss short hash functions based on universal hash function families as well as the design of special digest functions that are also suited for an implementation for \textit{shorthash}.

A further point we want to shortly cover regarding reducing secure hash functions to practice are length extension attacks. Hash functions that are based on the Merkle-Darmgård construction \cite{MC79}, like MD5, SHA-1 and SHA-2, are prone to this attack, which is e.g. described in \cite{KBC97}. A countermeasure is using an HMAC \cite{KBC97} instead of the plain hash function. This construct has proven to be more robust, but needs a little more than double the computation time. Wherever we use a construction like $\text{hash}(m_1 || m_2)$ we suggest realizing this as $\text{HMAC}_{m_1}(m_2)$. In order to protect $m_1$ and $m_2$ from an exhaustive search for either $m_1$ or $m_2$ yielding the same result, both $m_1$ and $m_2$ should fulfill the length requirements previously stated for hash values. A further possibility is using a cryptographic hash function that prevents this attack, e.g. SHA-3.

\textbf{Chosen Plaintext Attack.} As previously mentioned, we consider a setting where the unsecured broadband network is based on the Dolev-Yao model \cite{DY83}. This model involves a very powerful attacker that basically "carries the messages". Many papers analyzing MitM attacks on the Balfanz protocol \cite{BSSW02}, e.g. \cite{MS07, NR11b}, consider an attacker that is additionally capable of performing chosen plaintext attacks. This means the attacker is able to make Alice send chosen messages to Bob and vice versa.

When using a weak OOB channel, a chosen plaintext attacker can run a computationally more efficient attack on the Balfanz protocol. This attack comprises

- Eve finding a collision pair $g^e$, $g'^e$, so that $\text{hash}(g^e) = \text{hash}(g'^e)$,
- making Alice send $g^e$ to Bob, which will cause
- Alice to send $\text{hash}(g^e)$ via the OOB channel.

\footnote{There are collision attacks against MD5 but it is still second pre-image resistant as of today.}
5.3 Device Pairing Background

- Eve will then drop the message transmitted on the OOB channel\textsuperscript{18}.
- Thereafter, Eve will initiate a new pairing session\textsuperscript{19} with Bob and sends $g^e$ and $\text{hash}(g^e)$ over the unsecured network and the OOB channel, respectively.
- Bob accepts because the hashes match.

This attack can be performed computationally more efficient because the attacker does not need a second pre-image but any collision suffices; thus she can run a combinatorial search often referred to as birthday attack. To protect against this attack a \textit{longhash} function is necessary. In our scenarios, we consider the chosen plaintext attacker as being unrealistically powerful; nevertheless we consider awareness of this problem as important.

**Attacks on Interactive Protocols.** For interactive protocols a single authentication string suffices to verify the transmitted data of both parties. Both Alice and Bob contribute to the input of the function used to generate the authentication string. This also allows an attacker to choose a part of the input. When attacking our interactive version of the Balfanz protocol, the attacker Eve can choose $a'$ and $b'$ so that $\text{hash}(g^{a'b'}) = \text{hash}(g^{a'b})$. Thus Eve does not need to run an exhaustive search but — similar to the afore described chosen plaintext attack — can run a combinatorial search for collisions, which demands the hash output to be 256bit in order to get 128bits of security.

**Attacking an Efficient "Secure" Pairing Protocol.** Before we discuss secure and efficient pairing protocols and their requirements, we want to propose an attack on a recently published pairing protocol [NL15] claimed to be more efficient compared to other protocols and as secure; the formal security proof contained in the paper misses an important part. Figure 5.6 illustrates the protocol. To the best of our knowledge, our attack is the first attack proposed on this protocol. Further, our attack is very practical and almost as easy as a MitM attack on an unauthenticated Diffie-Hellman key exchange.

![Figure 5.6: Pairing protocol recently proposed in [NL15].](image)

$r_a$ and $r_b$ are random values generated by Alice and Bob, respectively. Since $r_a$ is transmitted via the OOB channel, and the protocol is designed to be efficient, the length of $r_a$ corresponds to a typical SAS length. $h$ is a cryptographic hash function and $h_{r_a}(g^a, g^b)$ is a keyed hash function providing a security level similar to our \textit{shorthash} function. The protocol is claimed to secure by a formal proof; but our attack makes a MitM attack almost as easy as attacking an unauthenticated Diffie-Hellman key exchange.

\textsuperscript{18}She can only perform this on a \textit{weak} OOB channel. Instead of dropping the message transmitted via the OOB channel, she could also delay it to the protocol session in which she attacks. Delaying to other protocol session is also only possible on \textit{weak} OOB channels.

\textsuperscript{19}In the other (more common) attacks we describe, the attacker hijacks an existing pairing session instead of starting a new one.
In this protocol, \( r_a \) and \( r_b \) are random values generated by Alice and Bob, respectively. These random values are necessary, because the authors act on the assumption that the public Diffie-Hellman keys \( g^a \) and \( g^b \) are long term keys. Since \( r_a \oplus h_{\text{shorthash}}(g^a, g^b) \) is a short authentication string (SAS) of length \( b \), the length of \( r_a \) is also \( b \). To be efficient in terms of user interaction, \( b \) has to be in range of 16 to 32 bits. The strength of the hash function \( h \) is irrelevant for our attack. The output length of the keyed hash function \( h_{\text{shorthash}} \) is also \( b \) to provide an optimal SAS, which yields a security level similar to our \textit{shorthash} function.

To verify the exchange, Bob first retrieves \( r_a \) from the authentication string. He is able to do so because he knows \( r_b, g^a, \) and \( g^b \), which allows him to calculate \( h_{\text{longhash}}(g^a, g^b) \), and finally to retrieve \( r_a = h_{\text{longhash}}(g^a, g^b) \oplus (r_a \oplus h_{\text{shorthash}}(g^a, g^b)) \). Bob can then perform the verification by checking whether \( h(g^a, r_a) \) matches the hash he received in the first message. The proof in [NL15] states that based on the fact that the attacker is committed to \( r_a \) before he knows \( r_a \), the attacker needs to find \( g^{a'} \) and \( g^{b'} \) fulfilling

\[
h(g^a, r_a) = h(g^{a'}, h_{\text{longhash}}(g^{a'}, g^b)) \oplus (r_a \oplus h_{\text{shorthash}}(g^a, g^{b'})).
\]

### Algorithm 1: Exhaustive search retrieving \( r_a \) from \( \text{hash}(g^a, r_a) \).

**Data:** \( g^a, h = \text{hash}(g^a, r_a) \), \( K \)

**Result:** \( r_a \in K \)

1. for each \( k \in K \) do
   2. if \( h = \text{hash}(g^a, k) \) then
   3. \( r_a = k; \)
   4. return \( r_a; \)
   5. end
6. end

However, the assumption the proof is based on is wrong. The attacker can retrieve \( r_a \) after intercepting the first message, because this protocol does not confine the attacker to an online attack. Since the attacker has access to \( g^a \), she can perform the offline attack shown in algorithm 1, retrieving \( r_a \) with at most \( b \) applications of a \textit{longhash} function. Having access to \( r_a \), Eve can choose an arbitrary \( g^{a'} \) and send \( g^{a'}, h(g^{a'}, r_a) \) to Bob. This substitution will not be detected, because the verification only checks whether the public Diffie-Hellman key in the first message to Bob (in this case \( g^{a'} \) was hashed along side the original \( r_a \). For the message to Alice, Eve has to choose \( g^{b'} \) and \( r_{\nu} \) so that \( h_{\text{longhash}}(g^a, g^{b'}) = h_{\text{longhash}}(g^{a'}, g^b) \), which corresponds to an offline search for a second pre-image of a \textit{shorthash} function’s image. Alice will use \( g^{b'} \) and \( r_{\nu} \) (which have been chosen by Eve) when generating the SAS.\(^{20}\) Because of Eve’s choice of \( g^{b'} \) and \( r_{\nu} \), Bob will now retrieve \( r_a \) from the SAS and confirm the (forged) hash sent in the first message.

### 5.3.5 MitM-proof Pairing and Optimized Verification Overhead

For discussing the design of a secure and optimized device pairing protocol, we will use insights gained from the foregoing discussion.

\(^{20}\)Eve cannot run a collision attack, because \( g^a \) is fixed. This is the case because Alice chose \( a \) as her private Diffie-Hellman key, and will use \( g^a \) in the calculation of the SAS.
5.3 Device Pairing Background

**Requirements.** In order to be both secure and user-friendly, we want the protocol that we use as basis for our pairing mechanism to meet the following requirements.

*SAS via OOB:* It must suffice to transmit a short authentication string (SAS) that was created by a short hash function with an output length of \( b \) bit, where \( b \) is between 16 and 32. The SAS (as the names says) is only used for authentication; neither is it allowed to be the secret, nor can the secret be derived from it.

*Single SAS:* Transmitting a single SAS must suffice for mutual authentication.

*SAS Protection:* An offline attack that breaks the protocol by an exhaustive search on the authentication string must be impossible. The attacker must be confined to online one-shot attacks on the authentication string.

*Optimal SAS:* For a \( b \)-bit long authentication string, the probability for a successful one-shot attack must be \( \leq 2^{-b} \).

*Offline Protection:* An offline attack on the pairing protocol must be infeasible by means of strong cryptographic operation, e.g. a cryptographic hash function with an \( l = 256 \) bit long output.

*Standard OOB:* The protocol must be secure when using a standard OOB channel.

*Minimal Messages:* Further, it should minimize the number of messages sent via the insecure network.

While *Offline Protection* in conjunction with a long authentication string suffices to construct a secure pairing protocol (see e.g. the Balfanz protocol [BSSW02]), the other requirements are necessary to make the protocol user-friendly while maintaining security. The *Single SAS* requirement demands that our underlying protocol has to be interactive.

Many secure pairing protocols have been proposed (see Section 5.2), but many of them do not fulfill all of these requirements. The Balfanz protocol [BSSW02] does not fulfill SAS via OOB, as it demands transmitting a long hash value via the OOB channel to guarantee security. The MANA protocols proposed in [GMN04] transmit short strings via the OOB channel, but the SASes are not optimal; further, the protocols demand a stall-free OOB channel. Both the Hoepman [Hoe04, Hoe05] and Wong-Stajano [WS05, WS07] protocols violate the *Single SAS* requirement.

**On Designing Protocols Fulfilling All Requirements.** To begin our discussion on designing protocols that fulfill all requirements, we discuss the *SAS over OOB* and *Single SAS* requirements and their implications. The SAS via OOB requirement restricts the length of an SAS to \( b \) bits, where \( b \) is between 16 and 32 bit. For security reasons the SAS must not contain secret information. It may only contain information that allows authenticating the transmitted data. Without loss of generality, we assume the transmitted data to be \( g^{a}, g^{b} \) in the following discussion.

Because secure \( g^{a} \) and \( g^{b} \) are much longer than the SAS, the SAS can only be used to authenticate a compressed representation of \( g^{a}, g^{b} \). This compressed representation can e.g. be realized by a short hash function, or by a keyed hash function, e.g. an HMAC with a short output. Since the realization of the compression function is part of the protocol, and thus known by the attacker, regardless of how the compression is realized, its result is short enough to easily find a second pre-image; and because both parties contribute to the transmitted data, it even suffices to find a collision. In summary, while the SAS itself is

\[21\] While designing the underlying cryptographic protocol, user-friendliness refers to minimizing the length of the authentication string that has to be transmitted via the OOB channel.
authentic, it is very easy for a MitM attacker to generate substitute $g^a', g^b'$ with the same SAS representation.

To prevent an offline search for matching $g^a'$ and $g^b'$, another verification in conjunction with checking the SAS is necessary. While the SAS is authentic, the second verification protects against exhaustive search, and thus is means to fulfill the Offline Protection requirement. The second verification is performed with the help of a longer compressed representation of $g^a, g^b$, which does not allow drawing conclusions about $g^a$ and $g^b$; it can e.g. be realized by a longhash function, an HMAC, or a cryptographic commitment scheme\textsuperscript{22}. We refer to this representation of the transmitted data as offline protection string. The offline protection string must be transmitted via the unsecured network as the OOB is limited to a single SAS per protocol session. Since the SAS is authentic and the offline protection string is immune to offline attacks, both these representations must be intertwined to gain both properties; such that a possible attacker is forced to bruteforce the offline protection string before she can find $g^a'$ and $g^b'$ matching the SAS. In other words, two riddles have to be solved to find matching substitutes $g^a'$ and $g^b'$. This intertwinedness can be realized by the timing of messages in an interactive protocol: Alice first sends the offline protection string (e.g. longhash($g^a$)) to Bob, waits for Bob to send $g^b$, and then sends $g^a$. This makes it necessary for the attacker to choose $g^a'$ before she knows $g^a$. Because of this, she cannot easily find a $g^a'$ which will yield the same SAS as $g^a$. She has to find a $g^a'$ that matches both the offline protection string and the SAS.

For protecting the authenticity of Bob’s data ($g^b$) in an interactive protocol, the offline protection string generated by Alice can be leveraged. It suffices for Alice to generate the SAS as shorthash($g^a||g^b$).\textsuperscript{23} The attacker knows $g^b$, but since she does not know the $g^a$ she is committed to, Eve cannot perform an offline search for a fitting $g^b$. The asymmetry of only Alice generating the offline protection string based on her Diffie-Hellman key might lead to the wrong conclusion of Bob getting more security about Alice’s identity than vice versa. This is not true, because pairing only makes sense if Alice and Bob trust each other;\textsuperscript{24} thus Bob does not have to question the authenticity of longhash($g^a$). The goal is guaranteeing that both Alice and Bob receive authentic Diffie-Hellman public keys. If they had evil intentions, they could e.g. publish the secret after pairing, which cannot be prevented.

All the protocols that fulfill the Offline Protection, SAS via OOB, and SAS Protection requirements share important properties discussed above.

- They use different means to protect against offline combinatorial search attacks and online one-shot attacks [NR09b],
- and they are based on the "commitment before knowledge" principle [NR11b], which says that a potential attacker must be committed to the transmitted data sent by Alice before the attacker knows this data. The above discussed second verification task plays the role of this commitment.

Adding the rest of the requirements, a single optimal SAS is always sufficient if offline attacks are not possible. The optimal SAS must fulfill the requirements we stated for the

\textsuperscript{22}Meant is a commitment scheme in the classical cryptographic sense with commit and decommit operations; this should not be confused with the "commitment before knowledge" principle, which can also be realized in other ways.

\textsuperscript{23}For efficiency reasons $g^a \oplus g^b$ or $g^{ab}$ can also be used. We suggest using $g^{ab}$ as it both is secret and has to be calculated anyway.

\textsuperscript{24}At least, their trust level should be higher than towards an arbitrary entity.
shorthash function, meaning random guesses on the SAS are the best possible attack; using a combinatorial algorithm pruning the space of possible SASes must not be possible. A standard OOB channel suffices, because even if the attacker can see the SAS and the SAS is delayed within the same protocol session, the attacker is still confined to one-shot online attacks on the SAS and an infeasible exhaustive search on the offline protection string. The SAS is not a key, it is just a representation the transmitted data has to match. The minimal messages requirement is also fulfilled by a pairing protocol designed as described above. A secure manually authenticated pairing protocol needs at least three messages over the unsecured network and a single SAS verification over the OOB channel (which can e.g. be modeled as Alice sending the SAS and Bob replaying with a 1-bit acknowledgment). Sending the offline protection string and $g^a$ in the same message is not possible because calculating a matching $g^{a'}$ knowing $g^a$ has to be prevented while only 32 bits of data can be transmitted via the authenticated channel. To prevent such a calculation, 128 authenticated bits need to be instrumental in the calculation of the offline protection string. Using the SAS to transmit a key that was instrumental in calculating the offline protection string will always yield an insecure protocol. Approaches that try to minimize the number of unauthenticated messages even further, like the one proposed in [NL15], can be attacked as described in algorithm 1.

While the two intertwined verification tasks can be realized in multiple ways, we focus on methods that directly bind both the offline protection string and the SAS to the transmitted data whose authenticity has to be guaranteed. Direct binding methods are both easiest to reenact and most efficient [NR11b]. Indirect binding methods based on a commitment scheme are proposed in [Vau05, LN06, NR11b].

For the rest of this chapter, we mainly focus on the improved Hoepman protocol [NR11b], illustrated in Figure 5.7. It follows the afore discussed design principals, fulfills all requirements and is both efficient and simple.

Before ending the discussion of secure manually authenticated pairing protocols, we want to point out the fact that the confinement of the attacker to one-shot attacks is also crucial for Password-Authenticated Key Exchange (PAKE) protocols, e.g. the EKE [BM92], as it allows using insecure passwords to build a secure protocol. PAKE protocols realize this confinement using a zero-knowledge proof; the parties proof to each other that their passwords match without sending any information about the password via the network.
5.3.6 Realization of the OOB Authentication Channel

Besides defining a secure and efficient cryptographical protocol, the realization of the OOB channel used for authentication is also crucial. In this thesis, we focus on OOB channels where the human user plays a major part in verifying the transmitted data. Further, we focus on protocols where the data transmitted over the OOB channel is an SAS.

We classify a concrete realization of an SAS verification with respect to the two dimensions SAS representation and verification method.

SAS Representation. A list of possible SAS representations can be found in [KFR09, UKA07, KSTU09b, KST’09, USK11]. Straightforward representations are numeric and alphanumeric strings.

The SAS can also be represented by real language words [KFR09]. Thereby, the SAS is subdivided into parts where each part maps to a word in a predefined list of $w$ words. The list of words can also be seen as digits of a base-$w$ number system and the SAS is represented in this base-$w$ system.

Another possibility to represent SASes is mapping them to real-language-like sentences. As described in [GSS’06], these sentences can either be shown on a display or read by a speech-synthesizer. The disadvantage inherent in all word and sentence based approaches is its being only comfortable for users that have sufficient command of the respective language.

Since humans are exceptionally good at recognizing images, methods that map an SAS to an image have been proposed and evaluated (e.g. [KFR09]). A more exotic way is representing the SAS as melody, which is also discussed in [KFR09].

Further methods of representing the SAS are QR codes and bar codes. These representations are an exception as they are not human interpretable, and thus are only suited for verification methods where one of the devices scans and interprets the SAS representation. A further peculiarity of the QR code representation is its feasibility to transmit more data than the other representation methods; thus it is possible to transmit a longhash and realize a secure manually authenticated pairing protocol that is comfortable without the need for the SAS length requirement. However, we do not utilize this advantage, because (1) we want a modular design where the secure pairing data exchange and the manual authentication are independent, and because (2) saving a single message over the high bandwidth broadcast network only yields a minuscule efficiency boost. Nevertheless, this might be interesting for specialized scenarios.

Verification Methods. Different SAS representations can be used in combination with a verification method in order to realize an OOB channel. But not all SAS representations are feasible to use with every verification method. Mainly, four kinds of verification methods are discussed in the literature (see e.g. [KFR09]).

Compare & Confirm: This verification method asks users to compare an SAS representation, which is displayed (or transmitted to the user in another way, e.g. audio) on both devices that are about to be paired. If the SAS representations match, the users should confirm the match by pressing a button$^{25}$. All the SAS representations discussed above are feasible for this method. While this method is very convenient in general, it comes with the inherent disadvantage that users can simply ignore it by pressing 'OK' without comparing.

25 Nowadays, this button might very well be a virtual button shown on a touchscreen.
5.3 Device Pairing Background

*Compare & Select:* This method shows an SAS representation on one device and a short list of candidates on the other device. Users must select the matching candidate in the list. For representing the SAS, strings and words are feasible.

*Copy & Enter:* This method asks the users to get an SAS representation from one of the devices and enter it in the other. Representations that minimize the amount of data the user has to input are best suited, e.g. alphanumeric strings. When users want to pair devices without giving both devices to one user, the word and sentence variants are also suited.

*Scanning:* This method asks the user to scan an SAS representation shown on the other device. This method is an exception, as the SAS representation does not have to be interpretable by humans. The QR code and bar code representations are made for this verification method.

Studies [KFR09, UKA07, KSTU09b, KST+09, USK11] showed that the *compare & select* method is most convenient. However, Kainda et al. [KFR09] argue that as best trade-off between usability and security the *copy & enter* method should be preferred. Despite the fact that the studies are more than five years old by the time of this writing, we consider the insights as still valuable. But we assume that the QR code method would perform much better today as almost everyone owns a device supporting it and popular Apps like Line\(^\text{26}\) and Signal\(^\text{27}\) use it.\(^\text{28}\)

All these methods realize an OOB channel of the above described *standard* type; the realized channel grants both authenticity and integrity, and further prevents attackers from replaying messages or delaying messages to another protocol session. Users have to meet in person for the device pairing and further manually transfer information between the devices. While for the *compare & confirm* and *compare & select* methods the human users transfer the 1-bit information of an SAS match and mismatch, respectively, the human user transfers the whole SAS for the *copy & enter* and *scanning* methods. In both cases, the transmitted information is authentic as users see each other during the procedure. Replaying and delaying to another protocol session are not possible, because users do not run multiple protocol runs at once, see each other during the whole procedure, and finish a single pairing session without interrupting it. However, blocking and delaying within the session are possible, as an attacker could e.g. start talking to the users as soon as the string to be compared is displayed; i.e. delay the transmission of the confirmation bit.

**Pairing Methods for Interface Constrained Devices.** We also want to shortly cover pairing methods for restricted devices, meaning devices that e.g. lack a display and thus cannot leverage the pairing methods described above. The afore mentioned methods that open a short pairing time window in which devices accept any incoming pairing request without further authentication is one possibility. Examples are the Bluetooth *just works* and WPS *push button* methods. Except for limiting the time in which an attack can be performed, no further protection against MitM attacks is provided.

These pairing methods provide sufficient security if they are only used in trusted networks, e.g. at home, and further, access to the broadcast network is either limited to trusted devices, e.g. WiFi using WPA2, or the range in which the network is accessible is small enough (distance bounding), e.g. Bluetooth LE.

\(^{26}\)http://line.me

\(^{27}\)https://whispersystems.org

\(^{28}\)Even if Line uses it for a different purpose, namely adding new contacts to the contact list, it familiarizes users with the QR code scanning process.
Further, using a cable to pair interface constrained devices is the most secure option, and might be acceptable for intra user pairing. Typically, today’s game consoles offer this option for pairing Bluetooth controllers.

A more secure button enabled device pairing method is proposed in [STU07], and discussed, among other pairing methods, in [KSTU09b]. This method involves pressing buttons on both devices a few times in synchrony. A method that is only based on audio signals is proposed in [STU08]. Soriente et al. propose further secure device pairing methods for interface constrained devices in [STU09].

5.3.7 Manually Authenticated Certificates

If desired, manually authenticated pairing protocols can be used for manually authenticating existing public keys or public key certificates (e.g. self-signed X.509 certificates [CSF+08, Yee13]). We do not use certificates for our pairing solution described in Section 5.4, but as our solution can be easily adapted to support them, we shortly describe two ways of realizing the establishment of such certificates.

Additional: Alice and Bob simply exchange their certificates after the actual pairing process via a secure channel established with the help of the authenticated pairing secret (see Figure 5.8). At the cost of additional messages, this realization allows for a clean separation of the optional establishment of certificates. The actual pairing and the establishment of certificates can be realized in distinct layers.

Figure 5.8: Improved Hoepman protocol [NR11b] with a subsequent transmission of further pairing data, e.g. certificates, via a secure connection established using the afore exchanged secret $g_{ab}^\ast$. 

124
5.4 A User-friendly Pairing Solution

Integrated: Alice and Bob use the pairwise authentication scheme in Vaudenay’s style [NR11b], which allows incorporating the certificates (or any further data that should be authenticated) to be transmitted within the actual pairing messages (see Figure 5.9).

![Figure 5.9: Pairwise authentication scheme in Vaudenay’s style [NR11b]. Randomness is injected by using the two nonces \( r_a \) and \( r_b \). This authenticating scheme can be used to transmit authenticated data beyond public Diffie-Hellman keys, e.g. certificates. Instead of the nonces, randomly chosen ephemeral public Diffie-Hellman keys could be used to additionally establish an authenticated shared secret.]

5.4 A User-friendly Pairing Solution

In this section we present our pairing solution, which is a basis for our pairing daemon (see Section 5.5), and thus in turn is a basis for our privacy-preserving service discovery framework. Further, the pairing solution proposed in this section is a basis for our IETF draft on device pairing [HK16a]. We cover all parts necessary to realize a pairing system, and present an implementation in Section 5.6. For all three components we especially take privacy into consideration; we do not want identities of pairing parties or device identifiers to be disclosed during pairing, as e.g. the Bluetooth pairing mechanism [Blu07b] does.

We split our pairing solution in three separate components discovery, secure pairing data exchange, and manual authentication (see Figure 5.10, which is a copy of Figure 4.2). Since the mechanics of these components are independent, each of these components can be realized in an arbitrary way, as long as the following interfaces are provided. The discovery component provides network parameters, e.g. the IP address and the port, allowing the secure pairing data exchange component to establish a connection between devices that are about to be paired. The secure pairing data exchange component generates an SAS which is handed to the manual authentication component. The manual authentication component generates an SAS representation, asks the user to verify the SAS, and returns the result of the verification.

We distinguish two kinds of pairings, namely inter user pairing and intra user pairing; the first is for pairing of devices belonging to separate users, the latter for pairing among devices of the same user. This distinction is advantageous in several ways. It allows specifying different authentication methods depending on the kind of pairing, relaxing
security requirements for *intra user pairing* in trusted networks (e.g. the home network), and — especially increasing convenience — making a single *inter user pairing* between a pair of users sufficient, even if these users have several devices (see Subsection 5.4.6).

### 5.4.1 Discovery

A very important part of device pairing, which we did not cover so far, is discovery. Before devices can engage in pairing and use one of the afore discussed pairing protocols, they have to be able to contact each other via the unsecured broadband network. Our solution solves this via service discovery. For achieving privacy, we cannot use our privacy extension because the privacy extension itself depends on an established pairing. To overcome this problem, we use another step of user interaction using an OOB channel to agree on private one-time identifiers. This step is necessary, because private discovery cannot be fully automatic without shared knowledge; further, it is not possible to combine the user interactions for discovery and authentication while guaranteeing commitment before knowledge.

**General Description of our Solution.** Basically, our private pairing discovery component works as follows.

- The user of device B initiates the pairing process, e.g. by pressing a button in a pairing application (see Section 5.5), while the user of device A prepares her device
5.4 A User-friendly Pairing Solution

for pairing, e.g. by launching a pairing application. This step can be omitted, if the pairing application is always active.

- Device B then generates a 32 bit nonce, publishes it (along with its network parameters) via the unsecured network, and presents it to its user. For the presentation, SAS representation methods discussed in Section 5.3 can be used.
- Device A automatically discovers the published nonce and presents all nonces it discovers in the current network to its user.
- The user of device B communicates the displayed nonce to the user of device A.
- The user of device A selects the matching nonce among the nonces presented by her device, which causes device A to send a pairing session initiation request to the device with the network parameters associated with this nonce. The contents of the session initiation request are part of the Secure Pairing Data Exchange component.
- As soon as device B receives a session initiation request (which is handled by the Secure Pairing Data Exchange component), it stops publishing the nonce associated with this pairing in order not to pollute the lists shown on other devices currently engaged in the discovery step. Device B will not accept any further initiation requests until a new pairing session is requested by its user, which prevents the replaying of authentication data to the next pairing session. This renders parallel pairing requests infeasible, but as the pairing involves pairwise user interaction, it is more convenient for users to serialize pairings anyway.

The device whose user started the pairing process is not the device who initiates the cryptographic protocol used in the Secure Pairing Data Exchange component. For realizing this discovery method, the compare & select method is well suited.

Leveraging DNS-SD/mDNS. The proposed general solution can be implemented with any service discovery solution. Since the main service discovery protocol used in this thesis is DNS-SD/mDNS (for reasons discussed in Chapter 3), we also use it as basis for the discovery step during device pairing (see Figure 5.11).

We introduce a new service type for pairing: _pairing_.tcp. The nonce that acts as a one time identifier is published as instance name of a _pairing_.tcp service instance in base64 encoded format. Alice will get Bob’s network parameters by resolving the service instance with the instance name matching the string displayed on Bob’s device. While typically Bob uses an instance name that is interpretable by Alice and Alice selects service instances to be resolved, in this case Bob uses a random instance name and selects the service instance Alice should resolve (his) by transmitting the necessary information via the OOB channel.

Privacy Protection During the Discovery Phase. To protect the privacy during the discovery phase, both parties must use a randomized hostname in the A resource record published during discovery. The randomized hostname should be chosen according to [HTW16]. Alternatively, the service instance name, which is a randomly chosen number, can be substituted for the current hostname. Besides randomized hostnames, devices must use MAC address randomization. We discuss both MAC address randomization and random hostnames in Chapter 6. Further, the nonces — like the name says — must only be used once for a single run of the pairing protocol.
Figure 5.11: Our DNS-SD based *pairing discovery* protocol. Before this protocol begins, Bob presses an 'initiate pairing' button in his pairing application which causes his device to publish the service instance to be discovered by Alice. If Alice’s pairing application has already been active when Bob publishes his instance, she will immediately get the desired information during the probing phase of Bob and does not have to query. (Analogous to the zone file syntax of Bind9, in this thesis we abbreviate domain names substituting @ for the origin if the origin is obvious in the respective context; see Chapter 3.)
5.4 A User-friendly Pairing Solution

Despite these means of protection, adversaries might have visual contact to the parties engaged in pairing, and might infer the fact that they are performing device pairing from typical user behavior.

