Colibri: A Toolkit for Rapid Prototyping of Networking Across Realities

Sebastian Hubenschmid*  Daniel Immanuel Fink*  Johannes Zagermann  Harald Reiterer  Tiare Feuchtnner

1University of Konstanz  2Aarhus University

ABSTRACT
We present Colibri, an open source networking toolkit for data exchange, model synchronization, and voice transmission to support rapid development of distributed cross reality prototypes. Development of such prototypes often involves multiple heterogeneous components, which necessitates data exchange across a network. However, existing networking solutions are often unsuitable for research prototypes as they require significant development resources and may be lacking in terms of data privacy, logging capabilities, latency requirements, or supporting heterogeneous devices. In contrast, Colibri is specifically designed for networking in interactive research prototypes: Colibri facilitates the most common tasks for establishing communication between cross reality components with little to no code necessary. We describe the usage and implementation of Colibri and report on its application in three cross reality prototypes to demonstrate the toolkit's capabilities. Lastly, we discuss open challenges to better support the creation of cross reality prototypes.

1 INTRODUCTION
Over the past few years, there has been a substantial increase in research prototypes for cross reality (CR) environments, mainly concentrating on the web and the Unity game engine [31]. One driving force behind this growth is the availability of increasingly sophisticated toolkits for these development environments: For example, collaborative mixed reality (MR) systems "have only recently advanced to the point where researchers can focus deeply on the nuances of supporting collaboration, rather than needing to focus primarily on creating the enabling technology" [7]. Although there has been a proliferation of toolkits in different areas such as visualization [6, 9, 27, 30] or logging [13, 22], networking has been mostly neglected and delegated to commercial solutions. Networking is an essential part in many interactive CR prototypes, for example to support collaboration across realities (e.g., multiple homogeneous [8] or heterogeneous [29] devices) or to connect complementary interfaces [33] (e.g., transitioning between desktop and MR [15]). In contrast to commercial applications, research prototypes have distinct requirements regarding the empirical reproducibility, data availability, latency, and privacy – ruling out externally hosted server software while simplifying development by neglecting edge cases (e.g., anti-cheat precautions) or artificial restrictions (e.g., reducing update rate to save on bandwidth). Commercial networking solutions may also create additional difficulties for CR prototypes: For example, a common requirement for CR prototypes is the synchronization of objects across coordinate systems with different initial reference points. Here, the provided object synchronization algorithms must consider the difference in coordinate systems, which makes the use of naïve synchronization scripts impractical.

To better address the distinct needs of CR research prototypes, we created Colibri (communication library), an open source networking toolkit for data exchange, model synchronization, and voice transmission. In the following sections, we review related work, outline the usage and technical implementation of Colibri, showcase its capabilities based on three prior research projects, and discuss open challenges to further support researchers in creating CR applications.

2 RELATED WORK
Toolkits have long been a driving force to reduce development barriers for research prototypes. For example, the Studierstube project [28] was one of the enablers of early AR prototypes. Similarly, toolkits such as ubitrack [24] and proximity toolkit [20] allowed for a proliferation of research prototypes using tracking and proxemic interaction, respectively. Other research-driven frameworks, such as Webstrates [17] and its variants [2,12,26], support developers in seamlessly synchronizing and sharing content across web-based devices. Especially in the field of InfoVis, toolkits such as IATK [6], DXR [30], u2vis [27], and RagRug [9] play an essential role to significantly reduce the effort required to create data visualizations. Recent toolkits such as MR Atlas [22] and ReLive [14] also include data capturing capabilities to record and analyze MR study data.

