XRShark: Mixed Reality Network Introspection

Meghan Clark
mclarkk@berkeley.edu
University of California, Berkeley
Berkeley, California

Mark W. Newman
mwnewman@umich.edu
University of Michigan
Ann Arbor, Michigan

Prabal Dutta
prabal@berkeley.edu
University of California, Berkeley
Berkeley, California

(a) (b) (c)

Figure 1: A connected office as seen in the augmented reality and virtual reality modes of XRShark.

ABSTRACT

As devices become increasingly mobile and wireless, the last-meter network monitoring problem continues to grow. Network introspection tools will need to support an up-to-date operating picture of dynamic networks, show the physical locations of all network nodes, and connect low-level spatial characteristics with the higher layers of the network stack. Such insights are poorly supported by existing tools that abstract away the physical reality of the underlying network. Fortunately, recent advances in localization technologies and mixed reality platforms provide an opportunity for a new approach: mixed reality network introspection. Mixed reality allows users to see network traffic situated directly in the real world. Physical situation of network activity supports an intuitive understanding of room-scale networks by harnessing human spatial intuition and visual processing, which is well-suited to rapid identification of unexpected activity and correlations in three dimensional space. We propose design goals and a general architecture for mixed reality network introspection, and implement a prototype called XRShark, a network visualizer with augmented reality and virtual reality modes. XRShark enables us to identify several behaviors of the local networks that would be difficult to detect using traditional tools. Based on our prototyping experiences, we discuss promising directions for mixed reality network introspection, as well as remaining challenges for the system design and interaction semantics.

1 INTRODUCTION

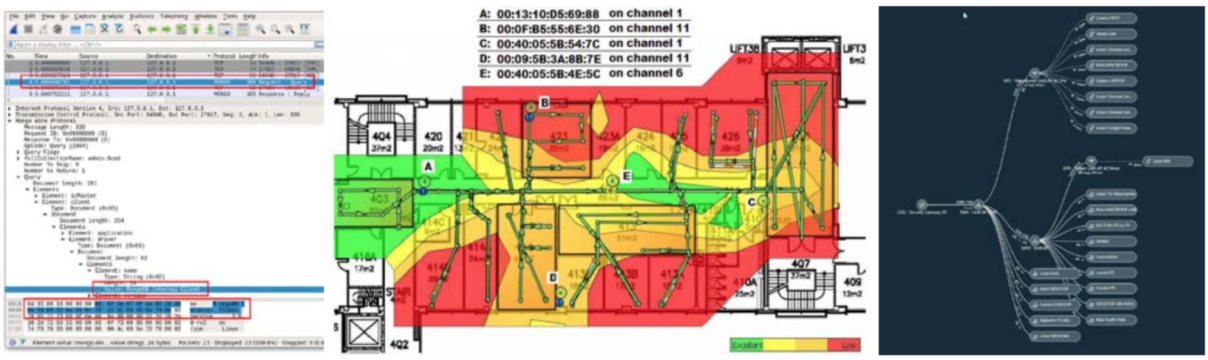
Wireless packet networks have existed since the debut of ALOHAnet in the early 1970s, but recent decades have seen them become truly ubiquitous. Advances in transceiver hardware have driven an explosion of wireless technologies, such as Bluetooth Low Energy, WiFi, and several generations of cellular networks. As 5G and LPWAN infrastructure roll out, the wireless expansion shows no sign of stopping.

While networking technologies have evolved over time, many of the staples of the modern network debugging toolbox would be familiar to early network pioneers. These tools are dominated by text-heavy readouts well-suited to the terminal, such as *tcpdump*, *traceroute*, and *dig* [27]. Many subsequent tools have inherited from these spiritual predecessors, such as the Wireshark packet analyzer with its text-heavy tabular views and filters [30]. Later came network topology graphs and 2D maps that despite their graphical nature remain abstracted from the physical space.

These tools may suffice for networks that primarily consist of large, static, or wired hosts, but the landscape has changed. The growth of wireless technology has developed hand-in-hand with a proliferation of network-connected devices, many of which are embedded, mobile, proximal, or cyber-physical systems. Modern local networks are therefore often highly dependent on spatial relationships that may change from moment to moment. Maintaining a real-time understanding of this kind of dynamic networking environment will require new tools that support rapid, intuitive, and physically grounded insights into real-time network activities.

Recent years have also seen the emergence of consumer-grade augmented and virtual reality platforms and commercially available localization systems that can track objects in 3D space with decimeter accuracy. With these advances, we can now do what we could not–and indeed, did not have to do–before: visualize wireless network phenomena situated directly in physical space.

We propose that the time is now ripe to explore *mixed reality network introspection*, a new way of understanding networks that yields serendipitous insights that have so far been poorly supported by existing network monitoring and debugging tools. Existing tools require users to know what they are looking for, and require time and mental effort to translate symbolic representations into an understanding of what happened. However, by helping users literally



(a) Wireshark (b) Ekahau (c) Ubiquiti

Figure 2: Modern network management tool displays.

see network traffic, we can leverage human sensory processing, which is tuned to reflexively notice patterns and unexpected activity in three-dimensional space. This approach could allow even casual users to quickly detect unexpected activity, build an intuitive impression of activity across the network, and understand how the physical layer is impacting higher-layer networking behaviors.

In this paper, we introduce design goals and a general architecture for mixed reality network introspection. To validate the essential system capabilities and user experience, we build a prototype called XRShark. We use XRShark to monitor and visualize a smart office with wireless sensors and actuators connected via WiFi (802.11) and OpenThread (802.15.4) mesh networking. We describe the rationale behind XRShark's implementation choices, report microbenchmarks to validate the semantics, and share examples of the unique network insights that XRShark can provide. We close with a number of remaining challenges to drive future research in this promising direction.

2 BACKGROUND AND RELATED WORK

A number of recent works in network introspection, mixed reality for networks, and device localization have hinted at this direction in bits and pieces. Together, they suggest a confluence of forces for which mixed reality network introspection is the natural next step.

2.1 Network Introspection

Several notable works have focused on designing network infrastructure for visibility. X-Trace is a framework that provides a comprehensive picture of multi-layered and inter-operating networked services for the purposes of root cause analysis [12]. By embedding metadata throughout a system, X-Trace is able to capture relationships and data flow between several different network layers.

Networks of embedded systems have similar debugging needs, but critically, there is an energy cost to both visibility and diagnosis [29]. While making metadata visible costs energy in low-power sensor networks, so does debugging. Consequently, Wachs et. al. claim that protocols should "minimize the energy cost of diagnosing the cause of a failure or behavior," and propose several formal metrics for evaluating protocol optimality.