**Leveraging QR codes.** The discovery process, like the afore discussed manually authenticated pairing protocols, uses an insecure network to transmit the actually desired data and an OOB channel to transmit meta data necessary for the protocol. But there are important differences. While the security of the cryptographic pairing protocol depends on the guaranteed authenticity of the OOB channel, the discovery can deal with tampered OOB data. Further, from an abstract viewpoint, data has to flow only in one direction during discovery; it suffices if Bob can communicate his network parameters to Alice.

These facts make exploiting the peculiarities of the QR code based OOB channel worthwhile, as it suffices to encode all necessary data, e.g. the IP address and the port, in the QR code. This would not be feasible for the cryptographic pairing protocol, as it would demand transmitting secret data via a standard OOB channel.

5.4.2 Secure Pairing Data Exchange

We discuss three different methods to realize the secure pairing data exchange layer; one for proof of concept and prototypical implementations and the other two for real world implementations. We do not consider group pairing methods as we allow mapping paired devices to groups via a user interface component discussed in Chapter 4 and Section 5.5.

**Ad-Hoc Protocol**

A very efficient and (theoretically) secure way to realize the secure pairing data exchange layer is to directly implement the improved Hoepman protocol (see Figure 5.7).

While this solution is very efficient and theoretically secure, it comes with the disadvantage of "implementing one’s own crypto", which often leads to faulty implementations breaching the security of the built system.

**Based on the TLS Handshake**

Among our proposed solutions, the TLS based solution is the most practical one as it leverages the TLS handshake [DR08] including the definitions of cryptographic primitives. It also inherits other benefits like session resumption (which has been merged with PSK in TLS 1.3 [Res16]). Further, it neither depends on altering TLS nor on altering any other protocol. It is also the basis for the current version of our DNS-SD device pairing Internet draft [HK16a]. In a nutshell, this solution

1. establishes a TLS connection using a cipher suite with an unauthenticated ephemeral Diffie-Hellman key exchange (TLS_DH_anon*),
2. exports the established secret via a TLS exporter [Res10], and
3. authenticates the established secret independent of TLS (which is part of the manual authentication layer).

---

29 In case of tampered data, the following secure pairing data exchange will fail and the pairing process has to be restarted. While this would be uncomfortable, it would not breach security.
In doing so, we detach the key exchange messages from the messages for transmitting the data to be authenticated. To guarantee an authentication of the secret, we involve it in the calculation of both the offline protection string and the SAS. Figure 5.12 illustrates our protocol. Since the protocol is independent of the details of the TLS handshake, it will also work with TLS 1.3 [Res16][30].

**Figure 5.12:** Our TLS based authenticated pairing data exchange protocol. For the sake of an easier explanation, we reduced the TLS handshake to the ephemeral Diffie-Hellman key exchange. The full TLS handshake is described in [DR08] and [Res16] for TLS 1.2 and 1.3, respectively. We use two random values \( r_a \) and \( r_b \), which are offline protected by \( \text{longhash} \) and generated anew for each protocol run. Alice first commits to \( r_a \) using \( \text{longhash} \) and sends her random value only after receiving Bob’s random value \( r_b \). In the SAS, these random values are bound to the secret \((g^{ab})\) extracted from the TLS connection. Attackers are confined to online one-shot attacks on the SAS.

Because our goal is to not alter TLS for this solution and to initiate the connection via TLS, it is not possible to commit a possible attacker to \( g^a \) before she knows \( g^a \). This commitment however, is crucial for preventing an offline attack on the SAS. If we would not inject the random values \( r_a \) and \( r_b \) to generate randomness, an attacker could perform an offline attack on the SAS via a combinatorial search for \( g^{a'} \) and \( g^{b'} \), so that \( \text{shorthash}(g^{ab'}) = \text{shorthash}(g^{a'b}) \). Transmitting \( \text{longhash}(g^{ab}) \) after \( g^a \) and \( g^b \) makes no sense at all, because

---

[30] If the final version of TLS 1.3 does not offer an anonymous DHE mode of operation, a certificate-based mode can be used without checking the certificate to simulate the anonymous behavior.
5.4 A User-friendly Pairing Solution

the attacker could just substitute longhash\( (g^{ab}) \) — the recipient does not know what the real \( g^b \) is before it is authenticated.

The protocol can be made secure by injecting offline protected randomness into the SAS. This is very similar to the Vaudenay’s style protocol (see Figure 5.9). Any previously known value can be authenticated in that way, because all data that is used as input for shorthash combined with random values that are offline protected by a longhash commitment before knowledge is authentic as long as the random values are unpredictable and have the same length as the output of a cryptographic hash function.

A thorough definition of the messages transmitted by our protocol in steps two and three (meaning after the secret has been extracted from the TLS context) can be found in our DNS-SD pairing draft [HK16a].

Integration into TLS

The most seamless realization is an integration of the manually authenticated pairing process into TLS. To this end, we are reviving and updating the Internet draft on short authentication strings for TLS [MGR14].

Even if the pairing process is integrated into TLS, our DNS-SD based pairing mechanism does not become obsolete, as the transport layer does not handle discovery and cannot address application specific considerations. Since the integrated pairing mechanism operates on the (secure) transport layer, it rather acts as a basis for realizing a pairing solution, making pairing even more convenient; especially for developers. We plan to leverage TLS session resumption (and PSK for TLS 1.3, respectively) for managing a pairing context. This, however, will demand softening the client/server concept in TLS, as we want both parties to be able to initiate the reestablishment of a connection. Even when full TLS integration is realized, the pairing daemon will not become obsolete as, in contrast to the pairing daemon, TLS will just manage per-connection states. The pairing daemon still has the task of managing pairing data for connections of various applications (i.e. cross-connection state).

5.4.3 Manual Authentication

Besides security, the main concern for the manual authentication layer is usability. Among the methods discussed in Section 5.3 we chose (1) compare & confirm with both alphanumerics strings and sentences as SAS representation and (2) QR code scanning.

For inter user pairing we focus on pairing methods for devices that have a display. Because of the proliferation of smartphones, we assume that almost every user has at least one device capable of performing one of these two pairing methods. A single inter user pairing per pair of users — which is performed with a device that has the appropriate interface — suffices, because we also provide pairing data synchronization (Subsection 5.4.6) and our automatic intra user pairing method. This allows establishing a pairing graph among the own and friends’ devices as the user desires, without the need for an appropriate interface on every device, and without the need for uncomfortable inter user pairing methods.

Compare & Confirm. Besides the fact that compare & confirm methods were the most user-friendly methods in the user studies discussed in Section 5.3, the method is also used by Bluetooth [Blu14] making it well known among users. We mainly focused
on the alphanumeric SAS representation, but we also take sentences comprised of real language word into consideration. The sentence representation was determined to be the most user-friendly pairing method for situations where users keep their devices in their hands while pairing [USK11]. However, we consider the alphanumeric representation to be superior, as for the sentence representation users need to have sufficient command of the language the words are chosen from, a fact not addressed in the study.

Without loss of generality, we assume an alphanumeric representation for the following description of how to realize the manual authentication layer using the compare & confirm method. The discovery and manual authentication layers both need manual interaction. In order not to confuse users, we stick to the same SAS representation for both interactions. During the discovery phase the user that did not initiate will be confronted with a compare & select task to choose the corresponding service instance. After the secure data exchange, both devices will show the SAS representation; the users compare these representation and, if they match, press a confirmation button in their pairing applications. This completes the pairing process.

**QR Code Scanning.** QR code scanning was classified as not being user-friendly in the user studies we discussed in Section 5.3. We believe this is mainly due to the fact that QR code scanning applications were not popular yet at the time the studies were conducted (2009-2011). Besides the wide spread QR code scanning capability among today’s devices, popular applications, e.g. the instant messaging application Line\(^{31}\) uses it for adding friends to the contact list, and the security focused instant messaging application Signal\(^{32}\) uses it for key fingerprint comparison. These wide spread applications make users familiar with QR code scanning. Our opinion on QR code scanning was also supported by an informal user study asking several people, especially with a non-technical background, if they are familiar with QR code scanning and what they think about it compared to other methods. We plan to conduct a real user study in future work.

We use QR code scanning for both service instance confirmation during discovery and SAS verification during the secure data exchange. To increase usability, one of the devices generates both QR codes, and the second device scans both. This allows the displaying device to change to the second QR code as soon as the data exchange is complete. The scanning device can scan both QR codes successively without any user interaction.

From the viewpoint of the user, the whole pairing process is very simple.

- Alice shows the QR codes on her device to Bob.
- Bob positions his camera so that it can read both Alice’s QR codes and waits for a confirmation message, which appears after both QR are read.
- As soon as the confirmation appears on Bob’s device, Alice presses a confirmation button in her pairing application.

We detailed the pairing process from a user’s points of view in Chapter 4.

Because Alice generates both QR codes, she both provides her network parameters to Bob, and initiates the cryptographic pairing protocol for which she needs Bob’s network parameters. Thus, for realizing this method, Bob needs to send a single message to Alice — providing his network parameters — before Alice can initiate the cryptographic pairing protocol.

\(^{31}\)http://line.me
\(^{32}\)https://whispersystems.org
**Interface Constrained Devices.** For interface constrained devices we provide an automatic pairing method described in Subsection 5.4.7, and a cable-based method. The cable-based method demands for the two devices engaged in pairing to be connected by a cable. Since the cable provides authenticity, a plain Diffie-Hellman key exchange suffices to exchange a pairing secret. Using the Diffie-Hellman key exchange comes with the advantage of both parties contributing to the secret.

We do not consider SAS based manually authenticated pairing methods for constrained devices, because for most applications automatic pairing (or cable) is sufficient in combination with pairing data synchronization. But since our pairing daemon proposed in Section 5.5 is highly modular, it is possible to integrate further pairing methods if desired.

Further, we focus on multi-purpose devices that might be paired with multiple other devices for various applications. We do not consider devices that have a single designated pairing partner device.

### 5.4.4 Summarizing Overview

Figure 5.13 summarizes and illustrates the whole pairing process comprising discovery, secure pairing data exchange, and manual authentication from an abstract point of view.

![Figure 5.13](image-url)
5.4.5 Single Interaction Pairing

It is tempting to think about a single interaction pairing solution. However, such a solution is either insecure or requires both a secure OOB channel and long values to be sent over this OOB channel.

In a single-interaction approach, Alice needs to send the authentication string via the OOB channel alongside the discovery data, which means, the authentication string is transmitted before any message from Bob can be taken into consideration. This in turn means, an SAS is not feasible because for protecting an SAS against offline attacks, Bob needs to confirm that he received Alice’s commitment. If this confirmation is not provided, offline attacks against the SAS are possible. Given a secure OOB channel that allows transmitting a sufficient amount of data, two possible approaches are:

1. Transmitting a long secret nonce $r_a$ over a secure OOB channel. Using this method, the authentication of both sides relies on the security of the OOB channel.
   - Alice first sends her network parameters to Bob (either via the unsecured network or an OOB channel).
   - Alice sends a long nonce $r_a$ over a secret OOB channel.
   - Bob sends $g^b, \text{hash}(r_a, g^b)$. Because $r_a$ is secret, this hash can only be calculated by Alice and Bob. Alice can verify Bob’s authenticity.
   - Alice sends $g^a, \text{hash}(r_a, g^a)$. The fact that $r_a$ is secret also allows Bob to verify Alice’s authenticity.

2. Transmitting a long secret nonce $r_a$ and a cryptographic hash over a secure OOB channel. Using this method, only authenticating Bob relies on the security of the OOB channel.
   - Alice first sends her network parameters and $g^a$ (either via the unsecured channel or an OOB channel).
   - Alice sends $\text{hash}(g^a), r_a$ via a secure OOB channel. While $\text{hash}(g^a)$ could be sent over a standard OOB channel, because for $\text{hash}(g^a)$ guaranteeing that Alice is the sender suffices, $r_a$ has to be sent over a secure OOB channel. The long cryptographic hash allows Bob authenticating Alice even if the channel was not secure. But, this method requires sending two long values over an OOB channel.
   - Bob sends $g^b, \text{hash}(r_a, g^b)$. Because $r_a$ is secret, this hash can only be calculated by Alice and Bob. Alice can verify Bob’s authenticity if $r_a$ has been transmitted over a secure OOB channel.

The only OOB channel realization that can feasibly transmit the required amount of data is QR codes. However, QR codes do not provide a secure OOB channel. If the attacker Eve photographs the QR code containing the secret nonce $r_a$, she might be able to make Alice accept $g^e$ instead of $g^b$.

5.4.6 Pairing Data Synchronization

To further enhance convenience, we propose pairing data synchronization among devices of the same user. This allows reducing the number of necessary manually authenticated pairings for both intra user pairing and inter user pairing.
5.4 A User-friendly Pairing Solution

Figure 5.14: If users’ devices (in circles) are paired to each other, it is sufficient to pair one of their devices to another user’s device to be able to establish a secure connection to all devices of this user in a privacy-preserving way. This is possible because pairing data can be synchronized among all devices of a user if pairing data synchronization is supported. User A paired his smartphone with B’s notebook which makes A and B friends; the pairing data synchronization allows B to connect to A’s devices with his smartphone, even if A does not know this device.

Users can specify one (or a few) of their devices as pairing master and pair all of their devices to a master device. Pairing data synchronization can then be used to get a fully connected pairing graph among the user’s devices. This reduces the number of necessary intra user pairings for \( n \) devices from \( n(n - 1)/2 \) to \( n \).

Further, an inter user pairing performed with a master device can be synchronized to all other devices of a user. This makes a single inter user pairing per pair of users sufficient, making the scenario shown in Figure 5.14 possible. Pairing data synchronization allows establishing secure connections to all devices of a friend, even if just one of them has been paired. Intra user pairings are always established with a master device. If two non-master devices engage in intra user pairing, the initiating device automatically becomes a master device.

Choosing master devices does not cause a loss of generality, as a user is free to make all devices a master device. We introduce the role of a master device because we want devices that are able to control the synchronization process to have the appropriate interface, e.g. a display, and to be chooseable by users. A user might also want to choose certain devices to be excluded from inter user pairing. To protect privacy, devices paired via intra user pairing are per default excluded from establishing inter user connections. Users may change this pressing a button after the pairing process or at any later point in time in the pairing manager user interface available on master devices. We discuss realizing a user interface in Chapter 4 and Section 5.5. The software user interface allows managing groups of paired devices, as users might not want to establish a fully connected pairing graph.

We realize pairing data synchronization via DNS-SD introducing the new service type \_psync\_tcp. To guarantee secure and privacy-preserving synchronization, we leverage our DNS-SD privacy extension detailed in Chapter 6. This service can also be used to create a pairing data backup.

Using pairing data synchronization, it is not necessary to pair interface constrained device among each other. They can each be paired with e.g. a smartphone and then be synchronized.
5.4.7 Automatic Pairing

To further increase user-friendliness, we propose a configurationless pairing method, which can be utilized in secure networks. In this context, secure networks are broadcast networks that restrict access to trusted users, e.g. a home WiFi protected by WPA2. We designed this pairing method as a comfortable means for 
\textit{intra user} pairing, but it is also applicable for \textit{inter user} pairing, e.g. in a conference WiFi network.

This pairing method is also based on DNS-SD/mDNS; we introduce a new service type \texttt{_autopairing._tcp}. It also utilizes our TLS based protocol — without the authentication part — which makes realization easier. A user has to activate automatic pairing in the user interface. Further, the user should set a (device) name. This device name is used as instance name for the \texttt{_autopairing} service instance. A device with activated automatic pairing will both publish its own automatic pairing instance and discover instances offered by other devices. For each discovered device, it performs our TLS based pairing during which the initiating device provides its name. The names (one extracted from the instance name, the other from the protocol) are then used to store the respective pairing information. A device will only initiate the pairing protocol if there is no pairing with a colliding name. Optionally, all pairings established during a specific automatic pairing session can be assigned to a group. This also allows for per-group name spaces and avoids aborting pairings that have colliding names but belong to different users. Collisions occurring in a single session, e.g. at a conference, are handled by DNS-SD/mDNS as the name is used as instance name, which in turn is guaranteed to be unique by DNS-SD/mDNS. For \textit{intra user} pairings, such collisions are easy to avoid. A further possibility to avoid collisions in \textit{intra user} pairing sessions is using names that are unique per se; e.g. email addresses.

5.5 Pairing Daemon

In this section we propose our pairing component which can either be realized as a standalone daemon handling pairing for all applications on a system, or be integrated with an existing daemon (e.g. a DNS-SD/mDNS daemon). Using the standalone daemon, pairing becomes even more user-friendly as a single \textit{inter user} pairing suffices to allow various applications to establish a secure connection. Since it seamlessly integrates all necessary components for establishing a device pairing, it is convenient for both users and application programmers. While discussing the pairing component we refer to it as daemon and assume it serves several applications. The pairing daemon does not tamper with other (proprietary) pairing mechanisms. A smartphone utilizing the pairing daemon can e.g. still engage in standard Bluetooth pairing; existing applications utilizing such pairing methods are also not disturbed.

The pairing daemon maintains a mapping from \textit{friend IDs} to the corresponding friends’ \textit{pairing data}, which is stored in a database (e.g SQLite). The user interface — our \textit{privacy enhanced phone book} we illustrated and discussed in Chapter 4 — can access this database. It looks like a phone book application and can be used to edit contacts, initiate a pairing with a stored contact, and add contacts to groups. Further supporting usability, friends can also be imported from existing phone book applications.

For users, the pairing workflow is convenient. It suffices to open the pairing application, choose the contact they want to pair with, and then initiate and verify the pairing as described in Section 5.4. The corresponding authenticated pairing data is then associated
with the contact and applications can now establish secure connections without any further user interaction.

As of yet, we consider a simple direct interface for applications providing either read access to the database or offering a single function `get_pairing_data(friend)` returning the pairing data corresponding to the desired friend. The pairing data can either contain a shared secret, which we consider as default case, or a certificate. If the friend ID does not map to pairing data a null value is returned; this can be used as an event starting the phone book application. We also plan to provide an indirect interface acting as a barrier against malicious applications.

The following subsections deal with the details about (1) the database maintained by the daemon, (2) how the daemon establishes a pairing, and (3) how an application can use the pairing daemon. We detailed the phone book application and the steps users have to perform for initiating a pairing in Chapter 4.

### 5.5.1 Database

The pairing daemon maintains pairing data as well as group related information for both inter and intra user pairings in a database (our implementation described in Section 5.6 uses an SQlite\(^{33}\) database). An example database — maintained by the pairing daemon on Alice’s smartphone — is shown in Table 5.1.

There are two tables containing the pairing information for intra and inter user pairing, respectively; and further two tables which contain information about groups for the two kinds of pairings, respectively. In this subsection, we discuss the main design aspects of the database. Implementations are free to add further columns, e.g. a group description or identifiers representing group hierarchies; new tables which are linked via foreign keys could also be incorporated. Implementations can further allow friends to be members of several groups.

We refer to the table containing inter user pairing information as **friend table** and the table containing intra user pairings as **device table**. These tables are indexed by a primary key referred to as `friendID` and `deviceID`, respectively; this primary key uniquely identifies a pairing. The corresponding groups are referenced using the `groupID` as foreign key.

For the inter user pairing, we do not distinguish the various devices a friend might have, because we assume friends can be trusted, and the only device we could certainly assign to a group would be the one that established the corresponding inter user pairing.

The pairing daemon manages groups, because these groups can be beneficial to several applications using the daemon. But the actual rights management that determines which groups or friends are allowed to get information via a secured connection depends on the application, and thus has to be managed by the application and the application’s user interface, respectively.

In the case of our privacy-preserving service discovery framework (Chapter 4), the pairing daemon manages groups and the service discovery daemon manages access rights to certain service types and instances. An exception to this is the pairing data synchronization service, for which the pairing daemon running on master devices decides which devices will receive which pairing data.

\(^{33}\)https://www.sqlite.org

\(^{34}\)The smartphone is the current device.
### Table 5.1: Example database maintained by the pairing daemon running on Alice’s smartphone.

From top to bottom there are the friend table, the friend group table, the device table, and the device group table. friend ID is the primary key of friend table; friend name can be a foreign key imported from an existing contact application. The pairing data contains the shared secret exchanged during pairing; it could also contain a certificate exchanged and authenticated during pairing. For this example database, Alice has paired all of her devices with her smartphone, and synchronized the pairings among the others. The first letter of a name determines the variable name of the Diffie-Hellman private key; e.g. b for Bob. The variable names for the secrets of Alice’s various devices contain the deviceID; e.g. a2 is the Diffie-Hellman private key chosen by Alice’s notebook. We use the same Diffie-Hellman keys a and a1 several times for ease of notation, because they refer to the same person and device, respectively. If the pairing protocol does not use separate nonces to guarantee randomness, these Diffie-Hellman private keys have to be ephemeral.
5.5 Pairing Daemon

5.5.2 Establishing the Pairing

The pairing daemon allows for a seamless experience when establishing a pairing as it integrates means for all three stages of device pairing, namely discovery, secure pairing data exchange, and manual authentication. For each of these stages, we consider the realizations discussed in Section 5.4. For the discovery stage we support DNS-SD with random instance names to protect the privacy during discovery. Once the devices know each other’s network parameters they establish a connection and either directly use the Hoepman protocol or our TLS handshake based protocol for the secure pairing data exchange. To realize authentication, the corresponding SAS is either validated via QR code scanning or via compare & confirm using an alphanumeric or a sentence representation for the SAS.

Since we designed these stages to be independent, a new realization could be substituted for each stage individually. Besides substituting other means we discussed in this chapter, we could even leverage a web of trust, e.g. PGP\(^{35}\), for authentication.

It would also be possible to substitute an integrated solution for all stages, e.g. Bluetooth. Pairing daemons could establish a Bluetooth pairing among devices, extract the Bluetooth link keys\(^{36}\), and use these link keys to derive a shared pairing secret. Thus the Bluetooth pairing would not only allow for subsequent Bluetooth connections, but for all types of connections applications using the pairing daemon want to establish. These connections would inherit the security level from the Bluetooth pairing, which would be insufficient in the case of just works pairing and might give users a wrong sense of security. Using e.g. Bluetooth as the only pairing method would destroy the flexibility of our pairing daemon. Further, Bluetooth does not protect the privacy during the discovery phase, broadcasting the devices’ IDs. Nevertheless, we propose the described integration of Bluetooth pairing as a further possibility to realize an intra user pairing, which should only be used in a secure network, e.g. at home.

The pairing daemon further allows automatically establishing pairings in trusted networks as described in Subsection 5.4.7. Independent of the chosen pairing mechanism, devices transmit a name managed in the pairing daemon’s user interface. These names are transmitted as first encrypted payload and can be used by either party to automatically associate a name with the pairing without the need of an additional per-pairing user interaction. Especially, this is necessary to allow automatic pairing.

5.5.3 Utilizing the Pairing Daemon

The pairing daemon benefits both users and developers; the former by making a single inter user pairing sufficient for various applications, the latter by an API providing shared secrets that can be leveraged to establish a connection.

Private Mutual Authentication. Let’s look at a general scenario to demonstrate utilizing the pairing daemon. We assume Alice wants to use an application that utilizes the pairing daemon to establish an authenticated encrypted communication channel to Bob.

First, Alice’s application asks the pairing daemon for Bobs pairing data including the pairing secret \(p_{ab}\). Her application will then use an arbitrary discovery mechanism — e.g.

\(^{35}\)http://www.pgpi.org
\(^{36}\)On many Linux distributions, the link key is extractable by accessing /var/lib/Bluetooth/[BD_ADDR]/linkkeys. On Windows a user with administrator rights can access a registry key for extracting the link key.
our privacy-preserving DNS-SD extension presented in Chapter 6 — for retrieving the network parameters of Bobs device.

Alice can then initiate a protocol that provides private mutual authentication. The authentication at this point is easy, because both parties share a common previously authenticated secret. An ad-hoc protocol serving as example is shown in Figure 5.15.

![Figure 5.15: Ad-hoc protocol for establishing a secure, privately mutually authenticated session using symmetric cryptography. The last message from Bob to Alice can already contain payload encrypted with a key derived from the ephemeral $g^{ab}$.](image)

Alice sends a connection initiation request to Bob, which must contain a private identifier. Only Bob should be able to relate this identifier to Alice. We refer to this identifier as temporary private ID (tpID). A given temporary private ID is only valid in a certain time interval. Alice constructs the tpID as $tpID_a = HMAC_{ps_{ab}}(timestamp), g^a, HMAC_{ps_{ab}}(g^a)$. The initiation request further contains an ephemeral Diffie-Hellman public key $g^a$, and a proof showing that Alice owns $ps_{ab}$. This proof can e.g. be realized by using an HMAC over $g^a$ using $ps_{ab}$ as key, $HMAC_{ps_{ab}}(g^a)$. This HMAC both protects the integrity of $g^a$ and proves the ownership of $ps_{ab}$.

Bob now has to relate this message to Alice. To this end, he has to maintain a hash table that maps temporary private IDs valid in two consecutive time intervals to friend IDs. After this, Bob validates Alice’s identity also calculating the HMAC $HMAC_{ps_{ab}}(g^a)$ and checking whether the result matches. The results only match if both parties used the same $ps_{ab}$. After this validation, Bob sends Alice $g^b$ and the corresponding HMAC proving to Alice that he is in possession of $ps_{ab}$. Bob’s message might also contain the first payload, which is encrypted by an ephemeral session key derived from the just exchanged Diffie-Hellman secret $es_{ab} = HKDF(g^{ab})$, where HKDF is the afore discussed HMAC-based key derivation function defined in [KE10].

Upon receiving this message, Alice checks Bob’s identity, and sends a confirmation along with her first payload encrypted with $es_{ab}$.

Alice’s tpID does not suffice for authentication, despite the fact that it is an HMAC using $ps_{ab}$ as key. A message just containing $tpID_a$ and $g^a$ could be replayed substituting $g^o$ for $g^a$.

For this authentication protocol, two messages suffice because an offline attack on $ps_{ab}$ does not have to be prevented as it is infeasible. The pairing secret $ps_{ab}$ has been derived using HKDF [KE10] from the Diffie-Hellman secret exchanged during pairing and is thus

---

37 Maintaining only a single temporary private ID would cause serious problems if the clocks of Alice or Bob were skewed.

38 Theoretically, there is a minuscule probability for a collision. But using the HMAC construction defined in [KBC97], an attacker is confined to an exhaustive search that is infeasible on today’s hardware.
5.6 Implementation

a secure key providing enough randomness to render an exhaustive search for a $g^{a'}$ that fulfills $\text{HMAC}_{ps_{ab}}(g^{a'}) = \text{HMAC}_{ps_{ab}}(g^a)$ infeasible. We would have to prevent offline attacks if $ps_{ab}$ was an insecure password. In this case we would need a password authenticated key exchange mechanism (e.g. [BM92]) to prevent the attacker from performing offline attacks on this password.

Using our TLS Based Protocol. For a practical implementation, we suggest to realize the afore mentioned authentication protocol by establishing a TLS connection using a PSK (Pre-Shared Key) cipher suite. In doing so, the application programmer just has to hand the shared secret provided by the pairing daemon to the TLS library. The application programmer neither has to worry about realizing a secure pairing method, nor about establishing a secure connection using the pairing secret. We detail the utilization of the pairing secret for establishing a privacy-preserving TLS PSK connection in Subsection 6.5.3.

UI considerations. An application utilizing the pairing daemon should provide the user with a view on the daemon’s contact management (phone book) application. This can e.g. be a user interface which presents available contacts and groups. Further, it should either provide means to manage access rights of these groups, or inherit these rights from defaults defined in the daemon’s database. Users should be able to add contacts and initiate pairings either by API calls to the daemon or by automatically opening the daemon’s user interface if a new pairing is desired.

5.6 Implementation

The implementation of our pairing solution — which runs on both Linux distributions and Android systems — comprises our pairing daemon and our privacy enhanced phone book. In this implementation, pairing daemon and the privacy enhanced phone book are realized as components running as separate processes communicating via Unix domain sockets. A generic pairing interface allows other applications to directly communicate with the pairing daemon; it further allows using a substitute user interface application.

5.7 Conclusion and Future Work

In this chapter we proposed efficient, secure, and user-friendly mechanisms for all stages of pairing, namely discovery, secure pairing data exchange, and manual authentication. The mechanisms can be seamlessly integrated and provide application programmers with means for an easy establishment of secure connections without being dependent on central services. We especially cared about making these mechanisms practical leveraging TLS as much as possible. While TLS is the de facto standard for establishing secure connections, as of yet it only provides means for authentication that rely on central services, e.g. certificate authorities, or pre-shared keys. Our pairing daemon integrates our pairing mechanisms seamlessly, and provides arbitrary applications with means for establishing secure connections. Except for a single user-friendly pairing process, the pairing daemon makes establishing secure connections as easy as establishing unsecured connections. This

39https://gitlab.com/kaiserd/pairing_daemon.git
aligns well with our philosophy of providing security at (almost) no cost for users, and — as a further benefit — for application developers as well.