Similarly, commercial toolkits can facilitate development across different platforms and provide essential development resources. For example, Microsoft's Mixed Reality Toolkit [21] is fundamental for creating interactions in MR environments. In terms of networking, commercial solutions (e.g., Photon Fusion [25], Unity Netcode [23]) provide highly flexible networking solutions, targeting a broad audience. However, as discussed by Friston et al. [10], this flexibility comes with significant limitations, such as requiring significant integration efforts, supporting only one platform (e.g., Unity but not web), requiring external third-party services, making strong assumptions – usually in favor of game development requirements such as anti-cheat behavior – and lacking capabilities necessary for research, such as data logging. Beyond commercial solutions, the Ubiquity system [10] is most closely related to our own work and addresses many of these issues. However, it focuses on the specific needs of large-scale social virtual reality (VR) systems: For example, Ubiquity aims to balance a large number of clients with available throughput, thereby introducing additional latency. In contrast, Colibri aims to provide the ease of use of commercial networking solutions while also making use of the distinct advantages of research prototypes (e.g., ideal lab conditions with little to no latency) to facilitate CR development.

3 COLIBRI
With Colibri, we aim to provide a networking toolkit that is specifically tailored towards CR research prototypes, favoring a quick and easy setup without extensive programming and reduced complexity through simplifying assumptions (e.g., small-scale deployment in controlled labs), while prioritizing latency. Our toolkit can be integrated into Unity [31] or web applications and uses a web server to manage clients and distribute data. Colibri takes care of common

*Contributed equally
networking tasks such as data exchange, model synchronization, persistent data storage, voice transmission, and logging.

The following sections describe the usage and implementation of Colibri. For brevity, we focus on its use in Unity projects, but similar methods are available for web clients. Please refer to our project page¹ for more detailed documentation and examples.

### 3.1 Data Exchange

In the most general case, Colibri facilitates sending data through publish/subscribe communication [4]. Data can be published from anywhere in the executed code, as illustrated with the following simple example of sending an integer value \((myInt)\) on a "click" channel:

```csharp
Sync.Send("click", myInt);
```

The sent data can then be received anywhere in the application by registering a listener as follows:

```csharp
Sync.ReceiveInt("click", (myInt) => {
    /* Will be called whenever an integer on
    "click" channel is received */
});
```

This allows developers to send data of any type, including custom classes (i.e., via JSON serialization), across the network. Published messages are ephemeral and thus best suited for sending events, such as button clicks.

### 3.2 Model Synchronization

For more complex cases, such as persistently synchronizing data models between different clients, Colibri provides a model synchronization behavior similar to state-of-the-art solutions (e.g., Unity Netcode [23]). Consider, for example, a data model that represents (and is attached to) a cube:

```csharp
public class CubeModel : SyncBehaviour {
    [Sync]
    public Vector3 Position {
        get => transform.position;
        set => transform.position = value;
    }
}
```

We extended Unity’s ```MonoBehaviour``` with a ```SyncBehaviour``` that keeps track of any properties marked with the ```[Sync]``` attribute, thereby automatically synchronizing the model’s marked properties with other connected clients. Similar to the data exchange, this synchronization supports all primitive data types (e.g., booleans, integers, floats, strings, and arrays thereof) as well as common Unity structs (e.g., ```Vector3```, ```Color```), out of the box, while custom classes can be sent through JSON serialization. For each data model, we define a manager component to keep track of all instances of the model:

```csharp
public class CubeManager : SyncedBehaviourManager<CubeModel> { }
```

Here, we only need to define an empty manager that inherits ```SyncedBehaviourManager``` for the data model. We add it to the Unity scene, and provide it with a Unity ```prefab``` to allow for dynamic instantiation of objects: when an object is created on one client, Colibri’s ```SyncedBehaviourManager``` will automatically instantiate this ```prefab``` on all connected clients. Models are stored persistently on the server and are automatically retrieved whenever a client connects. Since synchronizing the transform data (i.e., position, rotation, scale) of an object is a common use case, Colibri provides a ```SyncTransform``` script (and matching manager) to easily synchronize object transforms without requiring any additional code.

While Colibri supports the synchronization between different coordinate systems implicitly (i.e., by synchronizing ```localPosition``` and ```localRotation```), we plan to investigate the automatic alignment of coordinate systems between devices with different initial points of reference in the future.