There is a significant body of work focused not just on how to collect network information, but also on how to visualize it in ways that help users accomplish tasks. To aid embedded application debugging, WiFröst addresses the problem of linking together code execution with network behaviors in an embedded system context [19]. Showing correspondences across different layers and different domains is key for debugging many networked and embedded applications. Security professionals are also particularly interested in visualizing network information in ways that help humans identify anomalous patterns and malicious actors [8, 13].

2.2 Mixed Reality for Networks

The last few years have seen key pieces of research applying augmented reality to networked embedded systems. EyeSec is an augmented reality network introspection tool developed to help troubleshoot wireless sensor networks in the field [26]. The system architecture focuses on easily deployable sniffer modules that may be limited in what they can access, but can be retrofit to already deployed sensor networks. Device localization is performed by a smartphone using QR codes, and the phone also runs the AR visualization. The interface shows information tiles over each QR code enabled device, and the thickness of arrows between device tiles represents the total number of packets recently sent.

EyeSec addresses a number of the same challenges that we do, but limiting the visualization to just augmented reality misses the opportunities that virtual reality provides for remote monitoring and review of historical data. The semantics of augmented reality clash with replay of historical data, and the user must physically travel to the site to use the interface. The addition of a virtual reality mode, like we provide in XRShark, supports replay of historical data in its physical configuration, and the ability to perform real-time debugging remotely.

Other projects have used mixed reality to interact with smart devices at the application level. MIT's DoppelLab uses VR to view sensor data in virtual twins of smart buildings in real time [11]. Smarter Objects enables programming of Internet of Things by drawing connections between devices in augmented reality [14].

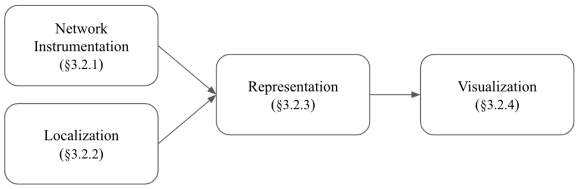


Figure 3: General architecture for mixed reality network introspection.

There has also been an interest in using mixed reality interfaces to visualize network information in the commercial world. For example, SeeSignal is an augmented reality product for visualizing spatial RSSI data [6]. However, SeeSignal only provides a static view of an RSSI map, and does not display any other network behavior or information. Splunk, an analytics company, has prototyped augmented reality interfaces that display system resource information on server racks in data centers [16].

2.3 Localization of Network Devices

To automatically localize and visualize traffic from network devices requires both pinpointing the location of each device and mapping each localized entity to its network identity. Anchor-and-tag localization systems like Harmonium [15] can address the first goal, but require manual effort to achieve the second. Tag-free localization solutions like ALPS [17] or the commercially available Quuppa [24], which use Bluetooth Low Energy radios, could potentially achieve both goals. Because the radio used for localization is already integrated into the tracked device, it is easier to automatically associate the localization and network identities of the device. Camera-enabled devices can use computer vision to localize themselves within the space, but may have difficulty sharing that location in a global coordinate system that other devices can understand. Camera-based solutions also do not generalize well to other types of network devices.

For situations where the visualization system is a smartphone, VisIoT converts the problem into an easier to solve domain by localizing a broadcasting network device directly onto the correct screen pixels on an augmented reality smartphone, using a combination of vision and RF information [23]. However, this approach creates dependencies between the localization and visualization components that reduce the modularity of the system.

3 ARCHITECTURE FOR MIXED REALITY NETWORK INTROSPECTION

To achieve our vision of physically grounded and serendipitous insights, the architecture for mixed reality network introspection must achieve a number of design goals. These goals have implications for each of the four main architectural components: network instrumentation, localization, representation, and visualization.

3.1 Design Goals

A mixed reality network introspection system should:

- Support multiple network representations.
- Show interactions between network layers.
- Have well-defined time semantics.
- Have well-defined location semantics.
- Enable quick filtering.
- Support "here-and-now" temporal and spatial synchrony.
- Support historical/remote temporal and spatial asynchrony.

There are a number of different representations used to conceptualize networks, including tabular, topological, and physical models. Each of these representations are appropriate for different tasks. Mixed reality is particularly suited to emphasizing activity related to physical aspects of the space, but the system should still allow users to augment their understanding and structure their exploration with these other views.

Networks are also multi-layered, and the characteristics of the physical layer may have impacts on the link layer and IP layer. Additionally, different layers are relevant to different tasks. Users should be able to switch between views of these layers and see the correspondences between them.

The time semantics of the interface should accommodate both real-time and historical data. When viewing real-time traffic, accurate timing is critical. However, with historical data, it may sometimes be more useful to step through one packet at a time, with greater emphasis on the sequence of events rather than the timing.

The location semantics of device placement will need to convey the accuracy, precision, and freshness of the location information. Not all network devices may be physically localized in space, and the difference must be clear.

The system must allow users to quickly construct, apply, and remove filters to avoid becoming overwhelmed by the firehose of data. Users should be able to quickly highlight only those things they are interested in.

Finally, the system should enrich the environment when the user is physically present at the site, but it should also allow the user to access the site remotely. Viewers should be able to have meaningful real-time insights, but also be able to review historical data as part of their investigations. This means supporting both synchrony and asynchrony in the spatial and temporal domains.

3.2 Architecture

The main architectural components of mixed reality network introspection and their relationships are shown in Figure 3. Though each component encompasses a large space of possible implementations, the design goals introduce a number of important considerations.

3.2.1 Network Instrumentation. The network instrumentation will need to capture information from a number of different layers, including the physical, link, and network layers. Where this instrumentation should sit depends on the topology of the network, the needs of the application, and the ability to access the routing infrastructure.

Collecting high-layer information is especially reliant on topology. Star networks can often be monitored from a single point, if you have access to the central router. Traffic to and from a mesh network can also be intercepted just from the border router, but to capture intra-mesh communications would likely require modifying each mesh router. Monitoring peer-to-peer networks like

Bluetooth Low Energy would require either expensive sniffers or the modification of a least one of the participants.

In cases where the infrastructure cannot be modified, network clients can still make a best-effort case to see what they can see. However, this limited perspective and the possible presence of undetected activity should be conveyed to the user.

Collecting physical layer information about the space is difficult and likely requires deploying RF measurement devices. While providing a comprehensive real-time map of the RF spectrum might impose significant infrastructure costs, to help users achieve narrower goals such as gateway placement, spectrum management, and wireless troubleshooting, we could potentially capture the relevant information with significantly fewer sensing points.