In the future we will work on making manually authenticated pairing available for a large number of applications and users. Therefore we will continue working on our Internet draft on DNS-SD pairing, as well as working on a draft that integrates manually authenticated pairing in TLS. We are currently working on transitive pairing methods that allow two parties that share a common paired peer to engage in a simplified pairing process (from a user’s point of view). We plan to use transitive pairing to allow pairing when the involved parties do not meet in person, but also to save the user interaction when users are in close proximity. In addition, we plan to conduct a user study to further improve the usability of the pairing daemon and to evaluate whether compare & confirm or QR code scanning is the most user-friendly SAS verification method. We also plan to provide an indirect interface to the pairing daemon, providing a barrier against malicious applications.
Bene vixit, bene qui latuit.
To live well is to live concealed.

Ovid

Efficient Privacy-Preserving DNS-SD

Contents

6.1 Introduction ............................................. 145
6.2 Related Work ............................................. 146
6.3 Problem Analysis and Requirements ......................... 147
6.4 Towards Privacy-Preserving DNS-SD/mDNS .................. 149
6.5 Efficient Two-Stage Privacy Extension ....................... 156
6.6 Implementation ............................................. 166
6.7 Privacy and Security Considerations ......................... 167
6.8 Network Performance Analysis .............................. 168
6.9 Privacy-Preserving Hostname Resolution .................... 178
6.10 Privacy-Preserving DNS-SD/DNS .......................... 179
6.11 Conclusion and Future Work ............................. 179

While DNS-SD/mDNS is very convenient, it causes the public exposure of identities along with information about the offered and requested services. Some of the information published by the announcements can be very revealing, including complete lists of family members. Another problem is the huge amount of multicast traffic caused, which is especially relevant for large WiFi networks.

This chapter presents a privacy extension that does not publish private information and reduces the number of packets sent while still not requiring any configuration except for an initial pairing per pair of users. A key feature of our solution is the ease of upgrading existing systems, a must for widespread deployment and acceptance. Our solution grants tunable privacy and reduces multicast traffic without affecting user experience.

Our solution presented in this chapter has become the basis for the IETF Internet Draft on DNS-SD privacy extensions, which has been adopted by the dnsssd working group, rendering our solution likely to become a standard for privacy-preserving service discovery.
I would like to thank Christian Huitema (Microsoft) for giving me the opportunity of coauthoring the IETF Internet Draft on DNS-SD privacy problems and mitigation techniques [HK16b], and for all the helpful discussions while working on the draft.
6.1 Introduction

DNS Service Discovery over Multicast DNS (DNS-SD/mDNS) is very convenient for users, but suffers from serious privacy problems. We discussed both the various advantages and problems of DNS-SD/mDNS in Chapter 2 and Chapter 3 along with an explanation of the protocol itself. The privacy problem, which DNS-SD/mDNS shares with other multicast-based service discovery protocols, arises because all information necessary for service discovery is sent in cleartext form to everyone in the same multicast domain.

Since DNS-SD/mDNS is the dominant service discovery solution in today’s local networks and supported by almost every device, we consider it especially important to provide means for mitigating the uncontrolled publishing of sensible data, which most users don’t even know about [KBSW13].

In this chapter, we present a privacy extension for DNS-SD, which

- guarantees privacy by sending private discovery data only to trusted hosts instead of multicasting it to everyone in the same network,
- significantly reduces multicast traffic, moving further towards the Zeroconf group’s goal of reducing multicast traffic,
- is transparent to client software using DNS-SD/mDNS,
- is transparent to the existing network infrastructure,
- allows automatic service discovery like standard DNS-SD/mDNS,
- allows presenting only services a user can use, reducing visual clutter,
- is fully backward compatible,\(^1\)
- is very efficient, with respect to all of network traffic, memory consumption, CPU time, and wall clock time.

Our privacy extension needs users to be able to authenticate each other, which in turn demands a pairing. To this end, our privacy extension leverages the pairing solution we proposed in Chapter 5. After this initial pairing — which can also be used by other applications — service discovery requires no further configuration allowing hosts to offer

\(^1\)Public service operation remains entirely unchanged; private services seem entirely unchanged within the trusted group.
and request arbitrary service instances, which do not have to be known at the time of pairing. The IETF Zeroconf charter\(^2\) has stated that minimal configuration is tolerated for security’s sake. We believe this includes privacy as well, which was confirmed by the IETF as the IETF dnssd working group has adopted our Internet draft on DNS-SD privacy extensions [HK16b].

To realize our goals, our solution extends DNS-SD by a meta service provided by a private service directory server (PSDS) running on each host (see Figure 6.2). It integrates seamlessly into DNS-SD because it merely acts as alternative way of propagating resource records, and is transparent to both applications and the network.

Figure 6.1 shows the bigger picture integrating the privacy extension with our service discovery framework; we discussed this integration in Chapter 4.

### 6.2 Related Work

We are not aware of any other publication regarding privacy extensions for DNS-SD. Nevertheless, the problem of DNS-SD/mDNS publishing device names has been addressed [ALRM08, KBSW13].

Aura et al. [ALRM08] investigate private information published on different network layers when connecting devices to a network. While they mainly look at other protocols, they also mention the privacy problem of device names published by DNS-SD/mDNS. As a solution, they propose network location awareness, that is allowing service discovery only in trusted networks. We consider that too restrictive and want to give the user the possibility to request and offer services in a privacy-preserving way even if the network cannot be trusted. Further we want to mitigate the configuration overhead of making assumptions about the trustworthiness of the network in use.

\(^2\)[http://datatracker.ietf.org/wg/Zeroconf/charter]
6.3 Problem Analysis and Requirements

The public announcement of device names by DNS-SD/mDNS is discussed in more detail in [KBSW13]. The authors conducted studies showing that almost 60% of the published device names contain real user names and that 90% of the users consider this as a privacy problem. They propose making users aware of the problem and changing hostnames as a solution and also refer to network location awareness and identifier free networks [GMP+08]. The privacy problem of SRV and TXT records being published is not mentioned. We consider it important to hide all possibly private information, because it can be seen by anyone as easily as the hostname and might deserve more privacy protection. Furthermore, users should not have to change hostnames manually.

Much research has been done in the area of privacy in wireless networks. Especially finding access points in WiFi-networks is related to our privacy extension, because it also has to solve the task of finding entities in a network using broadcast, where everybody in the vicinity can listen to the messages. [LAD+09] presents a privacy-preserving protocol for discovering WiFi access points. Since it is an extension to the 802.11 MAC protocol, it does not meet our requirement of not changing any deeper protocol layers.

Greenstein et al. [GMP+08] present an identifier free wireless link-layer protocol, which allows finding access points and communicate in the wireless network in privacy. Since it provides privacy on the link-layer, it also solves the private service discovery problem. But to allow private service discovery, the whole network infrastructure has to be updated, while our solution allows private service discovery by just updating the local Zeroconf daemon.

Generic protocols for privacy-preserving service discovery [CZH+99] and presence sharing [CDM07] have been proposed. Since they use a central entity [CZH+99] and depend on a trusted broker [CDM07], respectively, we did not use them because we want to keep DNS-SD/mDNS decentralized and do not want to rely on trusted third parties.

There is much research in the area of service discovery in pervasive computing environments [ZMN05] and mobile ad hoc networks [OVB07]. Zhu et al. present a model for privacy-preserving service discovery [ZMN06, ZZMN07]. Since it is a generic model, we consider it unnecessarily complex for our application area. Nevertheless it is a sophisticated model and we want to incorporate ideas like the usage of Bloom filters in our privacy extension in future work.

Mechanisms such as IPv6 privacy extensions [ND01] and locally administered MAC addresses also aim to reduce the ability of others to trace the whereabouts of end systems. These methods work independently of our privacy extension, the network and link layers, respectively, and thus can be used at the same time.

Device fingerprinting [KBc05, DYPL08] can be used to track the devices, even if the protocol in use has no explicit identifiers. It is outside the scope of this chapter to address tracing using information leaked by the physical layer [BMG+09] and tracing of the users by other metadata, such as the number or rate of connections devices open, or where they connect to [GKB12].

6.3 Problem Analysis and Requirements

Analyzing DNS-SD/mDNS and in particular its privacy problems shown in Chapter 3, we build a threat model by defining attackers that might abuse standard DNS-SD/mDNS, and define requirements that should be met by a DNS-SD privacy extension in order to prevent the corresponding attacks.
Chapter 6. Efficient Privacy-Preserving DNS-SD

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Enables attack type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Passive</td>
</tr>
<tr>
<td>Runtime Efficiency</td>
<td>—</td>
</tr>
<tr>
<td>Network Efficiency</td>
<td>—</td>
</tr>
<tr>
<td>Information Hiding</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 6.1: Unmet requirements open pathways to attacks.

6.3.1 Threat Model

The following attackers can be successful, because all data needed for service discovery is sent as cleartext; each responder potentially answers queries from each querier and each querier accepts answers from each responder.

*Passive:* The passive attacker wants to get as much information as possible by just listening to the multicast traffic. All plaintext information will be gained by this attacker type.

*Active:* An active attacker wants to get information by sending queries for services he is interested in. He might ask for all _presence_ service instances, extract the version numbers from the TXT records, identify the vulnerable versions and attack the corresponding hosts. He can also offer (fake) services or impersonate someone.

When designing a privacy extension, it is important not to open new attack vectors. One of those can easily be enabled if the extension causes either significant computational overhead or significant network overhead.

*DoS:* This attacker wants to bring hosts running mDNS to a halt by flooding the network with multicast messages in the mDNS multicast group. A motivation for a DoS attack on DNS-SD/mDNS could be just considering it fun to drain the battery of many devices.

6.3.2 Requirements

Considering these attackers, a DNS-SD privacy extension should meet the following requirements. Table 6.1 shows which attack types become performable when neglecting one of them.

*Information hiding:* All information published using DNS-SD should only be accessible by paired devices. This includes the hostname, the service instance name and service type, the port, and TXT records.

*Network efficiency:* The addition of privacy to DNS-SD should not cause a heavier network load than standard DNS-SD.

*Runtime efficiency:* Hosts must be able to process incoming multicast queries and answers in $O(1)$ and the constant calculations have to be efficient.

The runtime efficiency is also important for batterylife’s sake and to be able to process incoming queries at wire speed; otherwise, packets have to be dropped or cached in burst times, where many multicast packets arrive.
6.3.3 Consequences

Solutions based on encrypting whole packets and sending them without an application layer identifier are infeasible as they allow the DoS attacker to succeed. This is due to the fact that these solutions demand every incoming packet on the multicast DNS socket to be decrypted. Using symmetric encryption, each incoming packet has to be tested with keys corresponding to each paired service instance, requiring $O(\# \text{pairings})$ time. Using asymmetric encryption, including some broadcast encryption mechanisms [Del07, WQZDF10], the asymptotic run time goes down to $O(1)$, yet the calculations needed for asymmetric encryption are very expensive. Assuming the packets sent contain 2048 bits of data encrypted using RSA, a DoS attacker can send roughly half a million such packets per second on a gigabit Ethernet link, while a modern notebook or desktop processor can only decrypt about 600 packets in the same time. A further drawback of these methods is the battery power wasted.

6.4 Towards Privacy-Preserving DNS-SD/mDNS

In this section we will present a proof-of-concept solution showing that privacy in DNS-SD/mDNS can be granted with little effort. While this first solution provides privacy with imperceptible additional CPU time, the two-stage privacy extension presented in Section 6.5 outperforms this simple solution with respect to privacy, efficiency and user-friendliness.

6.4.1 Pairing

As pointed out in Chapter 4 and Chapter 5, pairing is essential for privacy-preserving communication when not relying on a central service. For this first solution, we do not use the pairing daemon proposed in Chapter 5.

The first solution’s naive pairing method establishes a relationship between a service instance and a querier (Figure 6.3) by transmitting a static shared data structure called privacy service data that is created by the service offerer and contains

- the service instance name,
- the service type,
• a service instance key used as substitution for the service instance name, and
• cryptographic key material.

The privacy service data is stored in two hash tables; one indexed by the service instance name and one indexed by the service instance key. If this service has already been paired with another service querier, the privacy service data creation and storing is omitted. The service data has to be transmitted to the service querier via a secure in-band [GAZK11] or an out-of-band channel, such as Bluetooth, NFC, photographing a QR code [MPR05, SEKA06], encrypted Email, or SMS. The service querier then saves the service data in his hash tables, which completes the pairing process. To meet the efficiency requirement, it is important for the service data to be retrievable in $O(1)$ time.

At first glance it might seem that this naive pairing method destroys the automatic detection of services in a new network. But information like the IP address of devices, ports that services use and other information about services like the version number or status fields can still be detected automatically, independent of the network parameters.

### 6.4.2 Hiding the Instance Name

When using DNS-SD/mDNS, the service instance name and service type allow the receiver of a packet to decide in constant time whether she is interested in the corresponding service instance. Since this service identifying data should be hidden from the public, we substitute the service instance key, which was agreed upon during pairing, for the service instance name. This is a simple solution for hiding the information while still guaranteeing $O(1)$ lookup time, because when receiving a query or answer the receiver can look up this key in her local hash table.

### 6.4.3 Hiding the Service Type

To hide the service type, some sort of type keys which only partners should be able to map to the corresponding service type are necessary. In order to meet the network efficiency requirement (see Section 6.3), we want a single service instance to be represented by exactly one pair of service instance key and type key. The advantage of one single obfuscated representation corresponding to a single service instance is that important DNS-SD/mDNS features such as known answer suppression, duplicate query suppression and duplicate answer suppression (see Chapter 3) work like they would without the privacy extension, because the DNS-SD/mDNS subsystem recognizes it as a single service instance. Establishing this mapping for the service instance name is easy, because a service instance name is mapped to a single service instance offered by a single service offerer. This is different for the type. Each group of paired users$^4$ has to use the same type key, to allow for a single pair of service instance key and type key. This is unpractical for two reasons. On the one hand all users of such a relation would have to agree on one type key, which is not possible using a simple, convenient pairing strategy; on the other hand it would allow all users in that potentially large group to know the service type in a query or answer of any of this group’s members.

To solve this problem, we use a special service type \_privacy\_tcp for all privacy-enabled service instances. Thus the published service type has no relation to the real service

$^4$all users A and B who are in the transitive closure of the relation A is paired to B regarding a certain service instance S
Figure 6.4: Resolving a service using the naive simple solution. Instead of browsing for service instance names corresponding to a certain type, they are retrieved from a local hash table built during pairing. Private data is either substituted by random strings exchanged during pairing, or it is encrypted.

This renders browsing for services in the usual way described in Chapter 3 infeasible, because browsing for _privacy._tcp would return all private services; e.g. when using a chat client with a privacy aware presence protocol not only the chat buddies would pop up as friends, but all other private services as well. To solve this, we have to change the way private services are browsed. Browsing for a service type returns PTR resource records showing service instances of this service type (including their name), which can then be resolved. Since we already obtained this information during pairing and stored it in local hash tables, we do not have to browse for a service type using multicast at all. Our privacy-preserving browsing happens locally and the matching service instances are resolved directly. We call this direct resolving because it directly resolves a service instance without browsing for the corresponding type.

An advantage of direct resolving is the reduction of multicast packets, because no browsing queries and answers have to be sent. Additionally, the number of resolving queries is also reduced, because queries and answers are only sent for service instances a user can really use. Normally, a resolve query is sent for chosen PTR records received during browsing. Depending on the number of service instances offered, a significant number of resolving queries or answers can be avoided using direct resolving.

The _privacy._tcp type has another advantage. All services that are found using a service browser but are not meant for the browsing user are listed in a single section _privacy_, and thus the visual clutter caused by those inaccessible services can be easily hidden. A privacy extension aware service browser is also able to not show private service instances the user is not paired to, without any user interaction.
6.4.4 Querying and Responding

This first solution can be easily integrated in existing DNS-SD/mDNS daemons, e.g., Avahi. The existing query and response sequence has to be altered according to Figure 6.4. When a querier asks for a resource record, the query is altered before sending, substituting the service instance key for the service instance name and _privacy._tcp for the service type. Receivers that paired one of their service instances with the querier are able to interpret the request. Before sending the answer, possibly private data in the resource record is encrypted, which is especially important for the key-value pairs in TXT records. When receiving a packet, the querier again recognizes it by looking up the service instance key in the hash table.

Packets that are not meant for a certain receiver are dropped; these packets could have been stored in the cache if they were transmitted in cleartext. This might seem like a disadvantage as cached entries could increase availability and reduce network load. But availability is not increased by caching, because the service offerer has to be available anyway and because DNS-SD/mDNS does not allow hosts to answer queries using their cache entries; hosts must be authoritative for a resource record to be allowed to answer [CK13b]. Further, resource records ignored by our solution mostly belong to services a user is not allowed to connect to; thus ignoring those packets can be seen as advantage, because the cache is not polluted with irrelevant entries.

Using this solution services can still be discovered in the traditional public way, granting backwards compatibility.

6.4.5 Implementation

We implemented our naive proof-of-concept solution as a privacy extension for the open source Zeroconf daemon Avahi6. We changed very little code in the avahi-daemon source files, just adding hooks to our loosely coupled privacy module before queries and answers are sent to and received from the client, respectively. When a client asks the daemon to discover a service, we update the packet by applying our privacy operations as described above, and let the daemon handle the updated packet. We alter the packet in a way that it remains a valid DNS-SD/mDNS packet, making the daemon believe the client asked for a _privacy service instance. When getting the answer, the daemon will recognize it as having been queried before, processes the answer and eventually wants to give this answer to the client; before this happens, we undo all privacy operations and present the plaintext service to the client. The client software does not need to know about the privacy subsystem at all, and the daemon thinks the client wants to query and offer service instances of type _privacy, allowing us to get all benefits of the daemon. Because of the very limited code changes and the loose coupling, it is easy to merge updates from Avahi to our modified daemon.

6.4.6 Performance Analysis

With respect to memory consumption and computational overhead, the simple solution scales in huge networks. Pairing data needs less than 1 Kb of memory per service, so even when paired with 10000 services, the resulting data structure of 10 Mb can be easily held in memory.

5https://gitlab.com/kaiserd/avahi_pnaive
6http://avahi.org
6.4 Towards Privacy-Preserving DNS-SD/mDNS

(a) PTR record hiding the service instance name. The _privacy service type is substituted for the _presence service type.

(b) A privacy aware SRV record. The port is XORed with the key exchanged during pairing. A randomly generated string that was also agreed upon during pairing is substituted for the hostname.

(c) TXT record showing encrypted key-value pairs. The key-value pairs were concatenated before encrypting and split afterwards to avoid inferring of the service based on typical key-value pair quantities and lengths.

(d) A record with hidden hostname. A mapping of hostnames containing user’s names to IP addresses is no longer available.

Figure 6.5: Resource records multicast by our privacy extension when using a chat client that is based on the _presence chat service. Critical data is substituted by identifiers randomly generated during pairing, or it is encrypted.

The computational overhead is imperceptible. For each incoming and outgoing packet a hash table lookup, and, depending on the resource record type, a few encryption or decryption operations have to be done, which is estimated\(^7\) to use less than 100 µs on average even on older mobile devices. The overhead of our extension is virtually non-existent compared to the delays, between 20 ms and 120 ms, that DNS-SD/mDNS is using for network efficiency and feasibility reasons [CK13b].

Using multicast as means for transporting service data poses a bottleneck (see Chapter 4). Each packet sent will be received and processed by each member of the same multicast group, meaning that multicast causes a lot of network utilization on the downlink of all members, if the multicast group is large and if a lot of multicast packets are sent. With \(n\) members and one packet sent per time unit, each member receives \(n^2\) packets per time unit. Our solution presented in Section 6.5, which has become the basis for our Internet draft on DNS-SD privacy extensions [HK16b], mitigates this problem.

6.4.7 Results

When using our basic privacy extension, all data that has to be multicast to discover and offer services is transmitted in a privacy-preserving way. Resource records multicast when publishing a _presence service instance on starting pidgin using our DNS-SD privacy extension are shown in Figure 6.5. The service instance key is substituted for the service instance name, a random string is substituted for the hostname, key-value pairs are encrypted, and the port is XORed with cryptographic key material exchanged during pairing. All the key-value pairs in the TXT records are encrypted as one unit and then split to fit the maximum length, which prevents inferring information based on the number and length of TXT record key-value pairs.

\(^7\)based on glib benchmarks and openssl speed aes-128-cbc
Chapter 6. Efficient Privacy-Preserving DNS-SD

(a) `avahi-browse -r _presence._tcp` on a paired peer. The host gets all information about the `_presence` service instance published by a chat application of a friend.

(b) `avahi-browse -r _privacy._tcp` on an unauthorized host showing encrypted or substituted data. If the unauthorized host would browse for `_presence._tcp` it would not discover this service instance at all.

Figure 6.6: Output of the command line service browser `avahi-browse` on a paired and unauthorized host, respectively. `avahi-browse` is used to show data received at the host.

Figure 6.5(c) shows the encrypted TXT record. Figure 6.6(a) shows the output of `avahi-browse` asking our modified daemon on a paired peer for the service we published above. This shows that client software, in this case the command-line service browser `avahi-browse`, works with our privacy extension without being changed. When using `avahi-browse` to ask for the same service type on an unmodified daemon or a modified daemon of an unpaired peer, the published service instance shown above will not be found. Browsing for the `_privacy` service type yields the result shown in Figure 6.6(b).

6.4.8 Discussion of Problems and Improvements

The presented proof-of-concept privacy extension hides all private information contained in published resource records, while being efficient and transparent. It is efficient because incoming packets can be identified as being relevant in $O(1)$, and thus reduce the efforts of even the efficient symmetric cryptographic operations we use. The major advantage of this solution is that none of the existing protocols in deeper layers of the network stack and none of the existing client software has to be altered. Only the Zeroconf daemon running on the users devices has to be modified, while afterwards still being able to exchange service information with unmodified daemons.

While being very easy to integrate in the existing infrastructure, the solution suffers from problems making it infeasible for real-world usage. The first problem is regarding pairing: Pairing needs to be more lightweight and user-friendly while maintaining a high level of security. Pairing has to be limited to once per pair of users who want to share service instances, instead of the first naive mode of once per pair of service instance and user. In addition to that, the naive pairing makes it impossible to discover new service instances without prior pairing; service instances unknown at the time of pairing should also be discoverable.

To solve the pairing problem, we can leverage the pairing method proposed in Chapter 5. This pairing method yields a shared secret per pair of friends. We can directly use this pairing method to enhance the proposed naive solution with respect to usability.

---

8We use `avahi-browse` to demonstrate data received at the host. Users do not have to bother using a service browser; an DNS-SD/mDNS capable application handles the service discovery transparently.
Instead of transmitting instance keys during pairing we can derive them from the shared secret as follows.

\[
\begin{align*}
\text{seed} &= \text{rounded timestamp} \\
\text{long_hash} &= \text{HASH}(\text{seed} \mid \text{secret}) \\
\text{instance_hash} &= \text{first 4 bytes of long_hash} \\
\text{instance_key} &= \text{BASE64}(\text{seed}) \cdot \text{BASE64}(\text{instance_hash})
\end{align*}
\]

As the instance key should neither allow third parties to derive the secret nor be linkable to previously used instance keys, the basis for calculating the instance key is the result of applying a cryptographic hash function to the concatenation of the secret and a seed based on a rounded timestamp.

A paired receiver can derive the identity of the service provider by concatenating the pairing secrets, which it shares with its paired peers, with the seed retrieved from the service instance name, hashing the result, and — after applying BASE64 — checking whether it matches the instance hash portion of the service instance name. For efficiency reasons, hosts can maintain a hash table which maps instance hashes to friend’s identities; since the instance hash involves a timestamp-based hint, it has to be recalculated when the timestamp interval runs out.\(^9\) The naive solution’s instance key, which is transmitted for each service instance during pairing, has the advantage that the receiver knows the corresponding de-obfuscated service instance name, as it was transmitted along with the instance key. But this is only beneficial if a user offered multiple instances of the same type, and even then it might suffice to display these in client software as e.g. alice’s 1st._presence[...], alice’s 2nd._presence[...]. This kind of naming is only feasible when the type name is not hidden.

This adaption mitigates the pairing usability problem. However, there is another problem whose pecularity is determined by a design decision. Using the first option, deriving the instance key as described above, we get a scalability problem. The time for de-obfuscation will scale unsatisfying as for each incoming obfuscated service instance, each shared secret has to be tried for checking whether the corresponding de-obfuscated service instance name, as it was transmitted along with the instance key. But this is only beneficial if a user offered multiple instances of the same type, and even then it might suffice to display these in client software as e.g. alice’s 1st._presence[...], alice’s 2nd._presence[...]. This kind of naming is only feasible when the type name is not hidden.

This adaption mitigates the pairing usability problem. However, there is another problem whose pecularity is determined by a design decision. Using the first option, deriving the instance key as described above, we get a scalability problem. The time for de-obfuscation will scale unsatisfying as for each incoming obfuscated service instance, each shared secret has to be tried for checking whether the corresponding service instance is relevant. Further the number of packets sent to publish a service instance scales linearly with respect to \(m\), where \(m\) is the number of pairings of a host.

The other option is sharing the same secret among all paired peers, yielding the same service instance key for all paired peers. This allows for efficient de-obfuscation and does not need additional messages. However, this solution renders revocation infeasible. When revoking the friend status from one peer, a re-pairing has to be done with all other paired peers before being able to announce services privately.

For both options, using a timestamp as the seed allows for calculating instance keys only once per time interval.

Further, since DNS-SD/mDNS has shown to cause significant load in huge networks [HSS09], the number of necessary multicast packets should be reduced significantly.

In summary it becomes obvious that despite the privacy we gain and various other advantages, obfuscating instance names is not an outstanding solution for private service discovery.

\(^9\)To cope with time skew, two such tables should be maintained for the current and previous timestamp intervals, respectively.
(a) One-stage. Hosts query each peer in the same network.

(b) Two-stage, first stage. Hosts use multicast just to query for the `_psds` meta service. Since the packet content is obfuscated, there are no privacy leaks.

(c) Two-stage, second stage. Hosts directly query PSDSes running on paired peers via a secure, mutually authenticated unicast connection.

**Figure 6.7:** While standard DNS-SD/mDNS multicasts all query and response packets revealing a lot of information about users and causing a lot of multicast traffic (a), our solution only multicasts protected information about the `_psds` meta service (b); all other services are queried directly using unicast (c). Private information is only accessible by chosen paired peers.

### 6.5 Efficient Two-Stage Privacy Extension

In Chapter 3 we showed privacy problems arising while using DNS-SD/mDNS, and presented a proof-of-concept solution in Section 6.4 showing that privacy in DNS-SD/mDNS can be granted with little effort. While this solution provides privacy with imperceptible additional CPU time, the solution we present in this chapter outperforms the proof-of-concept solution in privacy, network load, and user experience. It has become the basis for our Internet draft [HK16b], which has been adopted by the IETF dnssd working group, and thus is likely to become a standard for privacy-preserving service discovery.

Our solution adds *device pairing* to DNS-SD and divides service discovery into two stages: *private directory discovery* and *service exploration* via *private service directories* (Figure 6.7). *Pairing* provides *directory discovery* with means for mutual authentication, e.g. with an authenticated shared secret. *Directory discovery* provides *service exploration* with an authenticated connection. These components are independent with respect to means used for transmitting the necessary data. We detail these components in the following subsections.

#### 6.5.1 Pairing

The pairing step can either be realized by using our pairing daemon proposed in Chapter 5, leveraging a very convenient means of device pairing that can even be fully automated (in a trusted network) granting a fully configurationless experience; or by directly integrating one of the pairing methods described in Chapter 5.

Alternatively, any mechanism yielding pairwise secrets can be used, as the components — including device pairing — are independent.
6.5.2 Private Service Directory Discovery (first stage)

We realize the directory component as a Private Service Directory Server (PSDS), which is a lightweight DNS server running on each host supporting our privacy extension. Hosts offer their private service instances via their PSDSes.

The first stage’s task of our two-stage solution is discovering PSDSes running on paired peers (see Figure 6.8). To be discoverable via DNS-SD, the PSDSes offer a meta service of the type _psds._tcp.

Figure 6.8: First stage: PSDS discovery. To grant privacy, instance names and ports are obfuscated. In this example, Alice queries for available _psds service instances and receives five answers. She then checks whether some of these instances match her paired peers. We assume one service instance was offered by her friend Bob. For this matching _psds instance, she initiates the DNS-SD resolving phase. Her user interface will substitute Bob for psKvLG9SpAo=. (Analogous to the zone file syntax of Bind9, in this thesis we abbreviate domain names substituting @ for the origin if the origin is obvious in the respective context; see Chapter 3.)
Chapter 6. Efficient Privacy-Preserving DNS-SD

We introduce DNS-SD meta services as special services that are not offered by users or client software but by the service discovery subsystem itself and are intended for transmitting service discovery related data. The pairing data synchronization service proposed in Chapter 5 is another meta service. In the eyes of an unmodified daemon, a meta service looks like any other service. The meta service concept allows us to seamlessly integrate service information control in our privacy extension by using DNS-SD itself.

Various means of transmitting DNS-SD records corresponding to the `_psds._tcp` service instances can be used. For the privacy extension proposed in this chapter, we use mDNS (or standard DNS, respectively) with obfuscated instance names; in Chapter 4 we discuss other possible means of PSDS discovery, among them our Stateless DNS technique detailed in Chapter 8.

To allow hosts to efficiently discover `_psds` service instances in a privacy-preserving way, `_psds` instance names contain fast processable hints apprising hosts of the paired peer offering the corresponding service. If the hint does not match, the corresponding packet is dropped silently. These hints can either be based on a pairwise shared secret, or on a secret shared with all paired peers; it can also be a mix thereof, e.g. groupwise shared secrets. We discussed trade-offs between these options in Section 6.4.