### 3.3 Persistent Data Storage

While the examples above mainly focus on dynamic data exchange, some actions (e.g., calibrating a room for a study) are usually performed less frequently and require more persistent data storage. For this, Colibri offers persistent data storage, allowing data to be permanently saved on the server and easily shared across all connected clients. For example, consider saving calibration data (e.g., origin and orientation of coordinate system) in a ```[Serializable]``` ```Calibration``` class. Once the calibration is complete, it can be uploaded by specifying a unique name and the data (e.g., instance of ```Calibration``` that should be uploaded:

```csharp
Store.Put("calibration", calibrationData);
```

Similarly, any client can then retrieve this data from the server by specifying the name and expected class:

```csharp
await Store.Get<Calibration>("calibration");
```

### 3.4 Voice Transmission

Similar to prior toolkits [10], Colibri also provides built-in voice transmission for remote collaboration scenarios. Voice transmission can be easily included in any project by adding a predefined ```Voice Broadcast``` (for audio recording) and ```Voice Manager prefab``` (for audio playback) to the Unity scene, without the need for any additional code. Each new client will then appear as a customizable entity, thus supporting spatial audio.

### 3.5 Logging

In our own experience, prototypes often break unexpectedly when deploying to different devices (e.g., Microsoft HoloLens, Android, iOS). In these cases, showing the logging output on the device itself is often not feasible or involves tedious debugging setups (e.g., connecting an iPhone via cable to Apple’s Xcode IDE). To aid this troubleshooting process, Colibri can automatically send logging data, such as the console output, to the server and display it as live data feed on the web interface.

### 3.6 Technical Implementation

Colibri is built on a client-server architecture, with client implementations for Unity and JavaScript. When setting up a client, developers have to simply specify the server’s address (either local or remote) as well as their application name, which allows multiple research projects to share the same server without interfering with each other. Our server is written in TypeScript using NodeJS and provides a web interface built with Angular and thus runs on any platform supported by NodeJS (e.g., Windows, macOS, Linux). This allows for server-side broadcasting and recording (e.g., ```for voice transmission```), as well as centralized data storage (e.g., ```for persistent data storage```).

In the future, we want to investigate redirecting synchronized data to logging toolkits such as MRAT [22] or ReLive [13], which could free up client resources. In addition, broadcasting messages could further reduce setup friction for local area networks and allow for automatic server discovery.

We employ a single TCP port for connections with Unity clients and a WebSocket port for web clients for general data synchronization, which ensures that messages arrive safely and in order. In terms of data transmission, every client transmits differential updates as JSON packets, using functional-reactive programming frameworks such as UniRx and RxJS. In addition, we utilize UDP packets for voice transmission, while the persistent data storage uses HTTP requests to support large files.

¹https://github.com/hcigroupkonstanz/Colibri
were set up under lab conditions (i.e., Ethernet connection where
screenshot
“ channel.

Figure 1: Relive [13] is a CR application that combines an immersive
analytics virtual reality view (left) with a synchronized visual analytics
desktop view (right). The application state of Relive is kept in sync
between both views using Colibri.

4 CASE STUDIES

To demonstrate the flexibility and broad application range of Colibri,
the following sections describe how we employed Colibri in our
own projects, ranging from hybrid user interfaces for single users,
to remote multi-user scenarios. While the selected works demon-
strate the key features of Colibri, our toolkit has been used in many
additional (as-of-yet) unpublished works and is therefore steadily
improving. Please refer to the original publications for more details
on the specific use cases described below. All included projects
were set up under lab conditions (i.e., Ethernet connection where
available, otherwise 5 GHz WiFi with an average < 5 ms round trip
times) and are available as open source project2.