The network instrumentation must support both real-time visualization and historical replay. Capture must be fast enough to avoid falling behind real time, with small enough overhead to avoid negatively impacting the network. Replay requires the ability to store and retrieve captured data. Many existing tools, like Wireshark, allow user-initiated packet captures. However, to support serendipitous insights about unexpected transient behavior, the instrumentation system will need to be able to replay data that *the user did not know to record in advance*. This can be achieved by keeping a data buffer for replaying events that just occurred, and periodically off-loading local packet capture data to long-term storage.

3.2.2 Localization. Capturing the basic shape of the room is important for orienting users and device locations within a digital twin of the space. The room model can be easily parameterized as the dimensions of a rectangle or other basic shapes, potentially with the locations of key landmarks like doors and windows.

In Section 2, we discussed a number of possible methods for localizing devices within the room. Despite their different characteristics, these approaches may end up combined together into one visualization, making it particularly important for each localization system to convey the quality of the data in terms of accuracy, precision, and freshness.

3.2.3 Representation. Information from the network instrumentation and localization systems should be synthesized into a model that represents the known state of the instrumented network(s). This model would include up-to-date locations and other associated metadata for each network device. A standalone representation service would allow multiple different visualization applications to access the same data.

3.2.4 Visualization. Visualization must balance both exploration and exploitation. A key postulate of our vision is that mixed reality visualization naturally supports exploration and serendipitous insights by bootstrapping off of human real-world spatial intelligence. However, to support at-a-glance awareness of overall network activity, there are a number of aspects in which the semantics of the visualization should be carefully considered.

First, the placement of network hosts in the visualization should reflect the quality of the location information, including uncertainty and freshness. There will also likely be physically present devices whose locations are not tracked by the system. Other network hosts may not have a meaningful physical presence at all, such as those in the cloud. All of these entities will need to be visualized physically in a meaningful way.

Second, the semantics of viewing packets in real-time, when packets travel at the speed of light, presents a non-obvious fundamental challenge for any animated representation of packet transmission. Packet visualizations with any transit time (such as a ball moving from one host to another) end up violating rules of causality in the physical space, because the response to a packet may be sent before the original packet has been visually "received." To better match real-time packet semantics, we suggest representing packet transmission by the instantaneous appearance of a line between two end points. We discuss this in greater detail in Section 5.4.2.

For maximum utility, exploration must be complemented by exploitation. When potentially interesting behavior has been identified, the platform must allow for narrowing the scope and deeper investigation. This means allowing users to quickly filter their views to focus on relevant information.

Allowing users to switch between viewing different network layers, or switching between different models, can be done in a number of ways, such as by borrowing the mechanics of layer interaction from photo editing software, or a magic lenses metaphor [7, 9].

Enabling users to quickly construct and apply filters, however, is a difficult interaction design challenge. Many desktop tools use heavily text-oriented filters, which does not translate well into many AR and VR interaction systems. Mixed reality systems do support other modalities, such as direct manipulation and occasionally speech input, gaze detection, or pointing, presenting opportunities for novel ways of constructing and applying filters.

Finally, the visualization needs to support real-time uses, but should also support historical review of data. Similarly, the visualization needs to support real-space uses, but should also support remote viewing of data. While augmented reality is valuable for seeing the interaction between the environment and the devices, it requires the user to be present in the space and may cause confusion during replays. Any mixed reality network introspection system should also include a VR mode that supports review of historical data or viewing an instrumented site remotely.

4 XRSHARK PROTOTYPE

To explore the potential of mixed reality network introspection, we prototyped a tool called XRShark. Our goal in creating the prototype was three-fold: 1) to test the feasibility of the end-to-end system, 2) to explore the time and location semantics, and 3) to verify whether the user experience could provide unexpected and serendipitous insights. XRShark implements each of the key architectural components, as well as the core time and location semantics.

We deployed XRShark in a real-world connected office that contains low-power environmental sensors, smart lights, and several laptops, phones, and tablets. The environmental sensors are connected via a low-power OpenThread (802.15.4) mesh network with several mesh routers located in the room, and the remaining devices are connected via WiFi (802.11).

The sensors and smart lights are components of a distributed IoT application in development by embedded engineers. The sensors are custom environmental sensors that provide temperature, humidity,

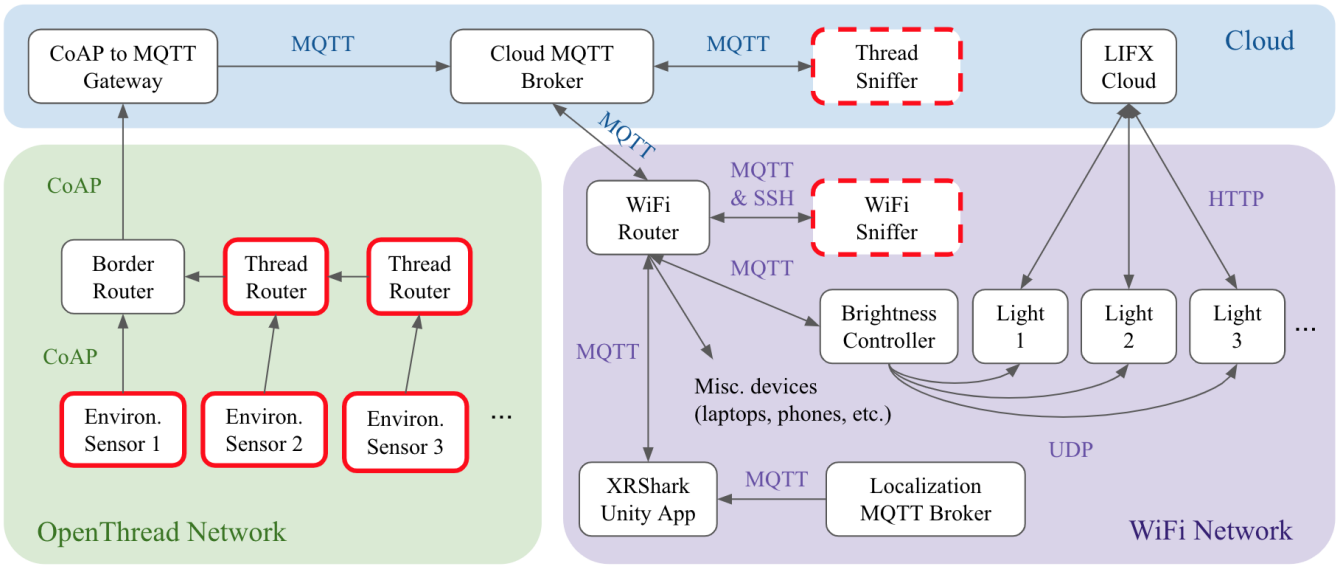


Figure 4: A smart office's networks modified to support mixed reality network introspection. Services that we added are highlighted in dashed red, and existing network devices we modified are in solid red. Only an illustrative subset of devices is depicted here. Since WiFi is a star network, we could capture all traffic with a single interception point (see Figure 5). Introspecting the Thread mesh network was more complicated (see Figure 6). Communication between the LIFX-brand smart lights, the LIFX cloud, and the brightness controller all pass through the WiFi router, but have been separated into their end-to-end relationships here for clarity about data flow. The localization MQTT broker is used by the ultra-wideband gateway (not pictured) to provide real-time mobile device locations.