Since the hints and thus the resulting ephemeral service instance name have a very low collision probability, our solution does not announce the `_psds` meta service, because reserving instance names is not an issue. In order to still immediately learn about available `_psds` instances, a host queries for paired peers’ `_psds` instances as soon as joining a network. For offering its own `_psds` service, the host sends a push message containing a resolve answer for its `_psds` instance to each of the discovered paired peers via the connection established to the paired peers’ PSDSes. This alternative way of announcing a service saves the relatively costly announcing process where a service instance has to be probed and announced thrice, respectively. It further halves the number of necessary browse queries for the `_psds` instance as our solution pushes the own instance as soon as discovering a paired peer’s one. The latter benefits all bidirectional service types.

Conditioning that `_psds` instances are only browsed when a host joins a network allows for a further efficiency enhancement: queries for `_psds` related resource records should always be sent with the QU (unicast answer requested) flag set. Further supporting the superiority of unicast answers for `_psds`, only very few peers would even profit from a multicast answer for resolve queries. Indeed, more than one online peer might be able to use a particular `_psds` instance; this, however, does not justify the use of a multicast message (see Chapter 4).

The means for protecting privacy in the first stage (directory discovery) of our two-stage solution are similar to the means used in the one-stage solution. Since the `_psds` service does not transmit any key-value pairs in TXT records, encryption is not necessary.\(^\text{10}\) The port in the SRV record can be obfuscated by XORing it with a hash of the shared secret. This helps mitigating directed DoS attacks on a PSDS. The hostname can be obfuscated according to [HTW16]. The second stage (service exploration) uses the privacy-preserving means of service distribution granted by the PSDSes.

In the reminder of this subsection we will detail different hinting options for the `_psds` service; efficient hinting is crucial for making private directory discovery viable.

\(^{10}\)In future versions, we might use encrypted key-value pairs.
6.5 Efficient Two-Stage Privacy Extension

Single-Hint

After pairing, hosts may provide another secret to the peer which is the same for all paired peers. If so, the host receives one such secret from each of its $m_t$ paired peers. We discuss a hinting method based on such secrets — the single-hint method — mainly for demonstration purposes. The single-hint method is inferior with respect to privacy and its efficiency gain does not justify its inferior privacy properties.

Given such secrets, a single hint identifiable by all paired peers is easily constructable. Using a single hint, it suffices to transmit $O(1)$ multicast messages for publishing a _psds instance. For the single-hint method, hosts construct their _psds instance names as shown in algorithm 2.

Algorithm 2: Create a single-hint _psds instance.

<table>
<thead>
<tr>
<th>Data: secret</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result: obfuscated_name</td>
</tr>
<tr>
<td>1 seed := rounded timestamp; // e.g. rounded to 5 minutes</td>
</tr>
<tr>
<td>2 long_hash := HASH(seed</td>
</tr>
<tr>
<td>3 instance_hash := first 4 bytes of long_hash;</td>
</tr>
<tr>
<td>4 obfuscated_name := BASE64(seed)</td>
</tr>
</tbody>
</table>

When a host receives a _psds service instance, it follows algorithm 3 to determine whether this service instance corresponds to a paired peer, and if so, to whom. While the host has provided a single hinting secret to each of its paired peers, it has received a dedicated hinting secret from each of its paired peers. Thus, each of those secrets may have been instrumental in constructing a received _psds service instance.

Algorithm 3: Process a single-hint _psds instance.

<table>
<thead>
<tr>
<th>Data: obfuscated_name, instance_hashes[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result: peer_ID (paired peer identifier)</td>
</tr>
<tr>
<td>1 seed := timestamp portion of the obfuscated name;</td>
</tr>
<tr>
<td>2 instance_hash := instance hash portion of the obfuscated name;</td>
</tr>
<tr>
<td>3 peer_ID := instance_hashes[seed][instance_hash];</td>
</tr>
<tr>
<td>4 if peer_ID != null then</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6 else</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8 end</td>
</tr>
</tbody>
</table>

The timestamp gives the advantage of making it sufficient to calculate long_hash only once every time interval the time stamp is rounded to. Choosing a coarser time stamp rounding is more efficient with respect to re-hashing, but enlarges the window of possible replay; this kind of replay, however, does not breach privacy, but might open pathways to

---

11Each host can be seen as a broadcaster (service provider) who provides a _psds instance to its paired peers; each of these paired peers needs the corresponding hinting secret for correctly associating this instance with the service provider. Since each host in the network can act as both provider and consumer, each host has a single hinting secret for providing its service instance and $m_t$ hinting secrets for associating the service instances provided by paired peers.
DoS attacks. The calculation of hash tables containing hints corresponding to timestamps is shown in algorithm 4.

**Algorithm 4**: Generate hint hash table.

```
Data: secrets[], peer_IDs[]
Result: instance_hashes[]

1. seed := rounded timestamp;  // e.g. rounded to 5 minutes
2. seed_old := seed that was used 2 iterations ago;
3. m_t := number of paired peers;
4. for i := 0; i < m_t; i := i+1 do
   5. instance_hashes[seed][HASH(seed | secret[i])] = peer_IDs[i];
5. end
6. delete instance_hashes[seed_old];
```

As described in Section 6.4 a single-hint yields the disadvantage of infeasible pairing revocation.

**Multi-Hint**

Harnessing the per-peer secrets exchanged during pairing for constructing per-peer hints, both a high level of privacy protection and an easy pairing revocation can be provided. The additional load is acceptable as it only concerns the browsing phase for _psds instances. For the multi-hint method, we introduce virtual service instances, which from the viewpoint of a standard resolver look like normal service instances; but several of these instances may represent a single real service instance. Virtual service instances can been as aliases for a real service instance. Introducing this concept is necessary, as for the multi-hint method, a host needs to offer a single _psds service instance, which is the real service instance, to each of its online paired peers using a different hint. The number of virtual service instances hosts offer at a certain point in time for representing their real _psds instances scales as O(m_t), where m_t is the number of paired peers. To reduce the number of virtual service instances necessary for representing a real service instance, a virtual service instance name contains multiple hints. Introducing our virtual service instances is fully compatible with DNS-SD/mDNS as the fact that they are only virtual is transparent to the service discovery subsystem. Using the multi-hint discovery method, hosts construct their _psds instance names as shown in algorithm 5.
Algorithm 5: Create a multi-hint \_psds instances.

**Data:** secrets

**Result:** obfuscated\_name

1. seed := rounded timestamp; \hspace{1cm} // e.g. rounded to 5 minutes
2. \( m_t := \) number of paired peers;
3. for \( i=0; i < \lfloor m_t/10 \rfloor; i := i + 1 \) do
   4. obfuscated\_name\[i\] := BASE64\( (\text{seed}) \);
5. end
6. for \( i=0; i < m_t; i := i + 1 \) do
   7. \( \text{long\_hash} := \text{HASH}(\text{seed} | \text{secrets}[i]) \);
   8. instance\_hash := first 4 bytes of \( \text{long\_hash} \);
   9. obfuscated\_name\[(i/10)] := obfuscated\_name\[(i/10)] | BASE64\( (\text{instance\_hash}) \);
10. end

Each instance name contains hints for up to 10 paired peers. If there are more pairings, multiple virtual service instances are necessary. When a host receives a \_psds instance, it performs the actions shown in algorithm 6.

---

Algorithm 6: Process a multi hint \_psds instance.

**Data:** obfuscated\_name, instance\_hashes

**Result:** peer\_ID (paired peer identifier)

1. timestamp := timestamp portion of the obfuscated name;
2. instance\_hash\_list := instance hash list portion of the obfuscated name;
3. foreach \( \text{ih} \) in instance\_hash\_list do
   4. peer\_ID := instance\_hashes[seed][ih];
   5. if peer\_ID \(!=\) null then
      6. process packet;
   7. else
      8. drop packet;
9. end
10. end

Direct Resolving

We introduced direct resolving while discussing the naive privacy extension in Section 6.4. Direct resolving can help improving the efficiency of PSDS discovery. Using the PSDS discovery method described above and illustrated in Figure 6.8, a host entering a network first browses for \_psds instances and then resolves the ones corresponding to its paired peers. But in contrast to standard mDNS, hosts gained prior knowledge during device pairing; a host knows of its paired peers. So instead of sending \_psds PTR queries causing paired peers to generate corresponding PTR instances with instance names only understandable by the host that generated the query, a host can create such instance names for all its paired peers and directly ask for the corresponding SRV and TXT records; meaning the querier creates obfuscated instances names instead of the responder. Figure 6.9 illustrates PSDS discovery leveraging direct resolving.
Figure 6.9: Alternative first stage: PSDS discovery using direct resolving. In this example, Alice directly tries to resolve the _psds instances of her three friends. Since only Bob is online, she only receives one answer. The result matches the standard procedure shown in Figure 6.8. Her user interface will substitute Bob for psKvLG9SpAo=.

When used in conjunction with standard DNS-SD/mDNS, direct resolving has the disadvantage of negating the possibility for discovering yet unknown services. But for _psds discovery this does not matter as each paired peer offers exactly one instance of this type. So there is definite prior knowledge about how many of these instances might be available, which is $m_t$; thus when asking directly for all $m_t$, 100% recall is achieved.

Further, since direct resolving prevents the listing of currently available _psds instances, it protects against a DoS attack where a malevolent host sends resolve queries for currently valid instance names forcing the service offerer to react.

We compare the efficiency of both methods in Section 6.8. Direct resolving is advantageous if a host does not have many paired peers, because direct resolving also queries paired peers that may not be online. Using the afore discussed indirect method, hosts learn about the online status of paired peers during the PSDS browsing phase.

6.5.3 Service Discovery via the PSDS (second stage)

After discovering the Private Service Directory Service of paired peers in the first stage, hosts query these PSDSes in the second stage to retrieve information (PTR, TXT, SRV) about private services offered by these paired peers; querying PSDSes is performed within a secure mutually authenticated session, guaranteeing the privacy of both parties.
6.5 Efficient Two-Stage Privacy Extension

![Diagram](image)

Figure 6.10: Ad-hoc protocol for establishing a secure, privately mutually authenticated session. $g^a$ and $g^b$ are ephemeral Diffie Hellman public keys of Alice and Bob, respectively; $prvSID_A$ is a privacy session ID based on the shared secret exchanged during pairing; it serves as a hint and apprises Bob of the fact that it is Alice who wants to initiate the connection. $apub_{Alice}$ and $apub_{Bob}$ are public keys that were authenticated during pairing. $aprv_{Alice}$ and $aprv_{Bob}$ are the corresponding private keys. Figure 5.15 shows an ad-hoc protocol for the same purpose using a shared secret instead of an asymmetric key pair.

Mutually Authenticated Secure Connection

To preserve the privacy of both service offerer and requester, we establish a secure, privately mutually authenticated session. PSDSes only answer queries associated with such a secure session. After discovering PSDSes, a host sends a session initiation request. The hosts learned the ports and IP addresses of the PSDSes during the first stage from the SRV and A resource records, respectively. To apprise peers of their own identity, hosts use hints discussed in the previous subsection. We showed an ad-hoc protocol for establishing a privacy-preserving mutually authenticated session using a Diffie-Hellman key exchange and a shared secret in Chapter 5 (see Figure 5.15). Figure 6.10 shows an ad-hoc protocol that could be used if an asymmetric key pair was exchanged during pairing. In both cases, messages associated with the private session are encrypted using authenticated encryption (AE) with keys derived from the established ephemeral Diffie-Hellman secret $g^{ab}$.

For a deployable implementation, we suggest to use DNS over TLS [HZH+16], using a perfectly forward secure cipher suite, preferably (EC)DHE-PSK. Using (EC)DHE-PSK, the session is realized by a secure connection leveraging a Diffie-Hellman key exchange authenticated by a pre-shared secret, which, for our privacy extension, is the shared secret exchanged during pairing. The `psk_identity`, hinting which shared secret should be used, has to be constructed as follows in order to preserve the privacy of both parties.

```c
struct {
    uint32 gmt_unix_time;
    opaque random_bytes[4];
} seed;
long_hint = HASH(seed | secret)
hint = first 12 bytes of long_hint
psk_identity = BASE64(seed) "." BASE64(hint)
```
Chapter 6. Efficient Privacy-Preserving DNS-SD

Alice sends browsing queries to the PSDS running on Bob’s device and then asks to resolve the corresponding service instances. She also directly resolves Bob’s \_presence instance. She does not resolve the hostname again, because she already learned about the corresponding A resource record during directory discovery.

The standard way does not protect the privacy of the host initiating the connection. A more detailed explanation of how to initiate a TLS connection to a PSDS can be found in our Internet draft [HK16b].

Service Exploration

Service exploration is performed by sending DNS-SD queries to PSDSes, light-weight DNS name servers running on each host supporting our privacy extension. These DNS servers answer queries corresponding to private service instances offered by the host they run on. The queries and responses corresponding to the service browsing and service discovery phases of DNS-SD are sent via the secure unicast channel described in the previous subsection. The address resolution phase has already been performed during PSDS discovery. Figure 6.11 shows an example of private service discovery leveraging a PSDS. Since a PSDS is a name server compatible to standard DNS, the second stage also integrates seamlessly into the existing DNS architecture.
The PSDS should support DNS push notifications [PC15], e.g. to facilitate an up-to-date contact list in a chat application without polling.

**More Efficient Resolving Methods**

Similar to the above discussed *direct resolving*, the *service exploration* phase can also harness more efficient resolving methods. We propose various methods, and discuss their advantages.

*Unsolicited Service Info Push:* As soon as the connection to a paired peer’s PSDS is established, the paired peer’s PSDS unsolicitedly pushes all service related information, i.e. the SRV and TXT resource records corresponding to the offered services. As soon as service related information changes, the corresponding update is pushed. This method is straightforward and very efficient for a small number of offered service instances, as it saves both the browsing phase and all query messages. Further, service instances offered by friends are very likely to be of interest.

*Unsolicited Service List Push:* Instead of pushing all service related information, the paired peer’s PSDS pushes a list of available service instances. This corresponds to receiving unsolicited browse answers for all offered service instances. The PSDS further pushes updates of the service list.

*Unsolicited Service Type Push:* After establishing the connection, the paired peer’s PSDS only pushes a list of offered service types. The pushed information corresponds to the answer to the standard browsing query for _services._dns-sd._udp, which lists all available service types. The PSDS further pushes updates of the type list.

*Selective Querying & Direct Resolving:* When querying PSDSes, hosts send queries directly to selected paired peers instead of asking every peer. Prior knowledge about who is likely to answer queries can be leveraged to make service discovery even more efficient. Firstly, a peer offering a certain service instance is likely to also offer this service instance at a later point in time. To utilize this fact, service instance names can be cached allowing for direct resolving omitting the step of service browsing (the _presence instance in Figure 6.11 serves as an example for direct resolving). The resolving step should not be cached as especially with mobile nodes network parameters are prone to change. Mimicking the concept of static service instances, static service instance names could also be exchanged during pairing. Secondly, a user can choose for certain service types to only query select paired peers or user defined groups of paired peers. For a synchronization service, a user might only want to query her own devices. Thirdly, if a host wants to browse a service type and learns during _psds discovery that none of the corresponding paired peers is online, the host does not even have to initiate browsing for this service type. In all these three cases the service browsing step is omitted or significantly reduced, respectively. In large discovery scopes reducing service browsing significantly helps reducing network load.

We propose the following combination as a solution: If a peer offers less than 20 service instances, it directly pushes all service information. If it offers between 20 and 100 service instances, it pushes a service instance list. If it offers even more, it only pushes a type list. Regardless of the number of offered services, PSDSes push updates of service information that was transmitted before. But a host may send one-shot queries, whose corresponding answer is not to be updated. For our analysis in Section 6.8, we only
consider the unsolicited push of all service information. In the future, we plan to utilize 
selective querying to further reduce the network load caused when querying peers that 
offer a lot of service instances.

Continuous Querying & Push Answers. Standard DNS-SD/mDNS uses continuous 
querying, i.e. sending the same query after a certain increasing time interval has passed, for 
providing an up-to-date service list. The multicast answers refresh corresponding resource 
records not only in the queriers caches, but also in the caches of all peers, which is helpful 
for services a lot of peers are interested in.

Our directory discovery, however, does not profit from continuous querying as a single 
_psd instance is only meant for very few peers and, even more importantly, the service 
has to be only discovered once when joining the network. After discovering the _psd service, 
a secure connection is established and all further discovery related messages are 
transmitted via this connection. For service exploration we achieve an up-to-date service 
list via push messages, which is more efficient in terms of network bandwidth utilization 
and is truly up-to-date because updates to service information are pushed as soon as they 
become available.

Separating directory discovery and service exploration allows substituting push answers 
for continuous querying, which yields another efficiency advantage of our solution.

Reducing Multicast. A further advantage of our solution is reducing multicast; service 
exploration is performed via unicast. This saves network utilization, especially in 
WiFi networks, and improves privacy, because even encrypted multicast allows inferring 
information about offered and requested service instances. Traditionally, services that are 
restricted to a small user group, e.g. file-sharing, are implemented by selectively handing 
trusted users’ access credentials, mostly in the form of a user-specific or shared password. 
Thus, information about services that can only be used by a few users is published to the 
whole local network, only to deny almost everyone access later. We do not want to replace 
the authentication process for those services; we want to hide information about those 
services from the public, because those services are not meant for the public in the first 
place. According to Stuart Cheshire [SC06], DNS-SD/mDNS is used to query services of a 
type one can sensibly use; services the querier can merely connect to, but not understand 
their protocols, should not be offered. We extend this by also not offering services to 
someone who is not allowed to access.

6.6 Implementation

For the two-stage solution we have a proof-of-concept implementation\textsuperscript{12}, which — like our 
one-stage privacy extension — is based on the open source Zeroconf daemon Avahi\textsuperscript{13}. As 
for the one-stage solution, we changed very little code in the avahi-daemon source files, 
just adding hooks to our loosely coupled privacy module. The current implementation is 
based on our previous work [KW14c]. We also have a proof-of-concept implementation 
leveraging Stateless DNS (Chapter 8) as means for directory discovery, which is based on 
[KRWS15]. Both implementations are transparent to existing client software.

\textsuperscript{12}https://gitlab.com/kaiserd/avahi-p2stage
\textsuperscript{13}http://avahi.org
6.7 Privacy and Security Considerations

We are working on a deployable implementation reflecting the state of this chapter. A thorough description for implementers can be found in our Internet draft [HK16b].

6.7 Privacy and Security Considerations

Our two-stage solution proposed in this chapter protects both the service querier and the service provider in both the directory discovery stage, during which the querier discovers the private service directory provided by the service provider’s device, and the service exploration stage, during which the service querier retrieves service related information — most importantly the network parameters — from the service provider’s private service directory.

Privacy and security of our solution depend on the shared secret established during pairing (see Chapter 5). Building on this shared secret, privacy and security during directory discovery and service exploration are guaranteed by the following means.

Ephemeral Identifiers: During directory discovery, privacy is protected by reducing the transmitted information to a minimum and by making it indistinguishable from randomly chosen data for unpaired peers. This minimal information consists of an identifier, which allows the receiver to tell whether the corresponding packet is meant for him, and network parameters, which allow the querying client to connect to the private directory service offered by the service provider’s device. The identifier is derived from the pairing secret by hashing the secret alongside a timestamp using a cryptographic hash function (we discussed cryptographic hash functions in Chapter 5). This yields ephemeral identifiers which are only valid in a time interval and neither allow unpaired peers to derive an identity nor allow linking to previous identifiers; thus, these ephemeral identifiers do not breach the user’s privacy. These identifiers are not relevant to security as their purpose is merely pointing out which key to use for decrypting the network parameters. Our solution encrypts the network parameters, and possibly further service related information, with the secret exchanged during pairing. The encryption provides all of confidentiality, integrity and authenticity because the encryption key has been authenticated during pairing; privacy is provided via confidentiality.

Mutually Authenticated Encrypted Connection: The connection through which service exploration is performed, is secured via an ephemeral Diffie-Hellman secret, whose exchange in turn has been authenticated using the secret exchanged during pairing. Since we rely on TLS PSK (using the pairing secret as PSK) for establishing this secure connection through which service related information is transmitted, our solution inherits the security of TLS-based data transmission, including privacy as confidentiality is granted. We protect the privacy while establishing this connection introducing a private mutually authenticated connection establishment by using ephemeral identifies (similar to the ones used for directory discovery) instead of identity-revealing key identifiers. Even if the security measures during directory discovery (whose sole purpose is protecting the users’ privacy) were bypassed by just randomly trying to establish a connection to a private directory service, this attempt would neither allow the attacker to receive any service related information, nor would it allow inferring information about the service provider’s identity.

14Availability, which is also a key part of security, has to be provided by the means for distributing the resource records, e.g. mDNS.
Our solution protects the users privacy on the application layer and assumes that deeper layers of the network protocol stack do not breach the privacy. This is especially important for link-layer identifiers; thus, mechanisms such as the MAC-address randomization should be used in conjunction with our solution for preventing the tracking of users.

6.8 Network Performance Analysis

Users do not want to pay for privacy by complicated configuration, but they also do not want to pay by performance loss. One of the main goals of our privacy extension is to be at least as efficient as standard DNS-SD/mDNS in terms of both network and host performance. Our privacy extension does not increase the network load; in fact we reduce it by using mostly unicast instead of multicast. Further the computational overhead on the host devices is imperceptible for the user, both in terms of responsiveness of the system and battery life.

In this section we show that for private services, i.e. services that are meant for a set of friends and not for the whole public, our solutions outperform standard DNS-SD/mDNS with respect to network performance, while providing privacy protection. The increase in efficiency boils down to the fact that our solution separates directory discovery and service exploration; for the latter, it selectively sends discovery messages to paired peers instead of everyone.

Our analysis uses the following variables and their respective default values

- \( n \), the number of hosts participating in service discovery. We look at network sizes ranging from 1 to 1000 nodes.
- \( m_t \), the average number of total paired peers per host. We assume an average of \( m_t = 50 \) as default.
- \( m \), the average number of online paired peers per host. We assume an average of \( m = 5 \) as default.
- \( n_o \), the number of nodes joining (and leaving) the network per second. The arrival rate \( n_o \) is calculated as \( n_o = n/t_o \), where \( t_o \) is average time a host stays online. We assume that hosts stay online for 10 minutes (600s) on average, which results in an arrival rate \( n_o \approx 1.7 \) for \( n = 1000 \).
- \( s_o \), the average number of service instances offered by a host. We assume an average of \( s_o = 5 \) as default.
- \#types, the number of service types available in the current network. We assume an average of \#types = 10 as default.

We assume that all hosts support our privacy extension and thus offer a \_psds\_ instance. Further, we assume that hosts offer and request all desired services as soon as they enter a network. This assumption does not sacrifice much generality as for the overall load it does not matter whether hosts offer a certain amount of service instances distributed over the online time or right when joining the network. Our assumption causes higher bursts when hosts come online, but firstly this is much more realistic than an even distribution as in real-world scenarios most services are offered as soon as joining the network, and secondly, coping with queries of peers is much more of a concern than the relatively few instances offered by a single host.

In the following, we analyze both the hinting methods proposed in Section 6.5 and standard DNS-SD/mDNS. We analyze the load a single host is exposed to, and further,
6.8 Network Performance Analysis

divide the analysis in two parts. Firstly, we analyze the load a host is exposed to when entering a network, and, secondly, while sojourning in a network.

6.8.1 Analyzing the Number of Necessary Discovery Actions

In this subsection we analyze the number of discovery actions necessary a host has to perform when joining a network (and while staying in a network). In this respect, we analyze our directory discovery methods (single-hint, single-hint direct, multi-hint, and multi-hint direct), our service exploration, and standard DNS-SD/mDNS. We define a discovery action as one of announcement, browse query, browse answer, resolve query, and resolve answer. Since these discovery actions — except for the announcement — cause roughly the same network load, for simplicity’s sake, we do not distinguish between them throughout our analysis. Resolve answers might indeed be significantly larger due to large TXT records, but not for the discovery of our _psds instances. Announcements, which are only used by standard DNS-SD/mDNS, consist of three probes and three announcing messages, respectively.

Further, we do not consider continuous querying, which is only used by standard DNS-SD/mDNS for keeping an up-to-date list of available service instances. We also did not take the lower transmission rate of multicast into consideration (see Chapter 4). To be able to quickly identify discovery actions involving multicast we color-code these actions in red.

Since we want to show the superiority of our solution for discovering private services, if there is a bias for simplicity’s sake, it is in favor of standard DNS-SD/mDNS.

Single-Hint PSDS Discovery

A host joining the network browses for _psds instances and gets $n$ answers because each peer offers one _psds instance. Since the host only wants to connect to paired peers, it resolves $m$ of these _psds instances, and receives $m$ corresponding answers via unicast\textsuperscript{15}.

Summing up, a host joining a network elicits
discovery actions directly affecting it, which are comprised of queries the host sends and answers the host receives.

While being part of a network, a host is affected by discovery actions caused by peers joining the network. Since $n_o$ peers join the network per second and actions sent via multicast affect each host, a host that stays in a network is affected by $n_o$ times the number of multicasts each joining peer sends. The unicast actions sent by the $n_o$ peers joining per second each have a probability of $1/n$ to affect a certain host, as we assume an even distribution.

Thus, the discovery actions a host staying in a network is involved in sum up to

\textsuperscript{15}As stated and discussed above, our mechanisms always set the QU (unicast answer requested) flag.
which corresponds to the above formula with a factor $n_o$ for multicast actions and a factor $n_u/n$ for unicast actions. These discovery actions comprise queries the host receives and corresponding answers the host sends.

Because the resolve queries are answered via unicast and these resolve queries are to be answered by a specific paired peer, we could send the resolve query via unicast to the source address retrieved from the PTR answer’s IP packet (to the mDNS port 5353) instead of multicasting it as standard mDNS does. This would further influence the analysis in favor of our solution. However, for our analysis we did not leverage information in the IP packet, which is the cleaner approach as the semantics conveyed by providing such a PTR resource record is the mere existence of a service instance. Using only the intended meaning of browse answers is compliant to standard DNS-SD/mDNS and further allows delegating the provision of PTR records to arbitrary hosts, e.g. to a proxy.

Single-Hint Direct PSDS Discovery

Using direct resolving, both the announcing and the browsing step are skipped. This yields the disadvantage of not knowing who is online before resolving, but greatly reduces network utilization in certain situations (see discussion below). Instead of browsing, the host sends resolving queries corresponding to all its $m_t$ paired peers. Since only $m$ paired peers are online, it will receive $m$ answers. The involved discovery messages sum up to

$$\sum_{\text{resolve query}} m_t + \sum_{\text{resolve answer}} m.$$

The number of discovery messages a host has to deal with while staying in the network can be derived analogous to the way described above

$$\sum_{\text{resolve query}} n_o \cdot m_t + \sum_{\text{resolve answer}} n_o \cdot m.$$

A further efficiency enhancement we do not consider is sending a single query which contains a hint based on the secret the host gave to its paired peers, instead of $m$ queries with $m$ hints based on the $m$ secrets the host received from its paired peers. However, this is not backwards compatible, as it sends a query for a nonexistent service instance, which elicits answers that were not requested from a DNS point of view.

Multi-Hint PSDS Discovery

Using a different hint for each paired peer multiplies the number of offered instances by $[m_t/10]$ because hosts offer virtual _psds instances where each instance name contains hints for up to 10 paired peers.\(^\text{16}\)

When joining a network, a host sends a browse query for _psds instances, which will cause $n \cdot [m_t/10]$ answers, because each online peer offers its real PSDS service via $[m_t/10]$ virtual service instances.

\(^\text{16}\)From the mDNS viewpoint, each host offers several service instances, but they contain information about only one actual service instance, namely the PSDS running on the respective host.
The host will then ask to resolve $m$ of these instances, because there are $m$ online paired peers that each offer a single service instance containing a hint the host understands. These $m$ resolve queries will cause $m$ resolve answers. In total this yields

$$\frac{1}{\text{browse query}} + n \cdot \left\lceil \frac{m t}{10} \right\rceil + \frac{m}{\text{resolve query}} + \frac{m}{\text{resolve answer}}$$

discovery messages. A host staying in the network is involved in

$$\frac{n_o}{\text{browse query}} + n_o \cdot \left\lceil \frac{m t}{10} \right\rceil + \frac{n_o \cdot m}{\text{resolve query}} + \frac{n_o \cdot m}{n \text{ resolve answer}}$$

discovery actions per second.

As with the single-hint method, we could send the resolve queries via unicast to further reduce the network load. Further, the not standard compliant method of sending a query for a nonexistent service instance which elicits several answers can also be applied to the multi-hint method. Instead of sending $m$ resolve queries, $\left\lceil \frac{m}{10} \right\rceil$ queries for nonexistent service instances with instance names containing 10 hints can be sent, each of them eliciting 10 resolve answers. This is not standard compliant because each of the $m$ paired peers offers a dedicated _psds instance, which normally requires $m$ dedicated queries.

### Multi-Hint Direct PSDS Discovery

Like single-hint direct PSDS discovery, multi-hint direct PSDS discovery neither involves an announcement phase nor a service browsing phase. Because direct resolving moves the process of interpreting hints from queriers to responders (from clients to servers), and for each host there are $m_t$ possible responders, it suffices to send $m_t$ direct discovery messages resulting in the same number of messages as needed for the direct single-hint method. A host joining a network elicits

$$\frac{m_t}{\text{resolve query}} + \frac{m}{\text{resolve answer}}$$

discovery messages, while a host staying in a network is involved in

$$\frac{n_o \cdot m_t}{\text{resolve query}} + \frac{n_o \cdot m}{n \text{ resolve answer}}$$

discovery messages per second.