4.1 Relive

Relive [13] is a CR visual analytics framework that combines an
immersive VR environment with a non-immersive desktop environ-
ment to enable in-depth analysis of MR user study data (see Fig. 1
and accompanying video3). Relive allows analysts to create visual-
izations of logged data, such as the movement trail of a study partic-
ident based on their recorded positions over time. This visualization
can then be inspected both on a desktop in 2D (e.g., as top-down
movement path) and in a VR environment in 3D (e.g., embedded
within the 3D reconstruction of the original study environment). We
implemented the 2D desktop application using web technologies,
while the immersive VR environment was implemented in Unity for
the Meta Quest 2.

Here, we used Colibri to automatically synchronize the entire
application state (e.g., visualizations) using the model synchro-
nization. For example, we created a data class to represent visualizations
(e.g., name, type of visualization, color) and utilized the model synchro-
nization function in Colibri to automatically keep the VR and
desktop environments in sync. Since synchronized data models are
consistent, refreshing the web page or restarting the VR client auto-
matically retrieves previously created visualizations. Relive supports
a variety of actions to aid data analysis, such as taking a screenshot
of the VR scene by pressing a button in the desktop interface. For
this, we registered a listener as described in data exchange (see
Sect. 3.1), which was then triggered by sending a message from the
web client on a predefined “screenshot” channel.

4.2 ARound the Smartphone

In the “ARound the smartphone” [16] project, we created a study
prototype in Unity that uses the Varjo XR3 as an augmented reality
(AR) head-worn display (HWD) to extend the physical screen of
a Google Pixel XL smartphone with a surrounding virtual screen
space (see Fig. 2 and accompanying video4). Using this setup,
study participants were tasked to navigate a grid map with touch
gestures, as known from state-of-the-art map applications. Since

Figure 2: In “ARound the smartphone” [16] we use an AR HWD to
virtually extend the physical screen space of a smartphone. In this
project network latency was a priority, as the virtual and real screen
had to be kept perfectly in sync.

the visualized map virtually extended beyond the smartphone screen
using AR, the map position had to be synchronized between both
devices (e.g., from smartphone to AR HWD). We achieved this
by attaching the Colibri SyncTransform script to the map object
in Unity and adding the SyncManager prefab to the scene – thus
keeping the location of both maps (i.e., on the smartphone and in the
AR environment) in sync without having to write any code. Here,
we can easily add spectators (e.g., another AR HWD) since the
map’s position is automatically broadcasted to all connected clients
(cf. spectator viewer [14]). As Colibri prioritizes low latency, both
the smartphone screen and the virtual extension were kept (almost)
perfectly in sync, despite the high refresh rate of current AR HWDs
(i.e., 90+ Hz). Due to different coordinate systems of the smartphone
and AR HWD, we mounted a fiducial marker to the smartphone to
align the map’s position between both devices.

4.3 Re-locations

Re-locations [8] is a CR environment implemented in Unity for
the HoloLens 2 that explores the remapping of different physical
workspace layouts in remote collaboration scenarios in AR, to facil-
itate a common reference frame and the use of deictics (see accom-
ppanying video5). For example, a pair of remote collaborators might
work on a shared task using their respective desktop and whiteboard
(i.e., two physical workspaces), which are likely in different relative
locations within their rooms (see Fig. 3). They see each other as
equivalent virtual avatars, allowing them to be co-present in each other’s work
spaces. To support a common spatial awareness and understanding,
e.g., through semantically correct pointing gestures across these
incongruous spaces (e.g., pointing at the whiteboard while sitting
in front of the desktop PC), Re-locations automatically remaps
the position and rotation of the remote user’s avatar to make sense in
the local user’s workspace layout.

To correctly set up these physical workspaces in the real world,
we calibrated each room once and saved the corresponding data
using the persistent data storage in Colibri. Workspaces were then
automatically retrieved by scanning an AR marker containing the
ID of the room. In addition, the project uses the Colibri voice trans-
m ission capabilities to enable spatial audio between collaborators.

5 DISCUSSION AND OPEN CHALLENGES

Although networking is an essential part of many CR applications,
the development of such research prototypes spans many more ar-
eas that require further support. Since sophisticated toolkits can
drastically reduce the technological barriers, we discuss four open
challenges, which we do not yet find adequately supported by exist-
ing toolkits.