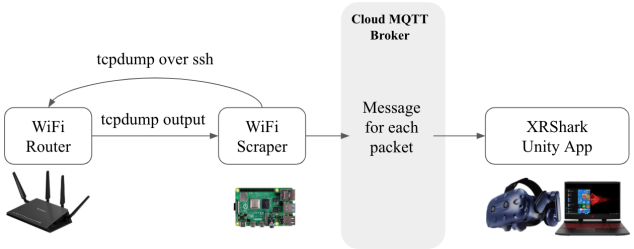


Figure 5: Capturing WiFi traffic. A scraper program runs tcpdump directly on an OpenWrt router and receives the output over SSH. The scraper sends packets to an MQTT topic structured according to metadata about the sender, receiver, and ports for filtering purposes. If running tcpdump directly on the router is not possible, tcpdump can be run on any computer and the available packets visualized.

brightness, and PIR-based motion detection. The smart lights are commercially available WiFi-connected lights from LIFX [18]. The embedded developers wrote an automatic brightness controller program that dims and brightens the lights to maintain desired brightness levels throughout the day. In addition to brightness readings, the application also uses the sensors' motion detection to automatically turn the lights on and off.

Initial discussions revealed that the embedded developers were particularly interested in visualizing the Thread mesh network. This would help them verify the correct operation of their OpenThread firmware implementation and help them adjust router placement to improve load balancing. They were also interested in seeing what office activities triggered the brightness and motion detection thresholds on the sensors, and in following the flow of data across the many links in the distributed application as it operates.

4.1 Network Instrumentation

The network structure of the smart office, shown in Figure 4, is somewhat complex, containing a star network (WiFi) and a mesh network (Thread) connected through cloud-based services. However, only a few modifications and additions were necessary to capture all WiFi traffic and all cloud-bound Thread traffic.

As a star topology, the WiFi network was simple to instrument. The WiFi router uses OpenWrt firmware [22], a Linux-based system that allows the use of tcpdump to capture packet data [27]. As represented in Figure 5, we developed a Raspberry Pi-based service that runs tcpdump directly on the router, receives the results through Secure Shell (SSH), and sends a message containing the packet information to a cloud-based broker via MQTT, a publish-subscribe network protocol [20]. This enables remote monitoring of the network activity. Our sniffer program has an interface for accepting packet filters. While pushing filtering into the capture infrastructure is a useful option for reducing network overhead, it

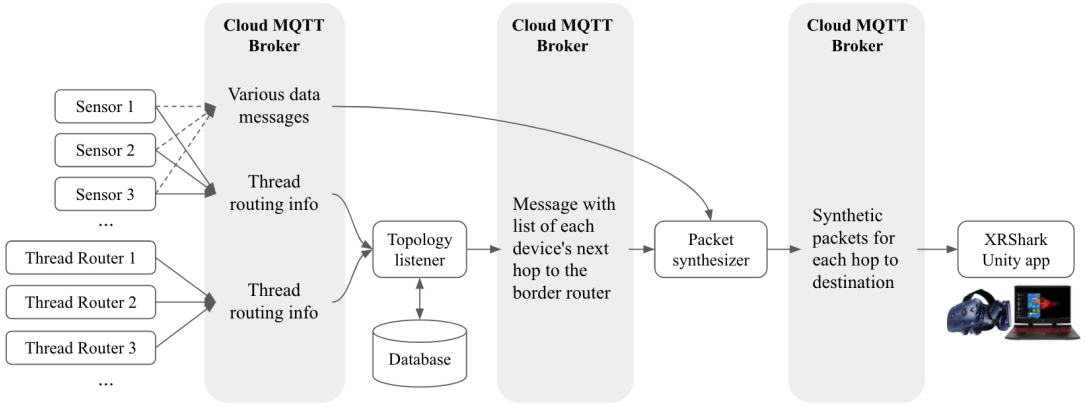


Figure 6: Synthesizing Thread mesh hops to the cloud. Sensors and routers report routing information. A service listens to these messages and tracks the global state of the topology. For each data message that the cloud receives from a sensor, a packet synthesizer generates synthetic packets for every hop through the mesh that must have occurred based on the topology.

would not be possible to change the filters on historical data. We always captured all packets that passed through the router.

Capturing the relevant Thread information was more complicated. While every sensor's data messages are sent to the cloud, the cloud cannot see what route the packet takes through the mesh network, which is what the embedded researchers were interested in visualizing. To keep the network overhead and firmware modifications minimal, we decided to take a synthetic approach where we retroactively determine the route each packet must have taken. We illustrate this process in Figure 6. The low-power sensors were modified to report the Thread router ID of their first hop in the network, and the wall-powered Thread routers were modified to report their routing tables upon infrequent intervals and upon any changes. We authored cloud-based services that monitor these messages and maintain a global picture of the topology. Each time a data message is received by the cloud, a packet synthesizer uses the current topology to determine what hops the packet must have taken through the mesh to reach the border router, and synthesizes a packet per hop that the XRShark Unity app then visualizes.

While the approach we took for capturing the Thread network imposes very little overhead on the Thread devices, its retroactive nature means it cannot detect cases when the actual behavior differs from the reported routing tables. It also cannot capture traffic local to the mesh. However, mesh-local packets could potentially be captured by modifying just the wall-powered routers to forward duplicates to a cloud endpoint, with some measures in place to prevent double-counting.

4.2 Localization

Most of the in-room devices are largely static. However, the smart lights could be controlled over the local network using a smartphone or tablet, so it was also important to be able to track the physical location of mobile network hosts.

Most of the static locations were determined by measuring real-world locations relative to a corner of the room with a measuring tape or laser range-finder. This manual process could be drastically sped up by supporting user placement of digital devices within the visualization. In augmented reality, users could raycast any automatically detected network devices to correct positions around the room. Various analytics could help aid the user in mapping a device's network identity to its real-world identity.

Figure 7: Decawave localization system. To localize mobile devices, we used the commercially available Decawave MDEK1001 kit, which provides decimeter accuracy at 10 Hz. Left to right are depictions of anchor, tag, gateway, and module. The UWB module is small and could be integrated into smaller tags. Four anchors covered a 15.5m x 4.5m room.