The not standard compliant method of sending a query for a nonexistent instance which elicits several answers can be applied in the same way as for the multi-hint non-direct method.

### DNS-SD via the PSDS

For service exploration, i.e. DNS-SD via the PSDS, we only take the unsolicited push of all available service information into consideration. This simplifies our analysis, but our adaptive solution discussed above reduces network traffic in the case of paired peers offering a lot of services.
A host that has joined the network and has completed the directory discovery phase by one of the above described methods, finishes service exploration by just receiving an unsolicited push of all possible resolve answers, which yields

\[ \frac{m \cdot s_o}{\text{resolve answer}} \]
discovery messages because each of the \( m \) online paired peers offers \( s_o \) service instances.

A host staying in a network has to push this information to paired peers that join the network, which sums up to

\[ \frac{n_o \cdot m \cdot s_o}{\text{resolve answer}} \]
discovery messages per time unit.

**Standard DNS-SD/mDNS**

Standard DNS-SD/mDNS only comprises a single phase for both directory discovery and service exploration. A host that joins a network directly announces the services instances it wants to offer (\( s_o \)).

Because we want to analyze the network impact of discovering the same amount of services via standard DNS-SD/mDNS and our privacy extension, respectively, we assume a host wants to discover the \( s_o \cdot m \) service instances offered by 'friends'. To achieve the same recall as our privacy extension, the host has to list all service instances available in the network, as it has no prior knowledge about which of these might be of interest. Even when listing all service instances, two major disadvantages arise: (1) the names of the service instances do not have to uniquely tell which friend offers the corresponding service, and (2) the user has to manually choose among a myriad of offered service instances. We will analyze browsing for both all service instances and a certain percentage \( p_s \) (lowering network utilization, but also recall).

After announcing its services, the host sends browse queries for each service type it is interested in, eliciting browse answers for a certain percentage of the service instances offered in the network. The host then sends \( m \cdot s_o \) resolve queries for the service instances offered by friends. The percentage of these service instances the host is able to browse depends on the percentage of listed service instances. This sums up to

\[ \frac{s_o}{\text{announce}} + \frac{\text{\#types}}{\text{browse query}} + \frac{p_s \cdot n \cdot s_o}{\text{browse answer}} + \frac{p_s \cdot m \cdot s_o}{\text{resolve query}} + \frac{p_s \cdot m \cdot s_o}{\text{resolve answer}} \]
discovery messages. Resolving answers are the same as for our privacy-preserving methods. However, while the privacy extension can leverage push messages and does not need resolving queries at all, standard DNS-SD/mDNS needs one multicast query for each service instance it wants to resolve.

A host staying in the network is involved in

\[ \frac{n_o \cdot s_o}{\text{announce}} + \frac{n_o \cdot \text{\#types}}{\text{browse query}} + \frac{p_s \cdot n_o \cdot s_o}{\text{browse answer}} + \frac{p_s \cdot n_o \cdot m \cdot s_o}{\text{resolve query}} + \frac{p_s \cdot n_o \cdot m \cdot s_o}{\text{resolve answer}} \]
discovery messages per second.
6.8 Network Performance Analysis

(a) Smaller networks with up to 100 hosts.  
(b) Larger networks with up to 1000 hosts.

Figure 6.12: Number of network actions per second a host is involved in while staying in a network, using our default values of $m_t = 50$, $m = 5$, $s_o = 5$, #types$= 10$.

Even if the unicast answer requested bit (QU bit) is set, responders should still send a multicast answer if the corresponding resource records have not been multicast in a time frame corresponding to a quarter of the records’ TTLs (see [CK13b] Section 5.4). This — as well as multicast answers in general — is beneficial only if the distributed service information is interesting for many peers, which is not the case for private services. Suppression techniques generally do not kick in when distributing information about private services among all peers. With respect to network load, our formulae for standard DNS-SD/mDNS represent a very favorable scenario. Multicast answers would additionally affect each host staying in the network almost without any benefit. For the browse answer and the resolve answer the factor $1/n$ would have to be removed. However, the linear influence of $n_o$ would stop at a certain point, because DNS-SD/mDNS sends multicast packets with a short delay to avoid sending the same information twice in a short time interval (see [CK13a]).

6.8.2 Comparing the Number of Necessary Discovery Actions

Figure 6.12 compares the above described methods for the discovery of private services, i.e. services that are only meant to be used by a limited set of friends. The plots illustrating our solutions comprise both the discovery actions necessary for directory discovery and service exploration. Since we only use a single service exploration method in our analysis, we name our methods according to the directory discovery methods.

Using Standard DNS-SD/mDNS in conjunction with multicast answers causes a large amount discovery actions. Each multicast query elicits $n$ answers that each host in the network has to deal with,\(^\text{17}\) leading to $O(n^2)$ discovery actions. Listing only 50\% of the available services reduces the number of discovery actions, but does not justify the reduced recall.

When always using unicast answers — which standard mDNS normally does not do — the number of necessary discovery actions becomes linear and comparable to our solutions. Still, our single-hint and multi-hint methods need significantly less discovery actions, because only a single service instance has to be discovered using multicast. Using our default parameters, our multi-hint direct method performs worse compared the standard

\(^{17}\)Dealing with these answers might be just ignoring them; nevertheless, it uses a part of the available network bandwidth.
and our indirect methods. This is due to the fact that independent of the number of currently online paired peers, a multicast query is sent for each paired peer. However, in the following analysis, we show that in many cases, this is a viable method as it has the best scaling properties.

### 6.8.3 Analyzing the Influence of Various Parameters

Analyzing the influence of various non-default values for our above defined variables, we parameterize our default plots in various ways. Our non-default plots are shown in Figure 6.13, which alters \( s_o \) and \#types, and Figure 6.14, which alters \( m_t \) and \( m \).

Changing \( s_o \), the average number of services offered per host, has a much more significant influence on solutions that do not separate directory discovery and service service exploration. For standard DNS-SD/mDNS, a larger \( s_o \) increases the number of discovery actions a host staying in a network has to deal with by a significant factor, because each part of the discovery process directly depends on the number of offered service instances. Each service instance has to be announced, and each service instance causes a further browse answer, a further resolve query, and a further resolve answer per host. For our solution, only service exploration is influenced by the number of offered service instances, which is handled by a single push per host. Because directory discovery is independent of the number of offered services, our solutions scale much better with respect to \( s_o \).

The number of service types in the network does not influence our solutions at all, as our service exploration method that is subject to this analysis, does not involve a type browsing phase. For standard DNS-SD/mDNS it determines the number of necessary browse queries. If hosts were to browse for specific types only, a larger number of types would allow for a more fine grained browsing phase. However, for discovering a set of private service instances offered by friends, which could be of arbitrary types, the number of types only increases the number of browse queries. As the number of types only affects the number of browse queries, the influence of this variable is not significant.

Changing \( m_t \), the average number of total paired peers of a host, does not have any influence on standard mDNS as mDNS does not use any information which is not about nodes online in the current network. Our single-hint solution is also not influenced by \( m_t \), as each host only offers a single \_psds instance. The multi-hint solution’s browse answers are affected, because a host does not know the sender of a browse query, and thus has to provide (via unicast) virtual \_psds instances containing hints for all paired peers. The most significant influence of changing \( m_t \) is on the multi-hint direct solution because it has to multicast resolve queries for each paired peer regardless of its online status. For large paired peer lists it is better to browse for available \_psds instances, and not resolve them directly; \( m_t \) could even be larger than \( n \).

The number of online peers (\( m \)) influences all methods significantly — except our multi-hint direct method — as it determines both the number of directory parts that have to be discovered as well as the number of service instances that have to be explored. For standard DNS-SD/mDNS the effect is very similar to increasing the number of services offered by each friend, because in both cases, (1) discovering a lot of service instances from a few friends or (2) discovering a few service instances from a lot of friends, results in
6.8 Network Performance Analysis

Figure 6.13: Number of network actions per second a host is involved in while staying in a network, using various values for $s_o$, and #types. If not further specified, we use our default values of $m_t = 50$, $m = 5$, $s_o = 5$, #types = 10.
Figure 6.14: Number of network actions per second a host is involved in while staying in a network, using various values for $m_t$, and $m$. If not further specified, we use our default values of $m_t = 50$, $m = 5$, $s_0 = 5$, #types= 10.
the same number of overall instances that need to be discovered. The network impact of standard DNS-SD is determined by the number of overall service instances and the number of hosts in a network. For our multi-hint direct method, the effect is negligible because the direct resolve queries are sent for each paired peer independent of whether a peer is online or not.

Figure 6.15 provides plots with a fixed $n$ of 100, and 1000, respectively, with a variable $m$ in the range of 1 to $m_t$ using our above defined default values of $m_t = 50$, $s_o = 5$, and #types= 10. It shows that while our multi-hint direct method is almost constant, the standard methods suffer from a large number of online peers and the resulting large number of available service instances.

Summarizing, our solutions outperform standard mDNS-SD with respect to necessary discovery actions for discovering private services that are only to be accessed by a few paired devices. Our multi-hint direct method shows the best scaling behavior. However, it suffers if the $m_t/n$ ratio is large, meaning if a host with a large list of paired devices wants to perform service discovery in a relatively small network. To overcome this problem, a host can choose either method based on the network size, which it can roughly estimate by listening to the multicast traffic. Further, we work on reducing hint sizes using methods based on Bloom filters. We did not consider concrete network aspects, namely packing multiple queries and answers in a single packet, and the fact that multicasts do not only affect every peer, but are transmitted at a lower rate to boot. While the former helps both standard DNS-SD/mDNS and our solution, the latter makes standard DNS-SD/mDNS even less efficient, as our solutions need significantly less multicast messages.

### 6.8.4 Unicast versus Multicast

Compared to DNS-SD/mDNS, we reduce the number of necessary multicast packets. Since other components of our service discovery framework also reduce multicast, using few unicasts to select devices instead, we analyzed and compared unicast and multicast in Chapter 4. Table 6.2 illustrates the influence of the various combinations of queries and answers via unicast and multicast.

\addtocounter{footnote}{2}

\footnote{From a concrete implementation’s point of view the case of a few friends offering a lot of instances is more efficient as the respective answers can be accumulated in a lower amount of packets.}
Table 6.2: Discovery actions caused by the querying host and the number of affected peers in a network with \( n \) peers, assuming queried peers will answer. Multicast answers affect all peers, a large number of which might neither be interested in that answer at the time of receiving the answer, nor at any later point in time.

6.8.5 Host Performance Analysis

Our privacy extension has an imperceptible computational time overhead compared to standard DNS-SD/mDNS, which, even on mobile devices, vanishes compared to packet delays of DNS-SD/mDNS that are in the range of 20 to 120 milliseconds. These delays mitigate the problem of multicast bursts when multiple devices are reset at the same time [CK13b].

6.9 Privacy-Preserving Hostname Resolution

Besides protecting service related information, our solution also protects the privacy during hostname resolution via mDNS without the need for additional messages.

During pairing, every user assigns names to paired peers — either manually entering the name or using the name provided by the peer via the pairing protocol.

Privacy preserving hostname resolution takes place during directory discovery. When discovering a \(_{psds}\) service instance a host is able to derive the service provider’s identity from the randomized instance name if a pairing has been established with this service provider. When resolving the randomized hostname contained in the SRV resource record, the host obtains the desired A (and AAAA) resource record of the service provider. Combining this information, the host can associate the service provider’s name (obtained during pairing) with service provider’s A (and AAAA) resource record.

A client application can just substitute a domain name containing this name, e.g. \(<\text{name}.privacy.local\rangle\), for the domain name containing the randomized hostname. This allows the local resolver to resolve this name. The name is never transmitted over the network for resolution. This allows establishing a privacy-preserving resolvable name with minimal overhead, and can even be used by hosts that do not engage in service discovery at all.

The process of Alice resolving Bob’s name via mDNS comprises the following steps.

- Bob offers a PTR and a SRV resource record for his \(_{psds}\) service instance, and an A resource record associated with his randomized mDNS name, e.g. “AB1234EF.local”.
- Upon receiving Bob’s PTR resource record, Alice processes the hint contained in the service instance name and learns that it is Bob’s service instance.
- Alice resolves this service instance retrieving the corresponding SRV resource record. She then resolves the randomized hostname, “AB1234EF.local”, obtaining Bob’s A resource record.
- The daemon on Alice’s device updates the local DNS cache, and creates a CNAME record, establishing “AB1234EF.local” as a canonical name for “bob.privacy.local”.
• Applications on Alice’s device can now simply use
gethostbyname("bob.privacy.local")\textsuperscript{19} for resolving Bob’s private host name.

6.10 Privacy-Preserving DNS-SD/DNS

Our two stage approach to privacy-preserving service discovery is independent of the
means used for transmitting DNS resource records. Among other such means — besides
mDNS — as of yet we considered standard DNS and our Stateless DNS technique (see
Chapter 4 and Chapter 8); using distributed hash tables and means not directly related to
DNS is also an option.

In this section we want to discuss applying our privacy extension to DNS-SD over
standard DNS, which is straightforward and works without any changes. For the first stage,
hosts supporting our privacy extension can publish their \texttt{_psds} instances via standard DNS,
which makes the Private Service Directories discoverable via wide area service discovery.
The second stage does not change at all. Service exploration via the PSDSes still works by
establishing a mutually authenticated session and direct querying.

Since DNS-SD/DNS allows service discovery in the whole Internet — in contrast to
mDNS which is confined to one multicast domain — we have to take NAT \cite{SH99} traversal
into consideration. Possible solutions are discussed e.g. in \cite{MHWW11, RAB+15}. We are
working on a further solution based on Stateless DNS (see Chapter 8).

Our multi-link DNS-SD extension described in Chapter 7 supports scaleable scopes
and thus supports arbitrary DNS-SD scope sizes between single-link local networks and
the whole Internet, e.g. a multi-link network of a campus.

6.11 Conclusion and Future Work

In this chapter we introduced our two-stage privacy extension allowing configurationless
service discovery without multicasting private data, while being efficient and transparent.
It is very user-friendly both in terms of overhead and control, and significantly reduces
the number of multicast messages sent, making DNS-SD/mDNS more efficient. A major
advantage of our extension to existing solutions (see Section 6.2) is that none of the
existing protocols in deeper layers of the network stack and none of the existing client
software has to be altered. Only the Zeroconf daemon running on the users’ devices has
to be modified, while afterwards still being able to exchange service information with
unmodified daemons. While our privacy extension provides means to publish services in
a privacy-preserving way, it still allows publishing nonsensitive services to all users in a
network via multicast, and thus is fully backwards compatible.

In the future, we will continue working on our Internet draft that specifies our DNS-SD
privacy extension. In addition, we plan to work on both more efficient hinting methods,
e.g. based on Bloom filters, and selective querying to further increase performance and
scalability.

\textsuperscript{19}This Unix function is obsolete and just serves the purpose of demonstrating the simplicity of resolving
Bob’s private local domain.
What we know is a drop, what we don’t know is an ocean.

Isaac Newton

7

Scalable Scopes for DNS-SD

Contents

7.1 Introduction .................................................. 182
7.2 Related Work .................................................. 183
7.3 Requirements ................................................. 184
7.4 DNS-SD over Stateless DNS ................................. 184
7.5 Implementation .............................................. 190
7.6 Analysis ......................................................... 192
7.7 Conclusion and Future Work ............................... 199

DNS Service Discovery over Multicast DNS (DNS-SD/mDNS) is widely used for configurationless service discovery in local networks. But since it is based on IP multicast, services are only resolvable within a single multicast domain. In addition to that, many institutions completely disable multicast for efficiency reasons rendering current local service discovery solutions useless. To distribute service information without multicast, DNS-SD can be used in combination with an authoritative DNS Server; but this demands configuration for both server and clients.

This chapter proposes DNS-SD over Stateless DNS (DNS-SD/sDNS), a solution that allows efficient, configurationless service discovery in multi-link networks and allows offering service information in arbitrary self-named scopes without configuration by leveraging Stateless DNS and the Raft consensus algorithm.
Chapter 7. Scalable Scopes for DNS-SD

7.1 Introduction

Configurationless service discovery is omnipresent as it is essential for convenient interconnection and communication of today's variety of devices in the edge of the Internet. A widely used configurationless service discovery solution is DNS Service Discovery [CK13a] over Multicast DNS [CK13b] (DNS-SD/mDNS), which we introduced in Chapter 3. It allows users to detect printers and streaming devices, to share data, and to communicate with others in a very convenient way. A particular benefit is that services can be requested and offered using DNS resource records, leveraging the solid and well established DNS; thus all means of offering and requesting DNS records can also be used for service discovery. DNS based service discovery cannot only be used in local networks leveraging mDNS but also — losing the configurationless-property — in the Internet using standard DNS servers (DNS-SD/DNS).

While the current means of DNS-SD distribution are appropriate for single-link local networks and the Internet, there is no efficient, user-friendly means of DNS-SD distribution for multi-link networks, used e.g. in universities or other institutions. Since link-local multicast packets are not propagated across routers, devices in different subnets cannot exchange service information using mDNS. Even if the routers propagated the messages, multicast-based solutions would not scale. For bandwidth conservation, many institutions deactivate multicast in their WiFi network denying mobile users the benefits of local service discovery. On the other hand, DNS-SD/DNS would pose an unacceptable configuration overhead and would not scale if every user was allowed to offer services. Where suited, an institution could use DNS-SD/DNS to offer a fixed number of services to its members.

To enable communication among the myriad of smart end user devices and thereby becoming an enabling technique for edge-centric computing [GLME+15], DNS-SD needs user-friendly, decentralized means to distribute resource records in multi-link networks. If not, users are forced to trust central service directory providers as soon as multicast is not enabled or services have to be discovered across links.

The Zeroconf community has reached consensus that adding multi-link support to DNS-SD/mDNS is necessary [LCBM15]. Apart from the community consensus, a petition

\[ \text{Figure 7.1: Service discovery framework architecture proposed in Chapter 4. In this chapter we present our DNS-SD/sDNS component.} \]
expressing popular demand for providing a DNS-SD/mDNS multi-link extension had been published.

This aligns well with our research and especially with this chapter, where we propose DNS-SD over Stateless DNS (DNS-SD/sDNS) facilitating versatile configurationless respectively low configuration modes of DNS-SD operation for multi-link networks. Our Stateless DNS technique (see Chapter 8) allows registrationless provision of DNS resource records via existing DNS cache servers. This technique allows discovering the service directory which is distributed among few hosts within the institution’s network, by providing NS resource records that delegate a special service discovery domain to these hosts. For synchronizing the service directory among these hosts, we use the Raft consensus algorithm [OO14].

Using the existing DNS infrastructure, the only additional entity needed is a lightweight reflector — implementable e.g. in a few lines of Perl — that can run within or (publicly) outside the current institution’s network. The reflector we use in this chapter offers the subset of Stateless DNS that is necessary for DNS-SD/sDNS; it offers a subset of the full Stateless DNS echo server interface proposed in Chapter 8.

The contribution of this chapter is a new way of DNS-SD resource record distribution that

• is user-friendly as it offers a configurationless mode of operation in multi-link networks,
• is versatile as it crosses multicast domains and adapts to arbitrary scopes,
• is efficient as it does not depend on multicast and poses negligible overhead on clients, and
• seamlessly integrates into the DNS query process using well established techniques making it backwards compatible.

Figure 7.1 shows the bigger picture integrating DNS-SD/sDNS into our service discovery framework.

7.2 Related Work

Much research has been done in the field of service discovery, especially for ad hoc networks [VP08]. In this chapter we focus on related work in the field of DNS based service discovery, because these approaches work with existing infrastructure and are either backwards compatible to standard solutions or can at least be easily deployed.

7.2.1 Scalable Multicast DNS-SD in Low-Power Networks

Since multicast causes significant network load in wireless networks [HSS09], techniques that make DNS-SD/mDNS more scalable — especially in 6LoWPAN networks [MKHC07] — have been developed.

Klauck et al. present and analyze their DNS-SD/mDNS implementation for low-power devices in [KK12] and propose methods to compress DNS messages to make DNS-SD/mDNS more suitable for 6LoWPANs in [KK13].

EADP [DRAW14] is a protocol for scalable service distribution in 6LoWPANs; it has been leveraged as means of distribution for DNS-SD in [DR14] (DNS-SD/EADP).

Since these solutions are designed for 6LoWPAN networks and also depend on multicast, they are not applicable to larger multi-link home or institutional networks. But they can
be incorporated in a DNS-SD framework offering appropriate means of resource record
distribution depending on the network and on the capabilities of the client.

7.2.2 Centralized DNS Related Service Discovery Solutions

There are also DNS related service discovery methods, SkyDNS\textsuperscript{1} and Consul\textsuperscript{2}, using
multiple central directory servers. For our use-case, they are not suitable because like
DNS-SD/DNS they demand setup and maintenance. Nevertheless, they might be the
solution of choice for an institution willing to maintain the service directory, because they
offer a configurationless experience for the user, if the directory information is propagated
using DHCP.

SkyDNS uses etcd\textsuperscript{3} as back end, which is a distributed key-value store also leveraging
Raft [OO14] to maintain consistency. Services are announced by sending the service
information with JSON over HTTP to the underlying etcd. Thereafter, these service
instances are retrievable using standard DNS queries.

Consul uses another synchronization algorithm and is also suitable for distributed
computing centers.

7.3 Requirements

RFC 7558 [LCBM15] defines requirements that should be met by a scalable service
discovery solution. It summarizes the requirements, desiring “[...] a mechanism [...] that
populates the DNS name space with the appropriate DNS-SD records with less manual
administration than is typically needed for a conventional unicast DNS server”. Our
solution (DNS-SD/sDNS) offers precisely that.

In this chapter we focus on the requirements for multi-link home and institutional
networks. For low-power and lossy networks, we propose to use the solutions mentioned in
Subsection 7.2.1. For single-link home networks the widely used DNS-SD/mDNS is well
suited, because it poses no significant impact on the network load [Rai15]. DNS-SD/DNS
or solutions presented in Subsection 7.2.2 are suitable for global scope service discovery.
Our service discovery framework described in Chapter 4 covers further requirements stated
in RFC 7558 and also protects the user’s privacy.

7.4 DNS-SD over Stateless DNS

To resolve services in multi-link home or institutional networks (where multicast does not
work across links) we introduce Scope Name Servers (SNS) that act as service directories.
Any host in the network can become an SNS following the rules explained later in this
section. A group of SNSes is responsible for one service discovery scope that in turn
is defined by a service discovery domain, e.g. ml.ssdisc.com\textsuperscript{4} or floor3.buildingB.ssdisc.com. We integrate the service querying process seamlessly into DNS such that

\textsuperscript{1}https://github.com/skynetservices/skydns
\textsuperscript{2}https://www.consul.io
\textsuperscript{3}https://github.com/coreos/etcd
\textsuperscript{4}ml → multi-link; ssdisc → scoped service discovery. The service discovery domain ssdisc.com is
operational for experiments.
the query process is exactly the same as when using DNS-SD/DNS. This allows client software to be independent of the resource record transmission mechanism.

A very important design goal is to allow a configurationless mode of operation for both clients and SNSes, including the process of becoming SNS. Further, the administrator of the respective multi-link network neither needs to deploy anything nor to be aware of our service discovery method.

The questions how to discover SNSes, how to become SNS, how to synchronize a set of SNSes, how to query, and how to register service instances are addressed in the following subsections.

### 7.4.1 Service Discovery Domains

We propose three options\(^5\) for clients to learn about service discovery domains: preset discovery domains, DNS-SD/DNS, and DHCP. The default service discovery domain can be preset, e.g. to `ssdisc.com`, so that clients work without any additional configuration. Further service discovery domains can be provided via DNS-SD/DNS, e.g. on `dom.ssdisc.com`. Institutions could also provide their own reflector, e.g. using one of our implementations, and distribute the corresponding service discovery domain via DHCP. This would only pose a minor configuration overhead for an institution compared to maintaining a centralized service directory, because only a lightweight reflector and a

\(^5\)The DNS interface for querying an SNS is independent of the method used to learn about the discovery domain.
one-time DHCP entry are required. Even if it is possible to provide an institution specific reflector, in most cases it is not necessary. Reasons for providing a reflector might be high privacy\(^6\) and availability requirements.

The combination of a service discovery domain and a DNS cache defines a service discovery scope; thus even when using a public reflector (and service discovery domain), service directory information is only available to hosts accessing the same DNS cache server. A user can create named sub-scopes independent of the underlying network structure by establishing itself as SNS for an arbitrary subdomain of a service discovery domain.

### 7.4.2 Providing the NS Records

Since an SNS might leave the network at any time, a very dynamic means of providing SNS information is required. Further, SNSes should only be discovered within the institution they currently sojourn in. To this end, we store name server delegations to SNSes in the current network’s DNS cache using our Stateless DNS technique\(^7\). This allows having location dependent SNSes for a single service discovery domain; depending on the network a host is currently discovering in, different resource records can be retrieved using the same global service discovery domain.

To make Stateless DNS work, the only additional infrastructure we need is a stateless reflector implemented e.g. in a few lines of Perl. It acts as authoritative name server for the parent zone of the service discovery domain.

Figure 7.2 shows the process of leveraging Stateless DNS to insert an NS resource record into the current DNS cache. Alice sends a programming query — which is a valid DNS query — by asking for an NS resource record with the name

\[
\text{C000020A.t100.ml.ssdisc.com IN NS}
\]

that will be handled by the local DNS server. Because the stateless reflector is authoritative for this query, it will receive the query from the local DNS server. The reflector then generates the following response using only information encoded in this query.\(^8\)

\[
\begin{align*}
\text{Question: } & \text{C000020A.t100.ml.ssdisc.com IN NS} \\
\text{Authority: } & \text{ml.ssdisc.com 100 IN NS ns1.ml.ssdisc.com} \\
\text{Additional: } & \text{ns1.ml.ssdisc.com 100 IN A 192.0.2.10}
\end{align*}
\]

Since it is a delegation to an in-Bailiwick [SS10] name server the cache will accept the answer if this in-Bailiwick name server — Alice’s notebook with the IP address 192.0.2.10 — is able to answer the programming query and NS queries for the service discovery domain.

\[
\begin{align*}
\text{Question: } & <\text{LABELS.}>ml.ssdisc.com IN NS \\
\text{Authority: } & \text{ml.ssdisc.com 100 IN NS ns1.<LABELS.>ml.ssdisc.com} \\
\text{Additional: } & \text{ns1.<LABELS.>ml.ssdisc.com 100 IN A 192.0.2.10}
\end{align*}
\]

The programming query answer’s sole purpose is to make the cache name server store the NS entries. The answer for NS queries of the service discovery domain is needed to be able to retrieve the SNSes’ IP addresses. The general form of the programming query is\(^9\)

---

\(^6\)When using a public reflector, the only transmitted data beyond the institutions network are scope names and the (local) IP addresses of the SNSes.

\(^7\)In Chapter 8 we also propose methods to store other record types and evaluate the proposed methods. The method used in this chapter (to store NS records) works reliably as it behaves like a normal sub-zone delegation.

\(^8\)The IP address of the new SNS is transmitted in hexadecimal notation.

\(^9\)We use () for grouping and {} for repetition count.
allowing specifying up to four name servers in a single programming query. All specified
name servers have to be able to answer the programming query and the NS queries for the
service discovery domain. Each of them has to return all NS entries\textsuperscript{10} to make the cache
name server ask the next available name server in the case the first one asked is offline.

7.4.3 Establishing SNSes

When entering a network, a host supporting our multi-link DNS-SD technique sends a
standard DNS query asking for NS records belonging to the service discovery domain
to get a list of SNSes (see Figure 7.3). The query corresponding to our example service
discovery domain is

\texttt{ml.ssdisc.com IN NS}.

In the bootstrap phase there will be no SNS and the host will become the first SNS using
the Stateless DNS method explained above (Figure 7.2). Because these NS entries cannot
be overwritten, the TTL should be chosen adaptively. The first SNS should choose a low
TTL when establishing its first NS entry, e.g. 10 seconds. To mitigate race conditions
arising when other SNSes exist whose TTL just expired in the moment the new host asks
to become name server, the new host must ask a second time after a random back off.

When the host gets a list of SNSes, it asks one of them whether it should also join the
set of SNSes and, if requested, joins the set. This concludes discovering the current SNSes
and the host can now ask for service information or register service instances as described
in Subsection 7.4.7. Figure 7.3 illustrates the process of SNS discovery.

It might happen that none of the returned SNSes answers. This is the case when all of
them have gone offline without the TTL expiring and with no new SNSes joining. Since
Stateless DNS cannot overwrite existing NS records, the discovery domain is blocked in
this situation. For this reason, we need a defined fallback discovery domain and means to
restore the standard discovery domain as soon as possible. The fallback discovery domain
can be derived from the standard discovery domain by appending $\cdot 0$ to the highest scope
defining label; thus the fallback domains for \texttt{ml.ssdisc.com} and \texttt{floor3.buildingB.ssdisc.com}
are \texttt{ml-0.ssdisc.com} and \texttt{floor3.buildingB-0.ssdisc.com}, respectively.
With increasingly low probability, the fallback domain might be blocked in the same way.
The fallback of the fallback is defined to be the domain with the corresponding integer
incremented by 1. When a discovery domain is blocked in this way, a host has to check
whether the next fallback domain has an SNS that answers and if not, become SNS for
this fallback domain (see Figure 7.3). To recover from this situation as soon as possible, a
host that has to become name server for a fallback domain sets the TTL of its \texttt{NS}
resource records to the remaining TTL of the NS records belonging to the standard domain.
This makes the standard discovery domain’s TTL and all fallback domains’ TTLs expire at the
same time and the host can register for the standard domain with all fallback domains
being unblocked. Because of the adaptive TTL, the chain of backup domains is expected
to be small. Further this problem will only occur in the bootstrap phase because the
system will stabilize as described later.