2https://github.com/hcigroupkonstanz
3https://youtu.be/BaNZ02QkZ_k
4https://youtu.be/p6cHwLxHWJg
5https://youtu.be/D6_B4Rux1U
Alignment of Coordinate Systems. Another common task when combining multiple systems on different points of the reality-virtuality continuum is the alignment of coordinate systems. Nowadays, many devices have powerful integrated tracking capabilities, making external tracking solutions (e.g., Optitrack) unnecessary. However, this also results in each device establishing their own coordinate system, which then needs to be aligned with all others. In our experience, these solutions are usually specific to the hardware configurations of each project and involve a combination of image markers, external tracking systems, or specific workarounds (e.g., starting the application in the same room position). A unified solution that automatically provides coordinate system alignment could greatly reduce the obstacles of combining multiple devices.

Transitional Interfaces. Another key area for CR applications is transitional interfaces, which support ongoing interaction across various points on the reality-virtuality continuum. For example, Apple recently showcased one such transitioning technique between AR and VR through turning a knob on the side of their new Apple Vision Pro HWD [1]. Although other techniques have been explored (e.g., [3, 11]), these are usually not easily integrated within other research projects. Here, we see the potential for a library of transitioning methods that could easily be replicated.

Asymmetric Device Capabilities. CR applications can span across multiple heterogeneous devices, such as desktop computers, handheld devices (e.g., smartphones or tablets for AR), or head-worn MR devices. Each device in this ecology has different capabilities or limitations and, at times, a distinct role to fulfill. Here, automatically detecting, communicating, and assigning roles based on capabilities could be beneficial. For example, in asymmetric collaborative environments for immersive analytics, a desktop device can be useful for displaying visualizations in 2D, while other MR devices could be better suited for 3D visualizations. Here, we could specify data transformations (e.g., [18, 19]) to automatically convert the data, based on the specific capabilities or roles of the display device.

Merge Policies. In many collaborative scenarios, multiple collaborators may manipulate an object simultaneously, such as two users moving or rotating an object at the same time (see Fig. 4). In such cases, a merge policy (e.g., averaging or summing up inputs) is necessary to properly incorporate the manipulations of both users (cf. [32]). Although Colibri already supports simultaneous manipulation of the different properties of an object (e.g., one user rotates, another translates) through its differential updates, our toolkit does not yet support simultaneous input of the same property (e.g., multiple users rotate an object).

Figure 3: The Re-locations [8] concept semantically corrects gaze visualizations across different user-defined layout areas within physical workspaces during remote collaboration in augmented reality. For example, if Anna is standing close to her desktop while looking at the whiteboard screen (right) the 1:1 mapping of her rotation will be interrupted, and her avatar rotates towards the whiteboard in Paul’s environment (left).

Figure 4: A composition merge policy allows two users to simultaneous manipulate an object, for example by summing up users’ rotation inputs [32].

6 Conclusion and Future Work

Colibri is a Unity- and web-focused networking toolkit for researchers to quickly prototype cross reality applications. Our toolkit requires little to no code and aids in common networking tasks, such as data exchange, model synchronization, and voice transmission. It prioritizes low latency to suit the needs of research prototypes. This toolkit was developed and extended through the course of our own projects (i.e., [5, 14]) and is steadily improved through new projects in our research group. However, to adequately support the development of cross reality applications, further work is necessary, for example by communicating asymmetric device capabilities, automating coordinate system alignment, providing a set of transition techniques to quickly change between interfaces at different points on the reality-virtuality continuum, and supporting merge policies. While Colibri presents a first step towards supporting research prototypes, we strongly believe that the development of more sophisticated toolkits will be needed to pave the way for increased adoption of cross reality applications in other application scenarios.

Colibri is available as open source project on GitHub: https://github.com/hcigroupkonstanz/Colibri

Acknowledgments

This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project-ID 251654672 – TRR 161.

References