To track the dynamic locations of mobile devices, we integrated a tag-based real-time locating system (RTLS) into XRShark. As shown in Figure 7, we used the ultra-wideband (UWB) Decawave MDEK1001 commercial off-the-shelf localization system [10], which costs $299 for 12 nodes (about $25 per node). Four nodes can be configured to act as wall-powered anchors covering the whole room, and one is converted into a permanent gateway. The gateway exposes location data for each tag and anchor through a local MQTT broker. The tags can be battery powered and provide a 10 Hz update rate with typically sub-decimeter accuracy. The actual Decawave module providing the localization is about the size of a U.S. quarter.

Recent flagship smartphone models, including Apple's iPhone 11 and Samsung's Galaxy S21+, S21 Ultra, and Note20, are all UWB-enabled, suggesting a promising future for adoption of UWB localization technologies [5]. However, the localization interface to

XRShark allows the UWB system to be replaced with any localization method able to provide three-dimensional coordinates.

4.3 Representation

In our implementation of XRShark, the representation for the network is a collection of manually crafted lookup tables included as a file in the XRShark Unity application. These tables include the mappings of localization tags to network identities, network identities to locations, network identities to human-readable labels, and network identities to type information (such as Thread or WiFi). Though most of these entries are manually generated, some are also dynamically updated. For example, the Unity application receives MQTT messages containing location information for tags in the UWB localization system, and updates the location for the corresponding network devices accordingly. Additionally, whenever a Thread or WiFi packet message arrives via MQTT, if one of the endpoints has never been seen before it is automatically added to the appropriate lookup tables.

Though this representation of the local networks is currently maintained by the Unity application, in the future we expect this functionality to reside as a separate network-based service. Much of the information contained in the lookup tables, such as mappings between network identities and names, is duplicated and kept up-to-date manually in multiple applications running in the connected office, including the brightness controller program. By separating the network representation into its own standalone lookup service, all of these applications can benefit from a centrally maintained and automatically updated metadata store.

4.4 Visualization

XRShark visualizes the network and localization information using a Unity application that runs on a HTC Vive Pro headset. While the Vive Pro is primarily designed for virtual reality, it also has front-facing passthrough cameras that enable augmented reality. The application runs on Unity, a cross-platform game engine with extensive support for three-dimensional software development [28].

As seen in Figure 1, XRShark provides two modes, augmented reality and virtual reality, which users can toggle between. In each mode, wireless devices are indicated by large balls of light and labeled with an ID. The WiFi devices and cloud endpoints are all blue, and Thread devices are green. In implementing XRShark we explored solutions to a number of challenges concerning the location semantics of host placement, the time semantics of displaying real-time traffic, and how to visualize different network layers.

4.4.1 Host placement. In virtual reality, the room is represented by four black walls of real-world dimensions with outlines of doors and windows to provide orientation landmarks. Devices on the local networks are either placed in their physical locations around the room, or, if the location is unknown, embedded randomly in the floor (Figure 8). When updating mobile device locations, the visualization applies exponential smoothing to reduce visual jitter.

In the case of a less precise localization method, node placement could still incorporate whatever information is available. For example, placement in the floor plane could be biased towards one side of the room or the other depending on signal strength. However, the difference between this meaningful placement and random placement would need to be indicated to the user somehow.

Internet-based hosts are placed randomly in the cloud, an actual overhead plane filled with misty vapors (Figure 9). While the cloud layer can be seen from below in both AR and VR modes, in virtual reality users can fly up to and teleport around the cloud, and look down upon the local network. This vantage point is often convenient for observing the mesh network topology (Figure 11). Random placement could be improved upon by co-locating internet hosts with similar IP addresses.

4.4.2 Time Semantics. A significant challenge that we addressed in our interaction design was how to display real-time network traffic when real packets travel at the speed of light.

We initially represented packets simply as small balls of light traveling from sender to receiver. However, upon first viewing real-time traffic, we immediately discovered that the visualization violated rules of physical causality due to the transit time of the animation. Specifically, a second packet could be released in response to a first packet while the first packet was still visually traveling through the air, having yet to arrive at its destination. This made the semantics of network interactions difficult to understand.

This happens because real packets travel so fast that they are essentially received as soon as they are sent. A traveling ball with such a property would never be seen, since it would be immediately arrive at its destination.

These "instantaneous" semantics are more accurately depicted as the appearance of a line connecting the two end points. As soon as the line appears between two network entities, it means a packet was just sent *and received* in that instant. In XRShark the line fades over a user-configurable amount of time to capture a sense of recent history, and in particular, bandwidth. The more packets that are sent between two entities, the more lines appear, eventually resulting in a pipe structure. Users can get an idea of relative bandwidth usage over the time window by visually comparing how many lines are between various entities.

When a line appears, a particle also emanates, traveling from source to destination. While *line appearances* represent real-time packets, the balls provide a sense of recent history. The movement makes directionality clearer and make certain traffic patterns easier to notice. In particular, the balls illustrate how far apart in time the packets were sent, displaying temporal patterns that columns of lines do not.

The time window for retaining lines and particles can be manipulated using the touchpad on the Vive Pro controllers. Adjusting the time window causes the lines to fade and the packet particles to speed up or slow down as necessary to complete their transits within the selected time. This mechanic means that at any point, users know that all visible traffic was sent within the time window indicated by the controller readout (usually set between three and 30 seconds). Users can also completely pause playback, freezing packets in mid-air. We have begun to experiment with allowing users to reveal payload information for individual packets.

A color gradient along the length of each packet line also indicates directionality, so that directionality can still be seen when particle movement is paused using the time controller.

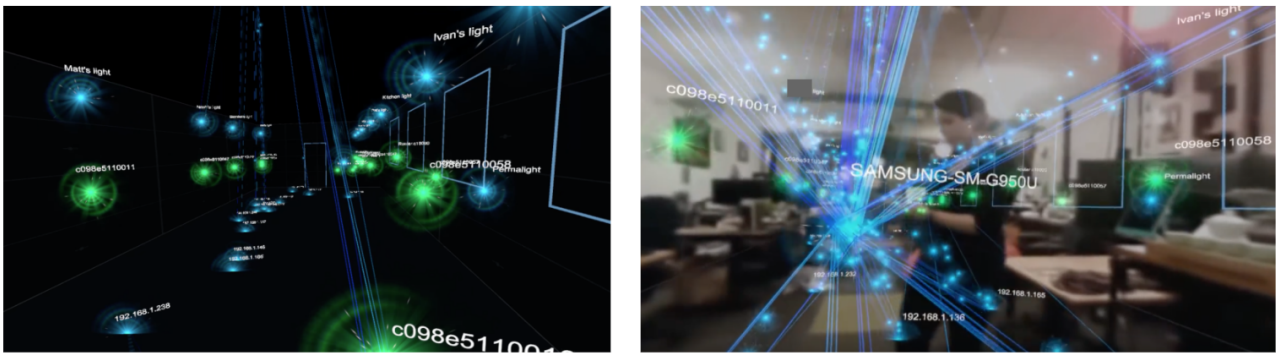


Figure 8: Local host placement. Local devices are placed in a digital twin of the room using location information provided by the system representation. Green orbs represent Thread devices, and blue orbs represent WiFi devices. If the physical location of a local device is unknown, then it is embedded randomly in the floor (a). Device locations are updated each time step, reflecting the latest localization information. In (b), as the smartphone moves, packets sent to it from the WiFi-enabled smart lights follow the phone's location.