\textsuperscript{10}Since the SNSes know each other, this does not pose a problem.
7.4.4 Synchronization of SNS data

To make the system robust, we need several SNSes for each scope. These SNSes have to synchronize so that clients receive consistent information. Further, we want a querying client to be able to abstract away from the existence of several SNSes that need to synchronize, considering each of them as an equal representative of a black box providing the desired information.

For synchronizing the service directory among the SNSes, we use Raft [OO14], a simple and efficient consensus protocol, that uses heartbeat messages and randomized timers to elect a leader whose state is replicated on the other members of the consensus group called followers.

The leader accepts the clients’ write-requests, appends them to its log, and during each heartbeat phase, sends a message containing log changes to each follower. Each follower replies with a confirmation to the leader. The leader applies a log entry to its state as soon as it reaches consensus, meaning it is confirmed by the majority of the followers. The followers apply these log entries to their log in the next heartbeat phase. If the leader fails to send a message within the heartbeat before a follower’s timer runs out, the follower becomes a candidate and sends a heartbeat message to each member, asking to vote it as the new leader. Members that did not get any other heartbeat message within the current interval will send an answer message voting for the candidate to become the new leader. If the candidate gets consensus it becomes the new leader.

After each heartbeat there is consensus\footnote{There is a negligible possibility for split votes. If a split vote occurs, consensus is very likely to be reached during the next heartbeat interval.} about the leader and the current state. Raft is very efficient with respect to network load as it only uses $2k$ messages per heartbeat, where $k$ is the size of the consensus group. All information needed to agree on log changes, leader changes, and membership changes is communicated in these heartbeat messages.
Mapping Raft terms to DNS-SD/sDNS, the consensus group corresponds to the group of SNSes, the state to the service directory, clients to non-SNS hosts, and write-queries to service instance publish requests.

**Dynamic Membership Changes.** To make Raft applicable for SNS synchronization we adapted Raft allowing dynamic membership changes. Since one of our main goals is a configurationless mode of operation, we need means to add a new member to the consensus group without manual configuration. A host that wants to become a new consensus group member has to send a join query to the current leader which transmits this information to all other members. For this log entry, consensus does not suffice; all members have to confirm it. To this end, it is necessary for the leader to be able to remove not responding followers from the consensus group. When a removed follower gets active again, it has to join as a new member. This also allows us to truncate the log. A new member gets the state from an arbitrary follower and the last log entries necessary for the current state from the leader. We will thoroughly describe and evaluate dynamic membership changes in the future.

### 7.4.5 Deciding on a new SNS

The number of SNSes for a scope should be chosen dependent on the number of hosts, the number of offered service instances, and the churn rate. The decision whether a querying client should become SNS should be based on a ranking taking the host’s expected time to stay online into consideration. As of yet, we are still assessing which kind of ranking to use. If there is only one or two other SNSes, the new client should definitely be chosen. The closer the number of SNSes gets to the desired SNS set size, the higher the rank of the new client should be in order to be accepted. There should still be the possibility to substitute the new client for an SNS in a full SNS set if the rank is appropriate.

### 7.4.6 Updating the NS Records

When the TTL of the current NS entry in the DNS cache runs out, the current leader has to reestablish itself and its followers as name servers. Since the number of name servers that can be provided via the DNS cache is limited by the reflector implementation — 4 in our current implementation — the leader chooses the followers with the highest rankings; we call an SNS that is established as name server *listed SNS*. The TTL grows with the average ranking scores of the current SNSes, but should not exceed a sensible limit. Since the current leader will not change as long as it is online, the system will stabilize. With increasing online time, hosts are more likely to be leader.

In very large scopes there might be a significant load on the listed SNSes as they are queried by the hosts (see Section 7.6). To cope with this problem, listed SNSes can relay queries to randomly chosen non-listed SNSes.

### 7.4.7 Querying the SNSes

While requests for publishing a new service instance can only be handled by the SNS leader, DNS queries can be handled by all SNSes. SNSes offer the standard DNS-SD/DNS interface to clients which allows asking for

- a listing of all existing service types,
Chapter 7. Scalable Scopes for DNS-SD

- a listing of service instances of a service type and
- the resolution of a certain service instance.

The process of querying is independent of the structure the SNSes are organized in. A client can query any of the SNSes retrieved from the DNS cache. Clients that need an up-to-date list of instances of a certain service can request DNS push \[PC15\] from the SNSes; this is important e.g. to provide an up-to-date list of online contacts in a chat application.

Resource records provided by SNSes have a TTL of 10 minutes, which seems to be a good compromise between avoiding stale information and efficiency. When hosts leave the network gracefully, they can send a sign-off message to an SNS allowing the SNS deleting all corresponding resource records and pushing this information to affected hosts. To make a single message containing a hostname sufficient, both hosts and SNSes store a mapping from host to resource records offered by this host.

### 7.4.8 Hierarchical SNSes

SNSes can delegate sub zones; e.g. to create sub-scopes or to delegate the resolution of certain service types. This could e.g. be used for load balancing or hiding the existence of certain sub-scopes.

### 7.4.9 Security and Privacy Considerations

Like DNS-SD/mDNS \[CK13a, CK13b\], our technique currently relies on the fairness of the participating hosts. By itself, neither technique offers privacy, and has the unmitigated risk of malicious modification of resource records. Privacy — and integrity for private resource records — can be added to either technique using our orthogonal privacy extension (see Chapter 6) that provides means for secure privacy-preserving service discovery among hosts sharing a previously exchanged secret.

Using DNS-SD/mDNS without the privacy extension, even passive hosts receive all resource records related to service instances as soon as they are requested or offered by anyone in the network (see Chapter 3). Each user can overwrite existing service instances by violating the protocol. Since every host gets these malicious resource records, such a violation can be detected; mitigating techniques currently do not exist.

Using public DNS-SD/sDNS, hosts have to actively ask an SNS for resource records. An SNS could use filters to only provide selected hosts with the requested records. Only hosts that are currently in the SNS-role are allowed to overwrite existing service instances. Furthermore, SNSes tend to be more trustworthy than regular nodes because they are expected to be stable nodes that are part of a network for a long time. However, malicious SNSes can silently overwrite or drop service instances. We work on mitigating this problem.

### 7.5 Implementation

We have several reliable implementations of the reflector (Perl, C, Java), which are ready for deployment. Figure 7.4 shows the small but operational Perl implementation of the reflector that is authoritative for our example domain ssdisc.com; it supports the name server
Figure 7.4: Stateless reflector implementation in Perl. This implementation runs as authoritative name server for our example domain ssdisc.com.
delegation method used for DNS-SD/sDNS. Our extensible C implementation\textsuperscript{12} of the full Stateless DNS echo server supports all Stateless DNS methods described in Chapter 8 and can be easily augmented to support new methods by providing a corresponding template file.

Our prototypical implementation of an SNS-capable service discovery daemon\textsuperscript{13} currently uses a single SNS per scope; we are in the process of implementing SNS synchronization. Our proof-of-concept implementation\textsuperscript{14} — realized as an extension to the Avahi Zeroconf daemon — already allows using DNS-SD in our campus WLAN where multicast is disabled.

### 7.6 Analysis

In order to get widely accepted, user-friendliness with respect to configuration effort is not sufficient; the solution also has to be efficient. For service discovery this means its working should be imperceptible to users with respect to both network load and computational overhead. Despite the fact that as of yet we did not thoroughly evaluate the network impact of our solution,\textsuperscript{15} the following analysis suffices to point out the user-friendliness, suitability and scalability of our solution.

We analyze the influence of our solution on all relevant entities, namely hosts, SNSes, the SNS-Leader, the Reflector, and the DNS cache. An SNS is also a host, meaning it has to perform host actions and specific SNS actions; the SNS leader also performs SNS and host actions in addition to SNS leader specific actions. We group the actions into the following categories

- Raft related,
- messages to and from the reflector,
- messages to and from the DNS cache, and
- host to SNS communication which consists of (1) service type listing, (2) service instance listing, (3) service instance resolution, (4) service registration and (5) deregistration, and (6) queries about joining the SNS cluster.

The unit of our analysis is a single scope because the actions causing the highest impact on the network load — service instance listing and resolution — do not propagate beyond scope boundaries. For actions reaching beyond scope boundaries, e.g. communication with the reflector, we examine an appropriate wider area.

Before going into more detail, we want to shortly address the main efficiency concern, namely the number of service instances a host wants to be listed. With DNS-SD the host requests a service listing for service types it is interested in and then selects the service instances it wants to be resolved. In huge scopes this number can be quite large and the SNS has to send the full list of PTR resource records corresponding to these service instances to the host. This only has to be done once per host joining a network; but a large number of hosts joining a network might cause a significant load on the corresponding SNSes. However, since the user has to choose among the service instances manually, we argue that this number should not be large. In the future, we will consider attribute

\textsuperscript{12}https://gitlab.com/kaiserd/sdns
\textsuperscript{13}https://gitlab.com/kaiserd/sns
\textsuperscript{14}https://gitlab.com/holst/mlsd_avahi.git
\textsuperscript{15}We are going to evaluate our service discovery framework thoroughly leveraging the Omnet++ discrete event simulator (https://omnetpp.org).
based service instance selection on the SNSes to cope with this problem. Nevertheless, our solution in its current state scales very well as we show in this section. We meet the scale requirement of RFC 7558 [LCBM15] — “It must scale to a range of hundreds to thousands of DNS-SD/mDNS-enabled devices in a given environment.” — as shown in the following analysis. Standard multicast DNS service discovery does not scale to scopes that large because the mere number of multicasts would tie the network. We do not take background traffic into consideration and consider only nodes supporting DNS-SD/sDNS. For ease of calculation we assume that the devices register all their services right when they join the network.

Our analysis uses the following variables and their respective limits.

- \( n \), the number of hosts in the scope. To demonstrate the scalability, we use \( n = 10000 \), even if such big scopes are unfeasible as long as users have to choose manually among all service instances of a requested type. When using a query mechanism that preselects service instances, our solution can handle scopes beyond 10000 nodes as the limiting factor is the number of requested service instances.
- \( n_o \), number of nodes joining (and leaving) the network per second. We use 10 minutes as minimum for the average time a host is online [HSS09], leading to maximum arrival rate of \( n/600 = 16.6 \approx 17 \) users per second.
- \( s_o \), the average number of service instances offered by a host. We regard \( s_o = 5 \) as a sensible default.
- \( s_l \), the average number of service instances a host wants to be listed. In huge scopes, we consider 5% of the service instances offered a reasonable upper bound, leading to \( \max(s_l) = n_s_o * 0.05 = 2500 \).
- \( s_r \), the average number of service instances requested by a host. In huge scopes, we consider 5% of the listed service instances a reasonable upper bound, leading to \( \max(s_r) = s_l * 0.05 = 125 \).
- \( k \), the total number of SNSes in the network. We consider \( k = 20 \) a sensible maximum as it is very unlikely for 20 SNSes to fail at the same time.
- \( k_l \), the number of listed SNSes, i.e. the SNSes stored in the DNS cache. With our current implementation \( \max(k_l) = 4 \).
- \( h \), Raft heartbeats per seconds. We consider \( h = 3 \) sufficient to provide hosts with current information. Increasing the heartbeat frequency to e.g. \( h = 10 \) would increase the number of transmitted packets, but not the network load because the service related information that has to be synchronized per time interval does not change.
- \( \text{size}_p \), average size of a PTR resource record. We use 100 Bytes as upper bound.
- \( \text{size}_{ST} \), average size of SRV and TXT record. We use a single variable because these records are always transmitted together. The upper bound used in the following is 500 Bytes.\(^{16}\)
- \( \text{size}_{PST}, \text{size}_p + \text{size}_{ST} \), average size of all records associated with a service instance, summing up to an upper bound of 600 Bytes.
- \( \text{TTL} \), the TTL of entries in the DNS cache. As described above, the minimum is 10 s.

Figure 7.5 shows the estimated network load the different entities have to cope with, and how the network load on an SNS is distributed to the different SNS actions.

\(^{16}\)These average resource record sizes are very high; in all load critical situations, many records are transmitted in one packet, which reduces the header overhead significantly.
Figure 7.5: Estimated network load caused by DNS-SD/sDNS, dependent on the number of
hosts in the examined scope. These plots use the default parameters discussed in this section.
We chose 2% of the hosts as SNSes, with a minimum of 4 and a maximum of 20. In scopes with
less than 4 hosts, every host is an SNS.

7.6.1 Host

A non-SNS host is agnostic to the Raft related actions. Further, it does not communicate
with the reflector. The communication with the DNS cache to get the current SNS list is
negligible as it is a tiny fraction of the many DNS requests when surfing the web. Queries
concerning joining the Raft cluster, and registering and deregistering services can be
neglected because these actions only demand a few messages per session. Listing service
types is also imperceptible because even in a very large scope the number of offered types
is expected to be manageable and the set of available service types is expected to change
slowly.

After discovering SNSes, a host asks for a listing of service instances followed by a
resolution request for chosen services. Even with \( \text{max}(s_l) = 2500 \) and \( \text{max}(s_r) = 125 \),
the amount of data received would only be \( s_l \times \text{size}_p = 250 \text{kB} \) and \( s_r \times \text{size}_{\text{PST}} = 75 \text{kB} \),
respectively, without taking compression into consideration. The load on the host while
sojourning in a network is very low even when using the afore mentioned maximum values:
\( n_o s_o \times 0.05 \times \text{size}_p \approx 425 \text{B/s} \) and \( n_o s_o \times 0.0025 \times \text{size}_{\text{PST}} \approx 125 \text{B/s} \) for service listing and
resolving, respectively; the arrival rate of new users per second is about 17 with each of
them offering 5 service instances on average, of which in turn 5% have to be listed and
0.25% have to be resolved. Even in our large example scope the network load a host is
exposed to only sums up to a manageable burst of 325 kB when joining the network and
550 B/s while sojourning in the network.

Further, the computational overhead on the host devices used for packet processing is
imperceptible to the user, both in terms of responsiveness of the system and battery life.
The host has far less load compared to standard DNS-SD/mDNS as managing services is
the SNSes’ task; this makes our solution feasible for low power devices.

7.6.2 SNS

Raft handling only needs two messages per heartbeat (2h messages per second), and only
messages propagating new resource records might be of considerable size. Thus, the load
caused on an SNS by Raft corresponds approximately to the number of resource records
the hosts publish in the corresponding scope \( n_o s_o \times \text{size}_{\text{PST}} = 16.5 \times 5 \times 600 = 50 \text{kB/s} \).
Deregistering services is also handled in the heartbeat messages; it only needs a hostname per host signing off, adding just $n_s \times \text{size}_p = 16.5 \times 100 \approx 1.7 \text{kB/s}$.

We do not consider service type listing. Typically there are only a few different service types (roughly $ns_{s_t}/s_t = 20$ in our example scope) and since most hosts already know which types they are interested in, they do not need to list them.

The load caused by service instance listing and service instance resolution corresponds to the load on a host when joining the network multiplied by the arrival rate, divided by the number of SNSes $n_s/s_t/k \times \text{size}_p = 16.5 \times 2500/20 \times 100 \approx 210 \text{kB/s}$ and $n_s/s_t/k \times \text{size}_{pst} = 16.5 \times 125/20 \times 600 = 62.5 \text{kB/s}$ for service instance listing and service resolution, respectively.

Pushing service deregistration information to clients needs $n_s/s_r/k \times \text{size}_p = 16.5 \times 125/20 \times 100 \approx 10.5 \text{kB/s}$, in the unfavorable case that each service a host resolved was offered by a distinct host.

The memory capacity needed by an SNS to store the service directory amounts to $ns_s \times \text{size}_{pst} = 30 \text{MB}$. When an SNS goes offline and a new SNS is chosen, the service directory has to be synchronized to the new SNS. Since in small scopes the caused load is insignificant, in large scopes the SNSes are expected to stay online for a long time, and only one of the SNSes has to transmit the directory to the new node, we do not consider these occasional bursts in the average network load per second an SNS has to cope with. If every ten minutes one SNS goes offline — which is a high frequency of SNS change for such a large scope — an SNS has to cope with such a burst approximately once every 3 hours.

In summary, an SNS has to cope with a network load of 340 kB/s in our large example scope when estimating the neglected actions to amount to approximately 10 kB/s.

### 7.6.3 SNS Leader

The communication to the reflector is negligible; even using the minimal TTL it only happens once every 10 seconds.

The Raft message load the SNS leader is exposed to corresponds to $k$ times the load of the follower SNSes, caused by messages to all $k - 1$ followers, plus the hosts’ publish queries which loadwise roughly correspond to an additional follower $n_s/s_r/k \times \text{size}_{pst} = 16.5 \times 5 \times 20 \times 600 = 1 \text{MB/s}$. This is a significant load, but our example scope is very large (as well as the average resource records size) and the SNS leader in such a large scope is expected to be very strong and connected to Gigabit Ethernet. Since a scope of this size takes time to grow, there will be many leader elections, eventually resulting in a strong leader and also strong followers. To reduce the load on the SNS leader in very large scopes, it can delegate all DNS queries to other SNSes.

### 7.6.4 Reflector

The load on the reflector is really low. For each scope that is part of its authority zone it has to communicate once with a single SNS leader before the corresponding TTL is about to end. Even if a single service discovery domain had so many scopes that it would be hard for a single reflector to handle them, several reflectors using anycast could be used without synchronizing, because the reflector does not hold any state.
7.6.5 DNS Cache

The load increase for the DNS cache when using DNS-SD/sDNS is imperceptible as usual web surfing causes a myriad of DNS cache requests. There is only one new cache entry per SNS TTL for a scope and hosts only ask for the SNS list when entering the network or when the SNSes stop answering.

7.6.6 Effect of Varying the Parameters

So far, we focused our analysis on the afore discussed default parameters \( s_l = n s_o \times 0.05 \), \( s_r = s_l \times 0.05 \), \( s_o = 5 \), and \( n_o \approx 17 \). While we consider these to be sensible defaults, it is interesting to visualize and analyze the effect of varying these parameters. To this end, we provide plots showing various other parameter configurations (Figure 7.6 and Figure 7.7).

Host. The influence on non-SNS hosts caused by DNS-SD/sDNS is negligible. Since even increasing the load on non-SNS hosts tenfold would still result in a barely noticeable network load, the parameter variations shown in Figure 7.6 and Figure 7.7 are unrecognizable.

SNS. The parameters \( s_l \), \( s_r \) (see Figure 7.6), \( s_o \), and \( n_o \) (see Figure 7.7) all have a linear influence on the network load an SNS has to cope with. The number of service instances each host wants to list and resolve \( (s_l \) and \( s_r \)), corresponds to the number of listing and resolving queries the SNSes have to answer, whereupon each of these answers is of constant length. Each service instance a host in a scope offers (summing up to \( s_o \)) calls for one service registering query (and one deregistering query when the corresponding host leaves the network) on one of the SNSes, and one constant length data block in a heartbeat message for each SNS — either to inform the leader of the newly added service instance, or when receiving the corresponding log update request from the leader. Each host joining and leaving the scope (summing up to \( n_o \) per time unit) gives rise to \( s_l \) listing queries and \( s_r \) resolving queries. It further causes \( s_o \) service instance registrations.

Since both \( n_o \) and \( (\text{in our examples}) s_l \) are linearly dependent on \( n \), the network load on an SNS is a quadratic function of the number of hosts per scope \( (n) \). An \( s_r \) that is linearly dependent on \( n \) causes a very high network load and acts as a benchmark, but is not expected in a real-world network. Most hosts will just resolve a few (fixed number) of service instances; even for a chat service with a potentially huge number of participants, because the service resolution is only performed when a user wants to chat with a certain contact (lazy service resolution).

For all shown parameter variations, which also cover limits that are beyond what is expected in real-world scenarios, the network load on SNSes is acceptable for scopes with up to 5000 hosts. For a scope with 5000 hosts and \( s_o = 10 \) service instances offered by each host, choosing \( s_l \) and \( s_r \) as 1% and 10, respectively, corresponds to each host searching a list of 500 service instances and choosing 10 of them. This scenario, which we consider realistic, is shown in Figure 7.7(c). Scenarios, where the SNS might get in trouble, as shown e.g. in Figure 7.6(e), are unrealistic as they are impractical for users.

The right hand plot of a plot pair in Figure 7.6 and Figure 7.7 shows the network load on an SNS distributed to the different SNS actions, respectively. Like stated above choosing a large \( s_r \) is unrealistic as users are very unlikely to manually choose a large amount of service instances for resolution. Therefore, the scenarios where the service
Figure 7.6: Estimated network load caused by DNS-SD/sDNS, dependent on the number of hosts in the examined scope. Each pair of plots shows the effect of a specific configuration of $s_l$ and $s_r$ on the network load on different entities and SNS actions, respectively. The other parameters are chosen according to the afore mentioned default values. For all plots, we chose 2% of the hosts as SNSes, with a minimum of 4 and a maximum of 20. In scopes with less than 4 hosts, every host is an SNS.
Figure 7.7: Estimated network load caused by DNS-SD/sDNS, dependent on the number of hosts in the examined scope. The pairs of plots show the effect of specific configurations of \( s_o \) and \( n_o \) on the network load on different entities and SNS actions, respectively. The other parameters are chosen according to the afore mentioned default values. For all plots, we chose 2% of the hosts as SNSes, with a minimum of 4 and a maximum of 20. In scopes with less than 4 hosts, every host is an SNS.
resolution action has a significant influence can be seen as unrealistic. For large scopes in realistic scenarios the service listing action dominates the influence on the network load.

**SNS Leader.** In scopes of nearly realistic size, \( s_l \) is not the main concern for the SNS leader (Figure 7.6). Changing \( s_l \) and \( s_r \), the load on the SNS leader does not change significantly because the followers answer client queries.

The main load on the SNS leader is caused by the Raft messages needed to synchronize the log state. The network load caused by these messages mainly depends on the number of service instances registered per time unit, which in turn is determined by \( s_o \) and \( n_o \).

Big values for \( s_o \) and \( n_o \) render a problem for the SNS leader. As Figure 7.7(c) and Figure 7.6(e) show, the load on the SNS rises fast when choosing \( s_o = 10 \). Assuming an SNS leader with a good connection can handle 1 MB/s, it should be able to handle scopes with up to 5000 hosts. Since the influence of \( s_o \) is linear, 1000 hosts can be handled, when choosing \( s_o = 50 \). These are reasonable high values to consider our solution as scalable in realistic scenarios. Further, as we pointed out before, the resource record sizes are expected to be smaller in real-world scenarios. Nevertheless, we plan to reduce the load on the SNS leader in the future, as we also plan to provide more sophisticated means for service instance browsing.

### 7.6.7 Unicast vs. Multicast

Compared to DNS-SD/mDNS we reduce network load in most scenarios as we forgo multicast. The influence of many multicasts on the network load is especially severe in huge 802.11 wireless networks [80212], because multicasts are transmitted using a very low transmission rate so that older devices not supporting higher transmission rates can receive the multicasts as well [VTLAY14]. Hong et al. [HSS09] show that 13% of their campus network bandwidth is used by DNS-SD/mDNS. We compared unicast and multicast with respect to network utilization in Chapter 4.

There are other disadvantages of multicast in 802.11 wireless networks described in [VTLAY14], like handling host sleep mode, which further increases battery drain, because devices have to stay awake if multicast traffic is waiting to be sent by the access point.

### 7.7 Conclusion and Future Work

Multicast DNS Service Discovery over Stateless DNS (DNS-SD/sDNS) provides a versatile, convenient and easily deployable means of resource record distribution for scalable DNS Service Discovery. It offers a configurationless mode of operation and seamlessly integrates in the DNS discovery process, allowing core-independent, user-controllable device interaction in the edge of the Internet. Our proof-of-concept implementation — realized as an extension to the Avahi Zeroconf daemon — already allows using DNS-SD in our campus WLAN where multicast is disabled.

We plan to address further security and privacy problems arising when offering service information across links and in scalable scopes. Further, we will evaluate our scope extension with respect to network efficiency using the Omnet++ discrete event simulator\(^\text{17}\). We plan to integrate DNS-SD hybrid proxy \([\text{Che15}]\) capabilities in the SNSes as soon as the Internet draft becomes an RFC, which is likely to happen soon. This will allow

\(^{17}\text{https://omnetpp.org/}\)
hosts that are not aware of SNSes to use DNS-SD in multi-link networks providing a very elegant way of being backwards compatible.
Communication of small key-value pairs is the essence of many distributed systems operations, ranging from rendezvous and synchronization primitives over file and service discovery to document-centric networks, distributed hash tables, and distributed caches. These key-value operations are often implemented in centralized servers or — where applicable and supported — over multicast. Both mechanisms are inadequate for many applications. Centralized servers are potential single points of failure, limit scalability, and require specific configuration. IP Multicast would be ideal, as it also defines a set of reachability scopes. However, it is currently not generally available, which is unlikely to change. Furthermore, multicast is sometimes blocked for performance reasons, especially in large wireless LANs.

Using only ordinary DNS requests and responses together with a scalable stateless reflector, Stateless DNS enlists the help of unmodified DNS caches to provide application-independent, scoped key-value communications as a straightforward basis for higher-layer distributed primitives.
8.1 Introduction

Many distributed applications depend on efficient, scalable, distributed, and configurationless key-value look-up. On the one hand there are centralized and centrally controlled hierarchical systems (like DNS), offering efficient look-up but needing setup, maintenance and configuration overhead; on the other hand there are methods for distributed key-value look-up, e.g. P2P networks or multicast, offering infrastructure independence, but are either not efficient, difficult to bootstrap or not generally available. Table 8.1 compares these key-value storage architectures.

In this chapter we show how to flexibly apply a hierarchical key-value look-up system, the DNS, to unite advantages of centralized, hierarchical and distributed solutions. We propose Stateless DNS, a technique allowing using unmodified DNS caching servers for storing and exchanging key-value pairs, which

- allows key-value look-up in $O(1)$ overlay hops,
- reduces network bandwidth,
- can be used where multicast is disabled or unavailable,
- is convenient, allowing configurationless usage,
- is easy to deploy, as it uses existing infrastructure
- grants efficient and flexible overlay multicast, and
- features reachability scopes$^1$.

To store information in a DNS cache,$^2$ hosts send a special *programming query* to our *echo server*, which uses only the information contained in this query to generate a DNS response the DNS cache will store. The programming query tells the echo server which subdomain name (key) the cache should use to store the information (value), allowing other hosts using the same DNS cache to retrieve the data with a DNS query for this subdomain name.

Since all information needed to generate an appropriate answer is contained within the programming query, our echo server does not need any configuration files or state. This in turn allows for a simple and easy to maintain implementation, e.g. a few lines of Perl using the Perl DNS library$^3$, and thus makes redundant or local deployment very easy. The lack of state and configuration also makes the echo server independent of the DNS caches, which are oblivious to the special nature of echo servers, because they appear like typical authoritative DNS servers. Therefore it is *not* necessary to have an echo server running within an institution. Any echo server instance can be used as long as it supports our query format. Statelessness further makes anycast deployment$^4$ trivial, because there is nothing that has to be synchronized between the echo server instances involved. The centralized architecture aspect the echo server adds to key-value look-up is no stronger than the typical P2P network’s super peers or rendezvous server for NAT traversal.

The reachability scope of stored information can be chosen by selecting the caching DNS server(s) used. Example areas are

---

$^1$It allows restricting the network in which values can be stored and retrieved by using a DNS cache which is bound to the desired network.

$^2$This can be the cache of a router at home, an institution, a provider, any public DNS server, or several of the aforementioned caches at once.

$^3$[http://www.net-dns.org](http://www.net-dns.org)

$^4$e.g. for load balancing, improving fault tolerance
8.2 Related Work

<table>
<thead>
<tr>
<th></th>
<th>Centralized / Hierarchical / Federated</th>
<th>Distributed P2P overlay</th>
<th>Distributed multicast</th>
</tr>
</thead>
<tbody>
<tr>
<td>storage</td>
<td>on known reachable servers</td>
<td>on different peers with unstable network parameters</td>
<td></td>
</tr>
<tr>
<td>retrieval</td>
<td>ask server for key → value or delegation</td>
<td>ask closest neighbor → value or closer neighbor</td>
<td>multicast to all peers → value</td>
</tr>
<tr>
<td>linkage</td>
<td>none / delegation / configured interconnection</td>
<td>P2P overlay</td>
<td>multicast group</td>
</tr>
<tr>
<td>advantages</td>
<td>$O(1)$ retrieval convenient for client</td>
<td>convenient for operator, no central setup or maintenance</td>
<td>$O(1)$ retrieval configurationless for client and server no central setup or maintenance</td>
</tr>
<tr>
<td>disadvantages</td>
<td>setup and maintain infrastructure</td>
<td>bootstrap, overlay hops</td>
<td>insufficient availability, network load</td>
</tr>
</tbody>
</table>

Table 8.1: Comparison of Key-Value Storage Architectures

- local area networks, e.g. for device discovery,
- larger institutional networks, e.g. for using multicast applications even if access points are configured not to forward multicast,
- in the WAN of an ISP, e.g. to locate nodes which are likely to be reachable with high bandwidth and low delays,
- or in the hole Internet by storing the information in several public DNS servers at once, e.g. for P2P networks or overlay multicast.

The area can also be flexibly adapted by using the DNS servers delivered by DHCP. This is desirable for applications like service discovery, where a user wants to be able to retrieve data based on his current whereabouts. For other purposes sets of (public) DNS caches can be served by echo servers.

We evaluated our Stateless DNS technique using three different echo server implementations (Perl, Java and C) and checked how well our different methods of storing data in DNS caches work on different versions of popular DNS server implementations, to test the applicability of Stateless DNS in local and institutional networks; we also evaluated Stateless DNS on a huge list of public DNS caching servers\(^5\) to assess whether Stateless DNS is applicable for globally exchanging key-value pairs. Our evaluation shows that Stateless DNS works very well for both local and global networks.

8.2 Related Work

Stateless DNS is related to research on different usages of DNS caches and to P2P networks based on distributed hash tables; the former because the functioning of Stateless DNS is based on DNS caches, the latter because distributed hash tables also provide distributed key-value look-up.

\(^5\)http://public-dns.tk
8.2.1 DNS Caches

Various research has been done in the field of DNS caches, especially regarding attacks and their countermeasures. Less attention has been dedicated to unusual inoffensive and helpful usages of DNS caches, the area which Stateless DNS is a part of. To our knowledge, research has been done on storing information within responses from existing domains but not on storing arbitrary DNS records. Further research has been done on evaluating different properties of DNS caches.

DNS Cache Attacks and Countermeasures. DNS cache poisoning attacks are related to Stateless DNS because they also try to make DNS caches store resource records in a way not intended by DNS. But they are fundamentally different in that cache poisoning aims to store resource records the attacker is not authoritative for, which is — in contrast to Stateless DNS — malicious. The DNS poisoning attack was first described in [Bel95] and [Sch93]. Vixie [Vix95] elaborates on cache poisoning attacks with respect to Bind and introduces the Bailiwick and trust-level rules described in Section 8.3, which prevent basic cache poisoning attacks from working. RFC3833 [AA04] describes name chaining attacks, which are a subgroup of DNS cache poisoning attacks. They sneak resource records that have another DNS name as RDATA into the DNS cache. An example is establishing an authority server bad.com that, when asked for NS records, answers with an NS for good.com which is then stored in the resolver’s cache. These attacks cannot be mitigated by the label-matching rule described in Section 8.3. Kaminsky [Kam08] proposed a cache poisoning attack that works despite Bailiwick rules; but since it depends on the birthday attack, which alters packets sent by well behaving hosts, it is not applicable for Stateless DNS.

Son et al. [SS10] evaluate the aforementioned cache poisoning attacks and give an analysis of the caching rules of several common DNS server implementations. Their analysis was very helpful for developing Stateless DNS.

Another very interesting attack on the DNS cache is the Ghost Domain Name attack described in [JLL+12]. The goal of the attack is to maintain domain name mappings in resolving caches even after they are deleted by the parent zone. To store records for bad.com in the cache of a DNS resolver after the delegation to bad.com has been deleted from the com zone, the attacker asks the DNS cache for another of bad.com’s NS records, which is then stored with a new TTL. When the resolving cache now is asked for a record within the bad.com zone instead of returning NXDOMAIN it will return the answer it receives from the name server referred to by the newly stored NS. It is similar to Stateless DNS in that it stores information in the DNS cache that cannot be retrieved by asking the DNS server hierarchy but only by asking the cache. However, the Ghost Domains are stored in the cache while they still are part of the DNS hierarchy and only have to be maintained in the cache thereafter, whereas Stateless DNS has to struggle with storing the information in the cache, in the first place.

Inoffensive Usage. Other research has been done on using the DNS system in a legitimate way to store information in DNS caches.

EphPub [CDCFK11] uses public DNS caching resolvers to store ephemeral cryptographic keys. Each bit of the key is connected to a tuple of a DNS cache and a DNS record, so that a certain bit can be set by asking the corresponding DNS cache for the corresponding DNS record which will store the record in the cache. The key is transmitted
as a sequence of the aforementioned tuples. To retrieve the key bits, each of the key’s DNS caches is asked for the corresponding DNS records with the recursion bit cleared so that the cache only answers if the records are in its cache; when receiving an answer the corresponding bit is set to 1, otherwise to 0.

DNStamp [NHO14] uses tuples of DNS caching servers and DNS records to store information that is used to verify timestamps. The list of these tuples depends on the generation time, the period the timestamp should be valid, and a hash of the timestamped data, in such a way that the TTL of the records in the caches contain the information to verify the timestamp.

Both EphPub and DNStamp use queries for already existing domains. They do not need additional infrastructure — not even an echo server — but many public servers have to be queried to store a very small amount of data. Further, they both solve a specific problem, while Stateless DNS can be used for arbitrary applications that need key-value stores allowing zone restriction.

Storing data in routers of Named Data Networks is addressed in [ACGT14]. It is similar in that it stores data in caches in an unintended way but is limited to Named Data Networks and thus does not work in today’s network infrastructure.

**DNS Cache Studies.** Ager et al. [AMSU10] evaluate the caching behavior (among other properties) of the public DNS servers of Google and OpenDNS and compare them to local ISP’s DNS servers. They argue that DNS servers that are subject to load balancing should keep their caches coherent. This would also benefit Stateless DNS as it would make a single programming query sufficient.

### 8.2.2 P2P networks

Structured P2P networks, like chorde [SMK⁺01] or Kademlia [MM02], also provide distributed means for key-value look-up in form of a distributed hash table. But where communication of small key-value pairs is sufficient, e.g. for service discovery, Stateless DNS is more lightweight, because it does neither need a running client nor does it have to maintain open connections. Stateless DNS further has the advantage of inherent reachability scopes. P2P networks can be used to store larger chunks of data, but Stateless DNS is still applicable for this application domain to help construct and bootstrap P2P networks (see Subsection 8.6.2).

### 8.3 DNS Caching Rules

Since Stateless DNS has to heed the rules DNS servers use to cache incoming resource records, we will first summarize these rules. Those rules were established because otherwise it would be simple to abuse the DNS caches for attacks. The most prominent of these attacks is DNS Cache Poisoning [Bel95, Sch93], where a malicious DNS Server overrides the name server entries in DNS caches so that the malicious DNS server is asked instead of the authoritative one.

#### 8.3.1 Label-Matching Rule

The label-matching rule demands that the received answer is only accepted and cached if it matches the corresponding query. When asking a caching resolver for the A resource
record of bad.com, and the name server of bad.com returns an A record for good.com, this record should not be cached. Also, when asking for an A record for good.com, an A resource record for subdomain.good.com is not accepted. Further when asking for an A resource record, only A records, name chaining (like CNAME) or referrals are allowed, but not other data resource record types like TXT.

8.3.2 Bailiwick

The Bailiwick [SS10] of an authority DNS server is the zone which it is authoritative for, from the viewpoint of a recursive resolver that was referred to this authority server. An authority server does not know what its possible Bailiwicks are, because it does not know who refers to it. In most cases the Bailiwick is one of the zones defined in the name server’s zone file or a zone received via zone transfer. However, there could be lame delegations it does not know about.

The Bailiwick Rule ensures that resource records from the authority and additional sections are only cached if they are within the Bailiwick of the authority server that generated the answer. Because of this, authoritative servers

- can only delegate zones that are within their Bailiwick,
- cannot map a new IP address to an out-of-Bailiwick name server even if they delegate a zone within their Bailiwick to this name server.
- cannot sneak further A records in the additional section if they are out-of-Bailiwick.

When asking a caching resolver for e.g. the A resource record of bad.com and the answer is

```
Answer:   bad.com IN A 1.2.3.4
Authority: good.com IN NS ns.bad.com
Additional: ns.bad.com IN A 5.6.7.8
```

the label-matching rule is satisfied, because the query and answer labels match. But the referral is out-of-Bailiwick; in this case the caching resolver is allowed to store the A resource record for bad.com, but is not allowed to cache the authority and additional sections.

The Bailiwick rule also prevents the caching resolver from caching the additional section if the answer was

```
Answer:   bad.com IN A 1.2.3.4
Authority: bad.com IN NS ns.good.com
Additional: ns.good.com IN A 5.6.7.8
```

and from caching the mapping for good.com if the answer was

```
Answer:   bad.com IN A 1.2.3.4
Authority: bad.com IN NS ns.bad.com
Additional: ns.bad.com IN A 5.6.7.8
good.com IN A 4.5.6.7
```

Bailiwick rules are not specified in an RFC but RFC 5452 [HvM09] advises to only accept in-domain records among other tips to make DNS more secure.
8.3.3 Trust-Level

The trust-level rule [EB97] specifies seven classes of trustworthiness for resource records depending on

- which server the resource records come from (own zone file, zone transfer, authoritative server, non-authority server) and
- in which section they are (answer, authority, additional).

Resource records that already reside in the cache can only be replaced by records of the same or a higher trust-level.

8.3.4 Stricter Rules

Some DNS server implementations use more rigorous caching rules, which we describe in Section 8.7.

Jiang et al. [JLL+12] propose stricter versions of the Bailiwick and trust-level rules to prevent the Ghost Domain Attack. A recursive resolver should only accept a zone’s delegation data from the authoritative server of its parent zone, and updates for entries that reside in the cache should only be allowed if the new resource record is in a higher trust-level.

8.4 Functioning of Stateless DNS

Our Stateless DNS technique allows any local user to store DNS resource records in the cache of the local DNS server, thus allowing hosts who share the same caching DNS server to exchange key-value pairs. It works by sending a special programming query to the local DNS server. The authoritative server replying to this query is a special DNS server we call an echo server, which generates the answer solely from information contained in the query, without consulting a zone file or another database storing the contents of the zone. It is in a sense more stateless than other DNS servers, hence the name Stateless DNS.

The generic form of a programming query is \texttt{data.method.alias.domain}, where

- \texttt{data} is the value, i.e. the data we want to store,
- \texttt{method} is the name of a method and determines the kind of answer the echo server will generate,
- \texttt{alias} is the key, i.e. an alias which can later be used to retrieve the data, and
- \texttt{domain} is the authoritative domain of the echo server that will generate the answer.

Anyone using the same caching DNS server can later retrieve the stored data by querying for the alias, i.e. for \texttt{alias.domain}. In the following examples we use \texttt{st-echo.com} as \texttt{domain}; it is fully operational as our echo server\(^7\) is authoritative for this domain.

To make Stateless DNS work, we have to heed the rules described in the previous section; if we do not, DNS caches will simply drop the resource records generated by our echo server.

\[207\]
Figure 8.1: Using our Stateless DNS technique, the notebooks (1 and 5) in subnet 1 and subnet 2 can exchange arbitrary DNS records, without any configuration or changes to the existing network infrastructure. This could be used e.g. so they can find each other and start direct communication, which otherwise isn’t easily possible because they are in different subnets and thus in different multicast domains.

8.4.1 Basic Example

We assume the notebook in subnet 2 (Figure 8.1, 1) sends a programming query by asking for an A resource record with the owner name

192.0.2.10.A.mynotebook.st-echo.org

to the local DNS server (Figure 8.1, 2). Because our echo server is authoritative for this query, the local DNS server will send the query to this echo server (Figure 8.1, 3). The echo server then generates a response from the information encoded in the query. Because the query used the simple “A” method, the echo server will just answer with an A record:

mynotebook.st-echo.org. IN A 192.0.2.10

The local DNS server might now cache this A resource record, allowing e.g. the notebook in subnet 1 (Figure 8.1, 5) to retrieve this resource record by asking for an A record with the owner name mynotebook.st-echo.org (Figure 8.1, 4).

This is a simple example showing the basics of our Stateless DNS technique. Most DNS servers would not cache this A record, because it does not heed the label-matching rule. We will propose various Stateless DNS methods — among them methods that work on almost all DNS servers — in the following sections.

8.4.2 Method Descriptions

Like stated above, our goal is to make a caching DNS server store a resource record generated by our echo server. Because name server implementations use different caching
rules, we developed different methods the echo server uses to answer a query. In the
following sections we present these methods. We specify messages by defining the query
and the generated answer records.

Analogous to the BIND9 zone file syntax, we use @ as an abbreviation for the origin
(see Chapter 3), which, for Stateless DNS, is the domain of the echo server (in our example
it is st-echo.org). For some of the methods we also need a special echo domain, which is
00<.alias>.@. Queries to this domain will always be answered in the following way.

**Question:** <IP>.00<.alias>.@ A
**Answer:**
<IP>.00.@ IN A <IP>

Data before a further 00 will be ignored, such that * .00.data.00<.alias>.@ is equivalent
to data.00<.alias>.@; this is required for some methods. The alias is optional and will
be ignored; it is important for some methods to heed the Bailiwick rule.

For each of the methods, we will present the example of trying to store certain data
under the domain mya.st-echo.org in the cache of the local DNS server. We will present
the answer our echo server generates for the particular example programming query and —
for methods where it is not obvious — the query used to retrieve the data and the answer
to this retrieving query generated by the DNS cache server.

We also provide sequence diagrams (Figure 8.2, Figure 8.3, Figure 8.5, Figure 8.6, and
Figure 8.7) illustrating the messages between the participating entities. The sequence
diagrams do not show the class and TTL for brevity’s sake which, in our examples, are
always IN and 100, respectively. Further, a real-world DNS cache server could react
in another way. For the sake of explanation we assume the DNS cache works with
the explained Stateless DNS method to demonstrate the DNS cache’s answer to both
the programming and the retrieving query; in the corresponding subsections we explain
problems that could arise.

Without loss of generality, we store A records in the following examples; thus we
use a concrete IP address and the IP address place holder <IP> as data portion of
the programming query for the text examples and sequence diagrams, respectively. Our
methods also allow storing other resource record types. Storing a TXT record works in
the same way; we just have to ask for a TXT record in both the programming and the
retrieving query, and provide appropriately formatted data.

In Section 8.7 we show, for a huge number of public DNS servers, how well the different
methods perform.

### 8.4.3 A: Direct Storage

The simplest class of methods are methods that tell the echo server to directly return the
desired record. The A method simply returns the record we want the server to cache, even
though the server did not ask for this record.

**Question:** <IP>.a.alias.@ A
**Answer:**
alias.@ A <IP>

In most cases this method will not work, because the answer does not answer the question.
But since the answer is within the Bailiwick of the echo server, it is not strictly incorrect
to cache it, and some servers — such as the DNS server included with Windows Server
2012 — accept this simple method. It is not cache poisoning because the echo server returning the answer is authoritative for this record. Figure 8.2 illustrates the A method. For the example programming query

```
192.0.2.10.A.mya.st-echo.org A
```

our echo server will respond with

```
;; QUESTION SECTION:
;192.0.2.10.A.mya.st-echo.org. IN A
;; ANSWER SECTION:
mya.st-echo.org. 100 IN A 192.0.2.10.
```

For this simple method we also exemplify storing TXT and SRV records using appropriately formatted programming queries. A TXT record could be stored using the following query

```
key=value.A.mya.st-echo.org TXT,
```

which will make the echo server respond with

```
;; QUESTION SECTION:
;key=value.A.mya.st-echo.org. IN TXT
;; ANSWER SECTION:
mya.st-echo.org. 100 IN TXT "key=value".
```

For a SRV record, an example programming query is

```
host.name-22.A.mya.st-echo.org SRV,
```

which will make the echo server respond with

```
;; QUESTION SECTION:
;host.name-22.A.mya.st-echo.org. IN SRV
;; ANSWER SECTION:
mya.st-echo.org. 100 IN SRV 0 1 22 host.name.
```

---

8Using the syntax of the `dig` utility.
8.4 Functioning of Stateless DNS

Note that even for TXT and SRV records, the method name is still A in this case, because it only serves to select the direct Storage method. The record returned depends on the query type, which consequently has to be changed to TXT or SRV, respectively.

Since storing TXT and SRV records instead of A records works in the same way for all methods, we will not continue to mention this for the following methods.

8.4.4 C*: CNAME Aliasing

Many DNS caches do not accept the A method because the queried owner name does not match the owner name in the answer. To circumvent this problem, we use a CNAME.

CA

The CA method uses a CNAME that maps the name of the programming query (which was requested) to the alias under which we want to store the record, while also providing the data for this alias. Figure 8.3 illustrates the CA method.

The answer generated by this method heeds both the label-matching and Bailiwick rules.

Question: <IP>.ca.alias.@ A
Answer:
  <IP>.ca.alias.@ CNAME alias.@
  alias.@ A <IP>

Using the CA method, for the programming query

192.0.2.10.CA.my.a.st-echo.org A

our echo server will generate the following answer

```plaintext
;; QUESTION SECTION:
;192.0.2.10.CA.my.a.st-echo.org. IN A
;; ANSWER SECTION:
192.0.2.10.ca.my.a.st-echo.org. 100 IN CNAME my.a.st-echo.org.
my.a.st-echo.org. 100 IN A 192.0.2.10.
```
Since many DNS server implementations (see Section 8.7) cache the CNAME but do not trust the A record, our echo server has a problem: The caching server will ask for alias.@, the domain the CNAME points to, but the echo server cannot answer this query because it does not have any state.

**CU**

The CU method associates an arbitrary, non-resolvable CNAME with the alias. This can be used to efficiently store data, if the cache accepts this.

```
Question: data.cu.alias.@  A
Answer:
  data.cu.alias.@  CNAME data.xy.
```

The .xy country code TLD was deliberately chosen in the private-use X* range of the ISO 3166-1 alpha-2 codes to maximize the probability that it will continue to stay unassigned. This is explicitly not possible with a non-ccTLD top-level domain because of ICANN assigning generic TLDs. Also for the *U methods the query type used for programming is irrelevant because they do not return any non-CNAME records.

When confronted with the example query

```
stateless-dns.CU.mya.st-echo.org  A
```

our echo server will answer with a CNAME pointing nowhere:

```
;; QUESTION SECTION:
;stateless-dns.CU.mya.st-echo.org. IN A
;; ANSWER SECTION:
stateless-dns.CU.mya.st-echo.org. 100 IN CNAME stateless-dns.xy.
```

### 8.4.5 **CC*: CNAME Chaining

The CNAME chaining CCA method is accepted by many DNS caches (Section 8.7). It utilizes the fact that many servers cache the entire CNAME chain and only ask again for the last owner name which actually resolves to an A record.

**CCA**

The CCA method returns two CNAMEs (Figure 8.4); the second points to the echo domain 00.@ described above and contains enough information in its labels to fully describe the desired resource record. This way, the server can safely query again for the last CNAME’s target and get the correct data, even though no state is held. Using the CNAME chaining CCA method, the query and answer are as follows.

```
Question: <IP>.cca.alias.@  A
Answer:
  <IP>.cca.alias.@  CNAME alias.@
  alias.@  CNAME <IP>.00.@
  <IP>.00.@  A  <IP>
```
8.4 Functioning of Stateless DNS

Figure 8.4: CCA method programming query. The owner name of the first CNAME containing the data (1.2.3.4) points to the desired alias. The alias is the owner name of the second CNAME, whose target label again contains the data. When a DNS caching server asks for this name in a second query, the echo server will be able to serve the desired A record by parsing the data contained in these labels.

This way the echo server can answer if the caching name server asks for the second CNAME’s target. The CCA method works with many DNS caches (Section 8.7). Because they cache all CNAMEs in a chain, they will cache the first two CNAMEs, and just ask again for the target name the last CNAME points to, which can be resolved by the echo server. Figure 8.5 illustrates the CCA method.

Sending the example programming query

```
192.0.2.10.cca.mya.st-echo.org A
```

to our echo server, it will answer in the following way.

```
;; QUESTION SECTION:
;192.0.2.10.cca.mya.st-echo.org. IN A
;; ANSWER SECTION:
192.0.2.10.cca.mya.st-echo.org. 100 IN CNAME mya.st-echo.org.
mya.st-echo.org. 100 IN CNAME 192.0.2.10.00.st-echo.org.
192.0.2.10.00.st-echo.org. 100 IN A 192.0.2.10
```

The retrieving query will yield the CNAME pointing to the echo domain 00.@, as well as the A record it points to.

```
;; QUESTION SECTION:
;mya.st-echo.org. IN A
;; ANSWER SECTION:
mya.st-echo.org. 85 IN CNAME 192.0.2.10.00.st-echo.org.
192.0.2.10.00.st-echo.org. 85 IN A 192.0.2.10
```

**CCX**

This method is like the CCA method, but doesn’t include the A record, thereby forcing the server to make another request for it. This comes from the observation that most servers make that request anyway, so that omitting the data can reduce network traffic. It is not expected to influence cacheability of the records.

Query: `<IP>.ccx.alias.@ A`

Answer:

```
<IP>.ccx.alias.@ CNAME alias.@
alias.@ CNAME <IP>.00.@
```

213
CCU

The CCU method combines the CU method with CNAME chaining. It is intended as a way to efficiently store arbitrary data, as opposed to IP addresses.

Query: data.ccu.alias.@ A
Answer:
  data.ccu.alias.@ CNAME alias.@
alias.@ CNAME data.xy

To the programming query

stateless-dns.ccu.mya.st-echo.org. IN A

our echo server answers:

;; QUESTION SECTION:
;stateless-dns.ccu.mya.st-echo.org. IN A
;; ANSWER SECTION:
mya.st-echo.org. 100 IN CNAME stateless-dns.xy.

The retrieving query for alias.@ will give the following answer, efficiently containing only the CNAME which contains the requested data.

;; QUESTION SECTION:
;mya.st-echo.org. IN A
;; ANSWER SECTION:
mya.st-echo.org. 58 IN CNAME stateless-dns.xy.
8.4 Functioning of Stateless DNS

8.4.6 $\text{D}^*$: DNAME Aliasing

The class of DNAME methods uses DNAME records to map all subdomains of an alias to an immutable subdomain of the echo domain, utilizing the double-00 syntax 00.*,00.@ described above. Subdomains are then of the form *.00.data.00.@, so that the actual subdomain used for the retrieving query is ignored. These methods work very well on almost all DNS servers.

DA

The DA method establishes the desired alias as a DNAME to an immutable echo host under the echo domain, so that all subdomains should return the same data. We also pretend to follow the DNAME by including the nominal target host in the response, resulting in the following query answer pair.

Question: <IP>.da.alias.@ A
Answer:
alias.@ DNAME 00.<IP>.00.@
<IP>.da.00.<IP>.00.@ A <IP>

In this response, the owner names in the query and the answer match. Because the initial query is affected by the DNAME, the owner name in the query <IP>.da.alias.@ is mapped to <IP>.da.00.data.00.@. Since the next record is an A record for exactly this name, it is also very likely that both answers will be cached. In case the caching server asks again for the A record, the echo server will be able to answer, because the data is included in the question as usual.

Because DNAMEs only affect subdomains names, not the subdomain itself, the retrieving query does not ask for the alias, alias.@, but for an arbitrary subdomain name thereof, *.alias.@. Figure 8.6 illustrates the DA method.

The programming query
Chapter 8. Stateless DNS

192.0.2.10.da.myast-echo.org A

will make the echo server answer in the following way.

;; QUESTION SECTION:
192.0.2.10.da.myast-echo.org. IN A

;; ANSWER SECTION:
myast-echo.org. 100 IN DNAME 00.192.0.2.10.00.st-echo.org.
192.0.2.10.da.00.192.0.2.10.00.st-echo.org. 100 IN A 192.0.2.10

The retrieving query

anything.myast-echo.org A

might cause the following answer

;; QUESTION SECTION:
anything.myast-echo.org. IN A

;; ANSWER SECTION:
myast-echo.org. 54 IN DNAME 00.192.0.2.10.00.st-echo.org.
anything.myast-echo.org. 54 IN CNAME anything.00.192.0.2.10.00.st-echo.org.
anything.00.192.0.2.10.00.st-echo.org. 100 IN A 192.0.2.10

In this case the CNAME was created by the caching name server. It is not necessary, but shows how the DNAME affects the queried name. Like stated above, anything before the second 00 will be ignored. This is why

anything.00.192.0.2.10.00.st-echo.org

will always return 192.0.2.10.

DX

This method is identical to DA, but doesn’t include the nominal target in the response. This should force the server to make another request, asking for <IP>.DA.00.<IP>.00.@.

Question: <IP>.dx.alias.@ A
Answer:
alias.@ DNAME 00.<IP>.00.@

DU

Similar to the CU and CCU methods, the DU can be used to store data in an efficient way.

Question: data.du.alias.@ A
Answer:
alias.@ DNAME 00.data.xy.

The desired alias is established as a DNAME to a host that doesn’t resolve to anything, so that all subdomains should return a proper CNAME, but no queries should go to the echo domain. This CNAME contains the desired stored data. In this case, the 00 delimiter has to be interpreted by the retrieving host, because no echo server is available for the unresolvable domain.

Querying our echo server for

stateless-dns.du.myast-echo.org A

will yield

216
The retrieving query for

```
anything.mya.st-echo.org A
```

might yield the following result.

```
;; QUESTION SECTION:
;anything.mya.st-echo.org. IN A
;; ANSWER SECTION:
mya.st-echo.org. 35 IN DNAME 00.stateless-dns.xy.
anything.mya.st-echo.org. 35 IN CNAME anything.00.stateless-dns.xy.
```

Again the CNAME might be automatically created by the caching name server.

**Semi-Atomic Retrieve-or-Store**

One notable feature of the D* methods is that the retrieving query, `anything.alias.@`, is very similar in structure to the programming query `<IP>.da.alias.@`. In fact, blindly sending the programming query yields a very desirable result: If the alias already exists in the cache, the stored value is returned. If not, the value included with the blindly-sent programming query is stored in the cache.

Consider the programming query

```
192.0.2.10.da.mya.st-echo.org A
```

If the DNAME for `mya.st-echo.org` is not yet in the cache, the caching server will consult our echo server, which will return a request as described in Section 8.4.6. It also caches the following DNAME record:

```
mya.st-echo.org. 100 IN DNAME 00.192.0.2.10.00.st-echo.org.
```

Now suppose a different host decides to store the value 192.0.2.10 using the programming query

```
192.0.2.10.da.mya.st-echo.org A
```

Because there is already a cache entry for `mya.st-echo.org`, the caching server resolves the DNAME internally and returns

```
;; QUESTION SECTION:
;192.0.2.10.da.mya.st-echo.org. IN A
;; ANSWER SECTION:
mya.st-echo.org. 54 IN DNAME 00.192.0.2.10.00.st-echo.org.
192.0.2.10.da.mya.st-echo.org. 54 IN CNAME .
192.0.2.10.da.00.192.0.2.10.00.st-echo.org. 100 IN A 192.0.2.10
```

The last name, `192.0.2.10.da.00.192.0.2.10.00.st-echo.org`, is not in the cache, and will be sent to the echo server. Because it ignores everything before the left .00. delimiter, it will synthesize a record from the data portion of `192.0.2.10.00.st-echo.org`, returning an A record of `192.0.2.10` — that is, the result of the first programming query.
However, if the DNAME record had not been in the cache at the time of the second programming query, the caching DNS server would have found itself in the same situation as for the first programming query, and would have returned the data contained in the second programming query, an A record of 192.0.2.10, while also caching

mya.st-echo.org. 100 IN DNAME 00.192.0.2.10.00.st-echo.org.

That is, it stores the data provided in the second programming query if and only if there is not yet an entry for the alias mya. This update happens as atomically as the cache updates of the caching DNS server. While this does not necessarily mean that it really is atomic, it does appear suggestive for arbitration schemes or similar applications.

8.4.7 N*: Dynamic Delegation using NS Records

The class of N* methods uses NS records to delegate an alias. This leads to very powerful methods which aren’t limited to simple data storage. Also, because delegations are at the core of the domain name system, these are the most robust methods.

NA

This method establishes a delegation for alias.@, by delegating to a server in the echo domain.

Question: <IP>.na.alias.@
Authority:
  alias.@ NS <IP>.00.alias.@
Additional:
  <IP>.00.alias.@ A <IP>

Even if alias.@ is delegated to an IP address that does not provide name server services, some caches store the IP address and allow the IP address to be retrieved later on. Since the name server is within the .00. domain, our echo server can answer an A record query even if the additional section is discarded by the cache.

For the programming query

192.0.2.10.na.mya.st-echo.org A

our echo server will answer with the following record set.

;; QUESTION SECTION:
;192.0.2.10.na.mya.st-echo.org. IN A
;; AUTHORITY SECTION:
mya.st-echo.org. 100 IN NS 192.0.2.10.00.mya.st-echo.org.
;; ADDITIONAL SECTION:
192.0.2.10.00.mya.st-echo.org. 100 IN A 192.0.2.10

Since this is a delegation to an inoperative name server, DNS caches that ask the host (192.0.2.10) that is now authoritative for the alias domain, will not get an answer and thus are very unlikely to cache the NS record. If the data was cached, it can be retrieved by explicitly querying for the name server with

mya.st-echo.org NS .

The cache then returns
8.4 Functioning of Stateless DNS

The additional section may not be present, so that either local parsing of the echo hostname or another query would be required.

**NU**

This method allows using NS records for efficient data storage. It is similar to the other *U methods.

**Question:** data.nu.alias.@

**Answer:**

alias.@ NS data.xy.