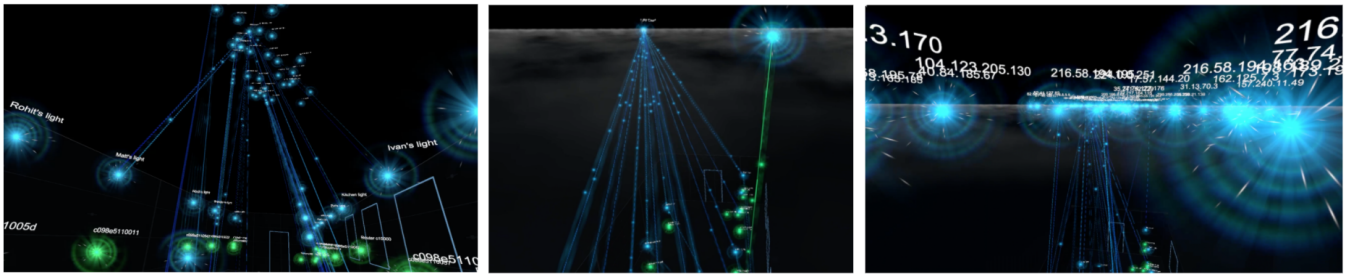


Figure 9: External host placement. Internet-based hosts are placed in a literal cloud layer above the room. In VR mode, the user can travel between the local and cloud planes and teleport to different locations.

We also found that the best way to display packets may differ between real-time and replay modes. During early development we tested the visualization by replaying short packet captures. We found that when replaying a capture, sometimes users care about having accurate timing information, but sometimes users primarily care about event order. To understand a particular conversation or protocol, users may prefer to step through each individual packet one at a time, preserving the sequence while ignoring timing information. Understanding the benefits of these different modes and when they should be available to users remains an important open question for interaction design.

4.4.3 Layers. We decided to visualize different network layers for WiFi and Thread. For WiFi, we were interested in seeing the end-to-end IP connections, so packet lines were drawn directly between IP source and destination, bypassing the WiFi router. For the Thread network, we were interested specifically in the mesh behavior, so instead of visualizing the end-to-end IP layer connections between the sensors and the cloud, we visualized each of the link-layer hops through the mesh (Figure 11). However, in some situations it may be more appropriate to visualize the link-layer connections of a WiFi network, or the IP-layer connections of a Thread network. The ability to switch the view between these layers as desired for each network would be useful.

The semantics of a packet animation may change depending on which layer is currently being viewed. For example, Figure 12 shows an animation we created for UDP broadcast packets. UDP broadcast packets are sent to a special broadcast IP address, which causes the router to duplicate the packet and send a copy to all connected devices. This is a logical broadcast action in the IP layer, but is not a broadcast in the physical layer. The expanding shell animation acts as a good metaphor for UDP broadcast, but it may imply physical layer behavior that is not accurate.

5 XRSHARK EVALUATION

To evaluate whether XRShark can support well-defined semantics and serendipitous insights, we examine a key microbenchmark and provide several illustrative case studies.

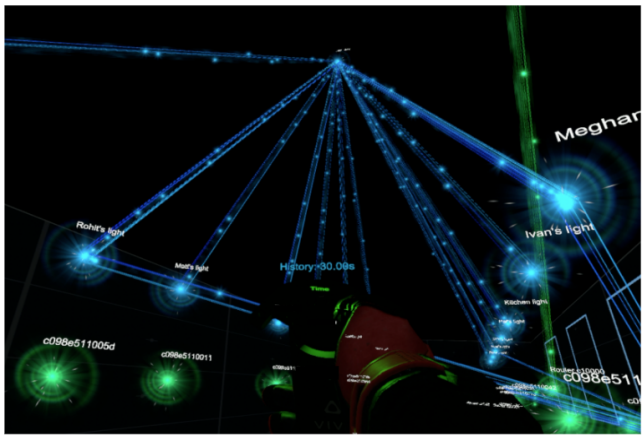


Figure 10: Controller in time mode adjusts history window. The appearance of a line indicates that a packet was sent and received in that instant. Each line also is associated with a small particle that travels from the sender to the receiver within an *n* second time window. While line *appearances* represent real-time packet activity, the balls indicate how many packets there have been in the last *n* seconds and how far apart in time they were sent. Using the controller to shrink or grow this retention window causes the packet particles to slow down or speed up their movement to complete the transit before disappearing.

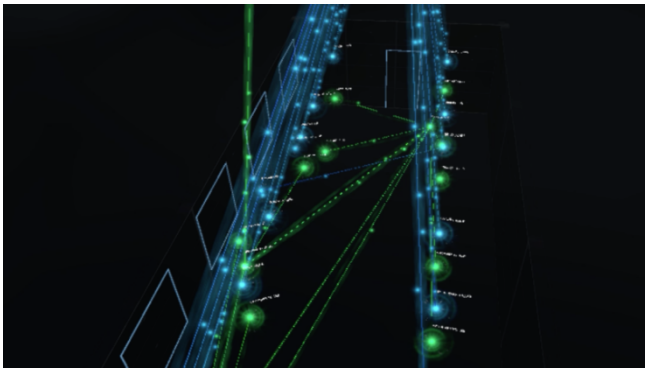


Figure 11: Looking down from the cloud. A bird's eye view of the network is useful for seeing topologies, such as the OpenThread mesh traffic shown in green. The blue lines are traffic between the LIFX cloud and the LIFX-brand smart lights in the room. As the room lights are manually dimmed from the cloud, the sudden change in brightness triggers the environmental sensors to report the new readings to a remote service.

5.1 Lip-Sync Error

It is critical for the real-time semantics of the visualization that activity be displayed shortly after it occurs in real life. Figure 13 shows an example of a streaming connection that might cause the the visualization to fall behind real time while it processes all the packets. We call this delay in real-time rendering the *lip-sync error*.

The key factor contributing to lip-sync error is not bandwidth, but rather the number of individual packet models that the visualization must animate each time step. We examine how lip-sync error grows over time at different packet rates and show the results in Figure 14. We synthesized MQTT messages representing WiFi packets between two devices in the room. We sent these messages at various fixed rates between 100 and 1000 messages per second. Upon processing each packet message, the Unity app compared the difference in elapsed time since the first packet according to the visualization time, and the elapsed time since the first packet according to the original packet timestamps. The difference between these elapsed times is the lip-sync error in that moment. While it is not yet clear what degree of lip-sync error users can tolerate in this application domain, we highlighted the one-second line as an plausible maximum threshold.

We found that any number of packets below approximately 210 packets per second stayed within the processing resources of the system and accumulated no error over time. However, higher packet rates resulted in the visualization falling further and further behind, creating a backlog that the system needed to process before catching back up to real time. Concretely, this means that watching or listening to streaming media for even a short period would effectively halt the ability to use the tool for real-time visualization as it becomes tied up processing the stream.

However, by using the same measurement technique we used for benchmarking, the visualizer can detect when lip-sync error has accumulated and take steps to mitigate it. For example, the visualization could switch to rendering a single stream instead of many individual packets, or it could begin dropping packets in an attempt to fast-forward to real time. We these techniques in mind, we remain optimistic about the feasibility of using mixed reality visualizations to view high-fidelity network traffic information in real time.

These results are also heavily influenced by the underlying hardware running the visualization application. We used a dedicated VR laptop with an i7-8750H processor with a 2.20 GHz base frequency and a 4.10 GHz high-performance frequency, 32GB of RAM, and a NVIDIA GeForce GTX 1070 GPU. Underlying performance management algorithms may explain why the difference in error between 750 and 1000 packets/second is less than the differences at lower packet rates.

5.2 Case Study: Lighting Control Application

Using XRShark's augmented reality mode, the embedded researchers were able to see the different components of their highly distributed lighting control application working together to make the lights dim or brighten. They were also able to see the WiFi lights syncing local changes with the LIFX cloud after most of the controller's incremental adjustments—an externality of the control system not originally factored in to the network overhead.

In AR, the researchers could see what stimuli, such as people walking across the room, would trigger motion or brightness packets from sensors. The sensors are thresholded to send packets in response to significant changes, and also have a refractory period

Figure 12: UDP broadcast packets. Broadcast packets appear as an expanding shell, emerging from the sender.

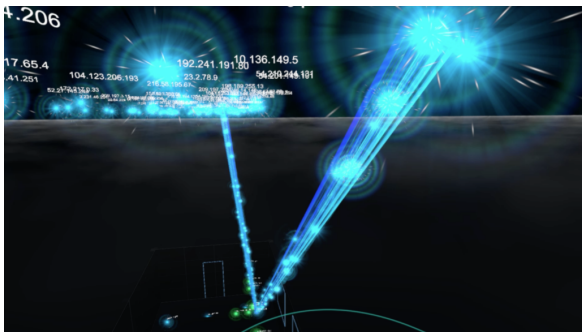


Figure 13: Netflix streaming traffic. A high-bandwidth stream like this can cause the visualization to fall behind real-time as it renders each packet in the stream.

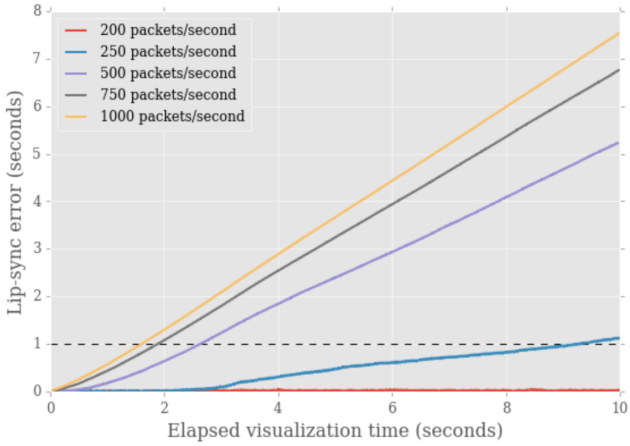


Figure 14: Lip-sync error. If the visualization cannot process packet messages quickly enough, over time the delay between when a packet was sent and when it is rendered in the visualization grows. In our implementation, lip-sync error accumulates when processing more than 210 messages per second. To combat this, if the system detects that the lip-sync error has exceeded a particular threshold (such as one second) it could take steps to catch back up to real time.

during which they will not report changes again. The researchers were able to build an intuition for the responsiveness of the sensors that would have been difficult to get from terminal readouts. They could also see that nearby desks' sensors were also occasionally triggered by a light's changes, which is easy to forget when designing controls with an idealized sensor-light pair in mind. Broken sensors also became much easier to identify in AR. If users put their hand over a broken sensor, they could see it did not send anything.

The ability to accurately visualize smartphone locations was useful because the researchers often triggered the control loop's corrective behavior by manually dimming or brightening the lights through the LIFX smartphone app. Figure 8b shows a still from the AR mode as a person walked around the office with a tracked phone. The volume of traffic was caused by opening the LIFX app, which discovers and collects status information from all of the local LIFX smart lights. The packets smoothly followed the person's phone as it moved.

5.3 Case Study: Mesh Topology Monitoring

The virtual reality mode allowed the researchers to make an interesting discovery about the Thread mesh topology. Initially, by looking down on the local network from the cloud and dimming all the lights to trigger all the sensors to send packets simultaneously, the researchers were able to view the Thread mesh topology and confirm that the routing was working as expected. At some point, a power outage occurred in the building, and the virtual reality view showed that the Thread mesh topology reformed inefficiently. Perhaps due to the mesh routers rebooting at staggered times, many sensors were bypassing nearby routers and sending packet to distant nodes. This configuration worked but was suboptimal. Using XRShark, we watched the Thread mesh topology correct itself over the course of a day.