An example programming query for the NU method looks as follows.

stateless-dns.nu.mya.st-echo.org. A

Our echo server responds to it with:

**Question:** stateless-dns.nu.mya.st-echo.org. A

**Answer:**

mya.st-echo.org NS

and will always only contain an NS record because the hostname is unresolvable:

**Question:** stateless-dns.nu.mya.st-echo.org. A

**Answer:**

mya.st-echo.org NS

**NR**

This method is like NA, but the returned NS record delegates to a (local) name server that is able to answer queries within its zone; this delegation heeds the bailiwick rule.

**Question:** <IP>.nr.alias.@

**Authority:**

alias.@ NS ns.alias.@

**Additional:**

ns.alias.@ A <IP>

The NR method is very powerful because it allows delegating a subdomain alias.@ to a DNS server running on any machine. The data portion of the programming query contains the IP address of the host that should become the name server for alias.@ and thus has to be able to answer queries for something.alias.@ in the following way:
Figure 8.7: NR Method
8.4 Functioning of Stateless DNS

Question: something.alias.@
Answer:
something.alias.@ A <some IP>
Authority:
  alias.@ NS ns.alias.@
Additional:
  ns.alias.@ A <IP>

It has to return itself as the authoritative server for alias.@, and has to return an IP address (<some IP>) for anything in the alias.@ zone.

For test purposes, we implemented another stateless server, that fulfills these properties and returns 1.2.3.4 as IP address for any name in the alias.@ zone. Figure 8.7 illustrates the NR method.

For the programming query

192.0.2.10.nr.mya.st-echo.org A

our echo server will answer in the following way.

;; QUESTION SECTION:
;192.0.2.10.nr.mya.st-echo.org. IN A
;; AUTHORITY SECTION:
mya.st-echo.org. 100 IN NS ns.mya.st-echo.org.
;; ADDITIONAL SECTION:
ns.mya.st-echo.org. 100 IN A 192.0.2.10

Assuming the host with the IP address 192.0.2.10 runs the minimal server described above, the retrieving query

something.mya.st-echo.org A

will retrieve the following.

;; QUESTION SECTION:
;something.mya.st-echo.org. IN A
;; ANSWER SECTION:
something.mya.st-echo.org. 100 IN A 1.2.3.4
;; AUTHORITY SECTION:
mya.st-echo.org. 84 IN NS ns.mya.st-echo.org.
;; ADDITIONAL SECTION:
ns.mya.st-echo.org. 84 IN A 192.0.2.10

If there is no additional section containing the host’s A resource record, this yields a name server which is in its own Bailiwick, but has no address provided. For the resolver, this is an irrecoverable situation, but may end up forcing it to use an older address. This is especially important when the resolver queries again for the AAAA record: We do not provide an answer, but the resolver already has the A record anyway. The echo server cannot repeat this A record in the additional section because it does not know about it.

NRR

The NRR method is very similar to the NR method but allows specifying up to four name server delegations. The IP addresses of the corresponding name servers are transmitted in hex-encoded form. For the programming query

C000020A.C000020B.C000020C.C000020D.nrr.st-echo.org. IN NS

221
our echo server will answer in the following way.

```
;; QUESTION SECTION:
;C000020A.C000020B.C000020C.C000020D.nrr.st-echo.org. IN NS

;; AUTHORITY SECTION:
st-echo.org. 100 IN NS ns1.mya.st-echo.org.
st-echo.org. 100 IN NS ns2.mya.st-echo.org.
st-echo.org. 100 IN NS ns3.mya.st-echo.org.
st-echo.org. 100 IN NS ns4.mya.st-echo.org.

;; ADDITIONAL SECTION:
ns1.mya.st-echo.org. 100 IN A 192.0.2.10
ns2.mya.st-echo.org. 100 IN A 192.0.2.11
ns3.mya.st-echo.org. 100 IN A 192.0.2.12
ns4.mya.st-echo.org. 100 IN A 192.0.2.13
```

Since it is a delegation to an in-Bailiwick [SS10] name server the cache will accept the answer if these in-Bailiwick name servers are able to answer the programming query and NS queries for the service discovery domain.

The programming query answer’s sole purpose is to make the cache name server store the NS entries. The general form of the programming query is

```
(<hexenc_IPaddr>.){1,4}nrr.@
```

allowing specifying up to four name servers in a single programming query. All specified name servers have to be able to answer the programming query and queries for the delegated domain, e.g. for mya.st-echo.org. Each of them has to return all NS entries to make the cache name server ask the next available name server in the case the first one asked is offline.

NAR

We further developed a method that allows updating NS entries. This method leverages DNS Update [VTRB97] and is proposed in [Str16].

## 8.5 Implementation

We have two deployable implementations of the stateless echo server written in Perl and C. The Perl implementation\(^\text{10}\) is a minimal implementation using the Perl DNS library\(^\text{11}\). It supports the aforementioned Stateless DNS methods.

Our flexible C Implementation\(^\text{12}\) — proposed in [Str16] — uses the ldns library\(^\text{13}\). It supports template files for method definition and thus does not only support the existing methods but can be easily extended. Further, it fully supports DNSSEC including negative answers. The C implementation acts as authority DNS server for our stateless echo domain st-echo.com.

\(^9\)We use () for grouping and {} for repetition count.

\(^{10}\)https://gitlab.com/kaiserd/minisdns

\(^{11}\)http://www.net-dns.org

\(^{12}\)https://gitlab.com/kaiserd/sdns

\(^{13}\)http://www.nlnetlabs.nl/projects/ldns
8.6 Applications

In general, all applications that would benefit from a key-value store that is already widely distributed and accessible by anyone without registration, could use Stateless DNS. It is important to remember that only hosts using the same DNS cache can access the same database. For some applications, this kind of locality might be an advantage. The main application area for which we are currently using Stateless DNS is service discovery; we will also shortly discuss other types of applications that benefit from Stateless DNS.

8.6.1 Service Discovery

We leveraged Stateless DNS to develop a versatile service discovery framework in Chapter 4 and as a concrete extension to DNS Service Discovery in Chapter 7. Since Stateless DNS offers a more generic interface, these solutions are not limited to DNS service discovery. Stateless DNS can be used as flexible, configurationless key-value store for any service discovery solution.

8.6.2 Further Applications

We are working on further applications that benefit from Stateless DNS. In this thesis we just want to briefly describe them.

P2P Networks

In Section 8.2 we proposed Stateless DNS as a lightweight alternative to P2P overlay networks in cases where the look-up of small key-value pairs is sufficient. Previously we compared Stateless DNS to existing P2P network structures with respect to providing flexible key-value look-up. Stateless DNS cannot only be used as replacement for existing P2P networks and distributed hash tables, but can also be used to help with existing P2P topologies supporting applications that need to store bigger chunks of data; thus Stateless DNS is also applicable as a building block for constructing P2P overlays, e.g.

- acting as an index structure for distributed hash tables,
- bootstrapping existing P2P topologies,
- helping with NAT traversal,
- re-joining of subnetworks that have become separated from each other.

Using the service discovery techniques described in Chapter 7, peers can rediscover each other using Stateless DNS even if their network parameters change; this can help to maintain a more robust topology.

Covert Communication

Cunche et al. [CKB14] propose a method for covert communication using Bit Torrent Trackers and distributed hash tables. Their method could also be adapted to using Stateless DNS.
DynDNS

DynDNS allows assigning a domain name to a host whose IP address is subject to change. When the IP address changes, DNS update [VTRB97] is used to update the corresponding resource records. The disadvantage is that DynDNS has to be configured and that DynDNS providers request money for their services. Stateless DNS could be used — e.g. in a Perl script — to add an alias under a publicly known Stateless DNS domain, e.g. \texttt{st-echo.org}. This alias is reestablished every time the TTL expires and as soon as the IP address changes the new IP address is used.

Stateless DNS on public DNS caching servers might not be as reliable as a DynDNS service; but it does not demand registering for a DynDNS service, is free of charge, and in cases where reachability is critical, a fix IP address is the better solution anyway. This way of using Stateless DNS can be seen as more dynamic way of dynamic DNS.

8.7 Evaluation

To evaluate Stateless DNS we primarily analyzed Bind9 as it is the most commonly used DNS server implementation\textsuperscript{14}. We further analyzed Unbound\textsuperscript{15} because it has very strict Bailiwick rules and thus we use it as a benchmark for very robust methods.

We also evaluated Stateless DNS with a huge list of public DNS Servers\textsuperscript{16} to get statistics of how well Stateless DNS works on a broad spectrum of DNS server implementations and configurations actually in service on the Internet.

8.7.1 Bind9

Bind implements all three caching rules described in Section 8.3. The label-matching rule prevents the simple A method from working. Despite the fact that the CA method does not violate the label-matching and Bailiwick rules, it does not work with Bind because Bind uses a stricter version of the label-matching rule, which demands to cache only the matching record itself, in this case the CNAME, but not the data resource record at the end of the chain. This causes the Bind resolver to ask the echo server for this CNAME which it has no answer for. But the label-matching rule of Bind in its default configuration allows storing chains of CNAMEs. This is why the CCA method works; the echo server can give an answer for the second CNAME. Son et al. [SS10] give a very helpful analysis of Bind. Adding to their work our analysis of Bind9 showed that its caching rules depend on whether dnssec-validation is activated. If it is, it just caches the first CNAME of a chain and asks again for this CNAME. It does not cache any further names the first CNAME points to. Because of this the CCA method should not work with Bind when DNSSEC validation is enabled.

The DNAME method heeds the strict label-matching and Bailiwick rules because the programming query is affected by the DNAME and thus remapped in such a way that the A record in the answer matches the query. Since the A resource record is within the echo domain containing its own data, it can be resolved by the echo server. This makes DNAME a very robust method that works even with the DNSSEC check enabled. But the DNAME is not updateable because a new programming query for the same DNAME will be affected

\textsuperscript{14}http://dns.measurement-factory.com
\textsuperscript{15}http://unbound.net
\textsuperscript{16}http://public-dns.tk
8.7 Evaluation

<table>
<thead>
<tr>
<th>Version</th>
<th>a</th>
<th>ca</th>
<th>cca</th>
<th>da</th>
<th>nr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bind 9 (default)</td>
<td>0%</td>
<td>0%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Bind 9 (dnssec-validation)</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Dnsmasq</td>
<td>0%</td>
<td>0%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Unbound</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 8.2: Percentage of DNS cache versions supporting Stateless DNS.

by the DNAME already residing in the cache and thus will be directly answered without consulting the echo server.

For records in the authority or additional section, Bind checks if the given NS record maps to a DNS server that is authoritative for the alias domain, which is the case for the NR method. Therefore, the NR method should work reliably.

We evaluated our methods with nine versions of Bind ranging from 9.2 to 9.10. Using the default configuration, the CCA, DA and NR work reliably (see Table 8.2). With DNSSEC validation activated, which is only possible in the newest six versions we tested, the CCA method does not work; the DA and NR methods still work reliably.

8.7.2 Unbound

The very strict caching rules of Unbound are based on the Internet draft [Wij09] that advises (among other tips) to use the cache with care. It advises to only cache the first CNAME of a chain and to only use the first DNAME of a chain. This prevents the CA and CCA methods from working. The Internet draft further advises to only use a DNAME from cache for queries for which that DNAME was returned previously. This allows our DNAME method to store the DNAME in the cache. But since our echo server never returned the DNAME in conjunction with the retrieving query, Unbound does not apply the DNAME returned by the programming query to the answer of any retrieving queries; thus it is impossible to retrieve the data from the Unbound cache, rendering the DNAME method unusable.

The results of our evaluation of several Unbound versions confirms the aforementioned theory (Table 8.2). While the NR method works reliably, all other methods do not work.

8.7.3 Dnsmasq

Dnsmasq\footnote{http://www.thekelles.org.uk/dnsmasq/doc.html} is widely used for local area network DNS caches like in home routers. Our evaluation shows that the methods that work with Bind also work with Dnsmasq.

8.7.4 Public DNS Servers

For the evaluation of public DNS Servers we use a test script that checks each of our methods on each public DNS server listed on \url{http://public-dns.tk}. The script tests if a server is reachable and takes load balancing with unsynchronized caches into consideration to reduce the number of false negatives.
Chapter 8. Stateless DNS

<table>
<thead>
<tr>
<th>Version</th>
<th>a</th>
<th>ca</th>
<th>cca</th>
<th>da</th>
<th>nr</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bind9</td>
<td>0%</td>
<td>8%</td>
<td>90%</td>
<td>99%</td>
<td>99%</td>
<td>798</td>
</tr>
<tr>
<td>Unbound</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>100%</td>
<td>9</td>
</tr>
<tr>
<td>MicrosoftDNS</td>
<td>81%</td>
<td>90%</td>
<td>93%</td>
<td>12%</td>
<td>100%</td>
<td>32</td>
</tr>
<tr>
<td>Overall</td>
<td>13%</td>
<td>31%</td>
<td>86%</td>
<td>66%</td>
<td>97%</td>
<td>1984</td>
</tr>
</tbody>
</table>

Table 8.3: Percentage of public DNS caches supporting Stateless DNS. Deviations from Table 8.2 arise due to connection problems, load balancing, and, foremost, non-default configurations.

Test Setup

After checking if the next public server to be tested is reachable and allows recursion, the script generates a pseudo-random alias that is used in the following test queries; the script then investigates for each method if the server caches the desired data by

- sending a programming query followed by
- sending the corresponding retrieving query.

If the response to the retrieving query contains the desired data, the currently tested method is marked as working on the currently tested server.

Since some applications also need to update a key-value pair, the script also evaluates whether an update is possible by

- sending a new programming query (updating query) using the same method and alias but different data followed by
- sending the corresponding retrieving query.

If the newly programmed data returns, the method is marked as updateable for the tested server.

Besides knowing if storing a record in the cache of a server works, it is also important to evaluate how long the record stays in the cache. Therefore, as soon as we get a positive result for a server, the script starts a retention test subprocess that probes in exponentially growing time intervals if the record is still cached.

If the one of the aforementioned retrieving queries does not yield the desired result, it does not necessarily mean the programming queries results were not stored. This is because some server groups, for example the Google DNS servers, are subject to load balancing. Therefore, it might happen that the programming query is handled by a different server than the retrieving query sent afterwards. In that case, the result is negative even if the record was actually cached. To handle load balancing, our script sends double the amount of programming queries if the last retrieving query failed until a certain threshold.

We continue to send programming queries even if the method was already marked as working to decrease the probability of some server being in the load balancing group which did not get a programming query yet and thus could falsely seem to be updateable. Only after that we start evaluating the updateabity of a method by sending batches of an exponentially increasing number of updating queries each followed by a single retrieving query.

With this approach of an exponential increase in the amount of programming queries, we preserve the quality of our results even in the case of load balancing, while also obtaining some information on the size of the load balancing groups.

18If the server does not fulfill these conditions, we remove it from the testbed.
8.8 Conclusion and Future Work

In some cases, if the number of servers in a load balancing group is much larger than the chosen maximum number of queries, it might still happen that we get a false negative result, but choosing a reasonable maximum number ensures that the number of such false negative results is very rare and does affect our overall results by only a negligible amount. Despite the initial test for reachability, some packets might be lost during the test, depending on network connection quality. To prevent a bad influence on our results, queries within the test are resent if they did result in a connection timeout. But still, we might get false negatives.

Results

The results (Table 8.3) show that the majority of public DNS caching resolvers support Stateless DNS, mainly because Stateless DNS works well with Bind and Bind is the most popular server among public DNS servers, making Stateless DNS suitable for global scale key-value look-up. Our retention test (Figure 8.8) indicates that in almost all cases the records stayed in the cache for at least ten minutes, which keeps refresh cycles at an acceptable rate, if needed at all. Since, despite our precautions, false negatives are still possible and since some caches behave strangely, e.g. timeout when confronted with an EDNS query, our results could be even better when taking only stable, well behaving caches into consideration, which would be the case when using Stateless DNS for a real application. Like stated above, synchronized caches on load balanced DNS caches would also benefit the stability of Stateless DNS.

8.8 Conclusion and Future Work

Stateless DNS is an easy to deploy and configurationless technique using unmodified caching DNS servers to provide arbitrary network applications with access to a flexible key-value store that features reachability scopes. In local and institutional networks it helps reducing the network load caused by multicast and allows using multicast functionality even if multicast has been disabled. It can also be used for overlay multicast at a global scale and helps with the construction and bootstrapping of P2P networks. As our evaluation showed, our technique works well with different DNS cache implementations and a huge number of public DNS caching resolvers.
In the future, we plan to elaborate on application areas besides service discovery. We would also like to further evaluate the benefits of Stateless DNS for different application domains with the help of OMNeT++\(^\text{19}\) simulations.

\(^{19}\)\url{http://www.omnetpp.org}
Configurationless service discovery makes network applications easier and more convenient to handle for both users and developers — and the framework proposed in this thesis makes it privacy-preserving, scaleable, efficient, and conveniently integratable in today’s network infrastructure to boot.

This closing chapter (1) provides a concluding summary (2) revisits our research questions stated in the Introduction, (3) and discusses future work.
9.1 Concluding Summary

In this thesis, we provided a privacy-preserving configurationless service discovery framework that solves both problems typical configurationless service discovery solutions suffer from:

- privacy, leaking of a lot of information, and
- locality, limiting connectivity and collaboration.

Using our solution, Alice neither tells other network participants about her presence, nor discloses any information about services she offers or uses. Only her friends may discover this information. When she intends to chat with her friend Bob, only Bob finds out about her wish to communicate with him. This does not limit Alice to only engaging in private data exchange with a specific friend; she can engage in service discovery with groups of friends and the public, if she wishes to do so, as well. Further Alice is not limited to a single network link. She does not have to care whether Bob is on the same link or not — she does not even have to understand the concept of network links — as the discovery just works.

We separated directory discovery and service exploration and realized them as two distinct layers. For achieving a configurationless mode of operation, hosts need a form of asking every available peer for information that allows deriving which peer is actually relevant. Asking every available peer both may breach privacy at a large scale and is inefficient. We use this way of asking for information only during directory discovery, which means once per joining a network (or when reconnecting). We protect the privacy during this public query process by only transmitting information which looks like randomly chosen data for unauthorized peers but serves as an identity hint for paired peers. Because the public query process takes place only once per joining a network and comprises data that both is minimal and does not breach privacy, it is much easier to allow for discovery in larger scopes. We leveraged the existing DNS infrastructure, more specifically the DNS caches, to store this information allowing all present peers to retrieve it, without the need for configuration.

9.2 Revisiting the Research Questions

Revisiting our research questions stated in Chapter 1, in this section we first summarize the answers to the component specific research questions drawing conclusions from designing and realizing our solutions. We then project these answers to our opening general research questions, closing the circle.

9.2.1 Device Pairing

How can two devices that do not have any common knowledge exchange an authenticated shared secret in a privacy-preserving way?

Our general approach answering this question comprises three phases.

Discovery: Devices discover each other to learn about each other’s network parameters associated with an unsecured high-bandwidth network. The bandwidth must be high enough for efficiently performing a key exchange protocol, e.g. a Diffie-Hellman
9.2 Revisiting the Research Questions

key exchange. Theoretically, any discovery method can be used to this end. The discovery process can, but does not have to, be performed via the high-bandwidth network. Protecting the privacy, unlinkable one-time identifiers, which are resolved via a user interaction, are the only information transmitted during this phase. We provided and discussed two concrete methods for realizing the discovery phase. The first method utilizes DNS-SD/mDNS to announce a pairing service whose instance name is a one-time identifier. The first device’s user communicates this one-time identifier to the second device’s user, who then chooses the identifier from a list of currently available pairing service instances. Resolving this service instance yields the desired network parameters. The second method uses QR codes for transmitting the network parameters; it is very convenient, but limited to devices featuring the necessary hardware.

Pairing Data Exchange: Knowing each other’s network parameters, devices can exchange a shared secret via a Diffie-Hellman key exchange. Two additions to the classic Diffie-Hellman key exchange allow proving the authenticity of the exchanged secret. Firstly, the initiating device transmits a cryptographic hash of its public Diffie-Hellman key. The other device replies with its public key; only upon receiving this message, the initiating device sends its public key. Secondly, a short authentication string (SAS) is calculated as a short hash of the established secret. This short hash is handed to the manual authentication phase. Privacy is protected during this phase, because only ephemeral public Diffie-Hellman keys and the hash of one these are transmitted. Neither can be linked to the devices engaged in pairing.

Manual Authentication: Both devices check whether they calculated the same SAS by involving their users. For instance, both devices display the SAS and ask their users to check whether the displayed SASes match. As no data is transmitted via the insecure network, privacy is not breached in this respect. However, users need to meet in person for authenticating the pairing, which can be observed. Manually authenticated pairing protocols, which we detailed in the background section of Chapter 5, comprise both pairing data exchange and manual authentication.

We seamlessly integrated these phases, added privacy protection, and designed a user-friendly discovery method, answering all aspects of our research question (see Chapter 5). We especially cared about user-friendliness.

Based on this general approach to the question, we discussed three concrete answers. All three can use either of our realizations for the discovery phase; they differ in the realization of the pairing data exchange and manual authentication phases.

Ad-Hoc Protocol: This realization directly uses the improved Hoepman protocol [NR11b], which is secure and efficient, but demands implementing crypto-functions.

Based on TLS Handshake: This realization leverages the TLS handshake [DR08] including the definitions of cryptographic primitives. It has the advantage of building on a well tested crypto implementation. It does not specify a new TLS extension, and thus is easy to deploy. We published an Internet draft specifying this solution [HK16a], which was adopted by the dnssd working group.

Integration into TLS: This realization integrates manually authenticated pairing into TLS specifying a TLS extension. We currently work on this solution and plan to provide an Internet draft in the future.

Further improving user-friendliness, we provide a pairing daemon managing device pairings across different applications. Besides secure pairing methods, the pairing daemon also
Chapter 9. Conclusion

supports a fully configurationless pairing method for trusted networks, e.g. for the home network. Our implementation of the pairing daemon is the practical answer to this research question. We detailed our pairing component in Chapter 5.

9.2.2 Directory Discovery

How can devices efficiently discover directory parts managed by peer devices in a privacy-preserving way without configuration?

Devices offer a private service directory service, which is discoverable in a configurationless privacy-preserving way. To allow for a configurationless operation, the fact that this service is available is made public, so that all peers may automatically browse the network listing available private service directory service instances. To protect privacy, the only data published is an ephemeral identifier that only paired peers can map to the service provider’s identity, and obfuscated network parameters, which are not linkable to the service provider. The ephemeral identifier is constructed as a hash of both the shared pairing secret and a timestamp that acts as a nonce. Hosts maintain a table of currently valid ephemeral identifiers, so that received discovery messages can be efficiently mapped to the corresponding service provider’s identity. Because all hosts learn about the availability of service instances, and paired peers are able to map the ephemeral identifiers to the corresponding service providers’ identities, if a pairing has been established, hosts can retrieve an up-to-date list of available private service directories without any user interaction. After learning the service provider’s identity, a host can use the shared secret established during pairing for initiating a private mutually authenticated session through which service exploration is performed. Unauthorized hosts may try to establish a connection to a private service directory service, but will fail without learning an identity. Besides this abstract answer, we also provided a concrete answer in form of a privacy extension for DNS-SD/mDNS. It uses DNS-SD/mDNS for announcing a private service directory service whose service instance name is the corresponding ephemeral identifier. While unpaired hosts are able to resolve this service, only paired hosts are able to connect to it and get information about it besides its ephemeral identifier, obfuscated network parameters, and an ephemeral randomized hostname of the offering device. MAC address randomization is leveraged to protect against tracking devices via link-layer identifiers. We detailed our privacy extension in Chapter 6. Further, we published an Internet draft specifying our privacy extension [HK16b], which was adopted by the dnssd working group.

How can this discovery process be realized even if multicast is not available, and how can it support discovery scopes with little to no configuration?

Since the way of announcing the private service directory service can be realized in various ways, it does not necessarily have to depend on multicast. Any configurationless technique that allows all peers of a certain scope to learn about service instances is sufficient. One technique that answers this question and seamlessly integrates into our DNS-SD/mDNS based realization is our Stateless DNS technique (Chapter 8).

Stateless DNS allows storing DNS resource records in the caches of recursive DNS servers. A host publishes resource records by sending a programming query — which is a
9.2 Revisiting the Research Questions

A standards-compliant DNS query — to a recursive server. The programming query is a query for a domain that is a subdomain of a service discovery domain for which a Stateless DNS echo server is authoritative. The subdomain’s labels under the service discovery domain contain the desired resource record’s data. Our echo server is the only component of Stateless DNS that needs to be available besides the standard DNS infrastructure. This, however, poses no configuration overhead as the mere existence of any echo server is sufficient. The echo server answers the programming query by parsing its labels and replaying the desired resource records, without holding any state. The recursive server stores the answer in its cache (if it meets the rules we detailed in Chapter 8). Peers in the same local network can retrieve these resource records querying the recursive server.

Answering the research question, a host can announce its private service directory service by publishing the corresponding PTR resource record via Stateless DNS. Since publishing NS records, which corresponds to typical DNS delegations, is more elegant, our solution proposes for hosts to publish an NS resource record delegating the authority for a subdomain of the service discovery domain to the host itself. This subdomain’s least significant label corresponds to the ephemeral identifier. This integrates seamlessly into the DNS query process because a query, e.g. for a PTR record, associated with this subdomain will be delegated to the offering host.

Scopes can be realized by the same mechanism. Each private service directory creates a private scope of its own. A host that wants to take responsibility for a public scope chooses the scope’s domain (a subdomain of the service discovery domain), and delegates this domain to itself acting as a Scope Name Server (SNS). If a host wishes to announce its private service directory in a scope already maintained by an SNS, it asks this SNS to provide the corresponding ephemeral identifier alongside its network parameters. We detailed multicast-independent service discovery and scopes in Chapter 7.

9.2.3 Service Exploration

How can a secure unicast connection for transmitting service related data be opened, given the network parameters obtained during the device discovery step, and can this connection be authenticated in such a way that both participants only disclose information about themselves if they both are who they claim to be?

The general answer to this question is given in the literature. We give a practical answer, that allows for an easy integration. The authenticated secret exchanged during pairing can be used for establishing a TLS connection in the Pre-Shared Key (PSK) mode of operation [Res16]. To protect the privacy, an ephemeral identifier has to be substituted for the PSK identity. We detailed service exploration in Chapter 6.

9.2.4 General Research Questions

We answered the first two of our general opening research questions by stating and answering the afore discussed specific research questions. The answer to the third question is intertwined with these answers; and the forth general research question is answered by the way of realizing of our designs.

1. How can configurationless service discovery be realized without breaching the users’ privacy?
This question is answered by combining the answers to our pairing, directory discovery, and service exploration questions. Our pairing component allows establishing a mutually authenticated shared secret (only once per pair of users). Directory discovery allows discovering relevant directory parts preserving the privacy by only publishing ephemeral identifiers which are generated involving the shared pairing secret. Service exploration allows retrieving desired service related information via a privacy-preserving connection leveraging both the network parameters retrieved during directory discovery and the shared secret established during pairing.

2. How can configurationless service discovery be realized for multi-link networks?

The answer to our specific question for a multicast-independent scope-supporting way of operation for directory discovery also answers this question, as being dependent on multicast is the very reason why traditional configurationless service discovery solutions do not support multi-link networks. Separating directory discovery and service exploration helped a lot in finding the answer as the service exploration phase is independent and is performed in a unicast session. Only the directory discovery stage has to overcome the hardships of not yet knowing whom to establish a session with, which reduces the complexity of the problem.

3. How can such a service discovery solution be designed to be efficient especially with respect to network bandwidth?

As previously stated, our solution sends service related information only to select paired peers, which not only helps preserving privacy, but also significantly reduces the network load. The basis for achieving the increase in efficiency is again dividing the discovery process in two stages. While the first stage has to be performed in the same inefficient way the whole discovery process is typically performed in, i.e. a way in which all present peers receive the discovery information, the second stage is performed through a direct connection. The directory discovery demands little additional network bandwidth which the standard solution does not need, but as our analysis in Chapter 6 showed, this pays off and our solution clearly outperforms the standard approach.\(^1\)

4. How can such a service discovery solution be designed to be easily deployable in today’s network infrastructure?

An easily deployable service discovery solution must neither demand any changes in existing protocols of the network stack, nor demand updates in well established infrastructure systems, nor pose a significant update overhead on users. Realizing our solutions as extensions to DNS-SD/mDNS, which just need to be installed once on client devices, answers this question. Since we published Internet drafts specifying our solutions, which already have been adopted, our solutions might be integrated in future systems by default.

\(^1\)The standard approach is only more efficient when a very little amount of services is discovered, in which case the little overhead of device discovery does not pay off. But, in cases where such a little amount of services is discovered, the effect on the network load is negligible anyway.
9.3 Future Work

In the future, we will foremost work on our Internet drafts [HK16a, HK16b], which are based on Chapter 5 and Chapter 6, respectively, accompanying them on their way to becoming RFCs, i.e. Internet standards. We consider this an important step in the pursuit of privacy in the information age. As a part of this work, we will work on reference implementations of these drafts.
Bibliography


[80212] Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, 03 2012.


S. Bortzmeyer. DNS Privacy Considerations. RFC 7626 (Informational), August 2015.


Bibliography


[FMB+14] Tilman Frosch, Christian Mainka, Christoph Bader, Florian Bergsma, and Thorsten Holz. How secure is textsecure? 2014.


Bibliography


Bibliography