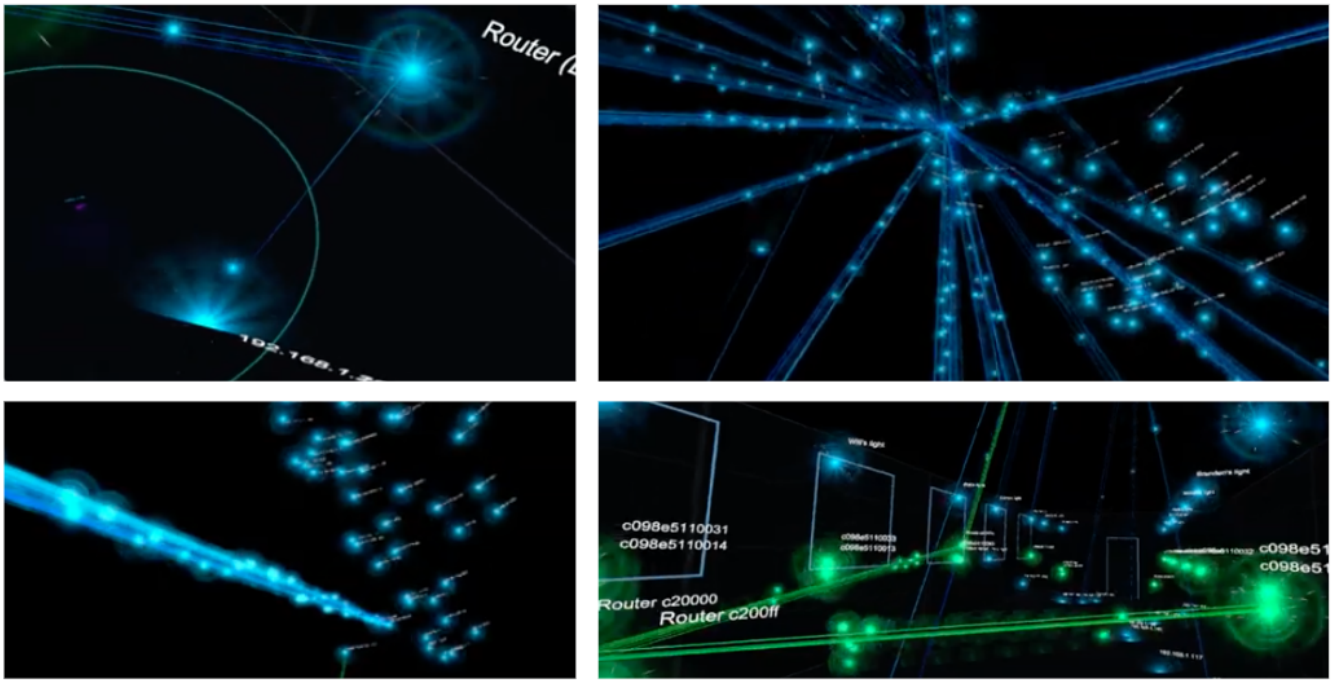


Figure 15: Example behaviors seen in XRShark. Clockwise from top-left: Address Resolution Protocol (ARP), Netflix streaming from the cloud, Thread mesh routing, and LIFX smart lights synchronizing with the LIFX cloud above the room.

5.4 IoT Device Trust

We found that Amazon's smart speaker, the Echo Dot [2], is dormant unless it is spoken to or it is streaming music. This kind of visibility into IoT device activity could potentially help build trust with users, an especially important consideration for devices that collect sensitive data with embedded microphones or cameras.

Additionally, we discovered that iOS devices with the LIFX app will periodically perform discovery and sync with LIFX lights over the local network even when the app is completely closed, likely due to integration with Homekit, Apple's smart home framework [4].

5.5 Serendipitous Discovery

While viewing traffic from Netflix, the popular video-streaming site [21], we discovered that streaming *one* TV episode produced *two* large connections to separate IP addresses. We also discovered that in our facility, Netflix content is not served by a Netflix IP address, but instead an aggregator node on the statewide regional research network (previously unknown to us) that connects educational institutions to the Internet backbone. We were pleased and surprised to learn more about our public network infrastructure.

We saw a very large beam emanating from a laptop one day, resembling streaming media, but unaccounted for. When we later went to shut down the laptop, it announced that a software update was ready.

During a demo of the system, an untrained observer pointed out a connection between a laptop and the cloud that was notable for its high frequency and regularity and asked what it was. By checking which process was bound to the outgoing port and the registered owner of the cloud IP, we determined that it was Slack, a web-based messaging platform [25], polling an instance of Amazon Web Services. We later confirmed that Slack does, in fact, run on AWS [1]. Because XRShark presents network activity so intuitively, an untrained observer was able to spot a pattern and prompt an investigation that resulted in new discoveries about our digital environment.

6 OPEN CHALLENGES

Further research must be conducted to understand the trade-offs in modification effort, network overhead, and fidelity when it comes to instrumenting different kinds of network topologies, including peer-to-peer technologies like Bluetooth Low Energy. How well do these approaches scale? How robust are they to node failures or high latency transmissions? A systematic characterization of the infrastructure design space for mixed reality introspection would be highly informative. In addition, a number of open challenges in this area remain.

6.1 Active Probing

Multiple testers of XRShark expressed an interest in crafting and injecting packets into live networks while they are in the headset. This feature has interesting implications for the design of the network instrumentation infrastructure. Should packet injection capabilities be coupled with monitoring services? When and where should

additional services or even special-purposes devices be added to a network for packet injection? How can we do so safely?

6.2 Physical Layer

Another feature requested by multiple testers is visualizing real-time information about the RF spectrum, particularly for informing gateway placement and understanding wireless coverage and performance. Physical layer information is also critical for understanding packet loss and interference. However, collecting this information in a scalable way is a challenge.

6.3 Security and Privacy

The question of how to address the security and privacy concerns raised by collecting and exposing a comprehensive picture of local network activity (and potentially supporting packet injection) remains a major unresolved problem. Implemented insecurely, each component of the introspection system could potentially open the network to observation or even active attack. Encryption and a thoughtful access control scheme are absolute requirements. One potential solution is to use a high-performance publish-subscribe system designed around a decentralized access control framework like WAVE [3], which would allow users to grant, revoke, and audit data access robustly.

6.4 Provenance

A fundamental characteristic of networks is that they look different depending on what device you view them from. A major interaction challenge is creating intuitive ways of indicating the provenance of the data, and helping users understand that the view only represents the perspective of the capture device(s). In some applications, however, the fact that the data represents the viewpoint of a network-based observer can be a feature. For example, the view looking down on the space from the cloud could show what the local network looks like to the public internet, providing a quick picture of the attack surface. In electronic warfare it would helpful to see how the opponent's systems view your RF behavior. In consumer networks, if users could see their unencrypted browsing activity flying through the air, and understood that any WiFi device on that same network could see it as well, it might improve general understanding of when and why to use HTTPS and VPNs for protection. Questions around how to collect, represent, and visualize provenance and network perspective information in an intuitive way are a source of inspiration for additional research.

7 CONCLUSION

With the promise of localization and mixed reality technologies finally coming to fruit, mixed reality network introspection is becoming a feasible approach to an intuitive, discovery-oriented, and spatially aware understanding of networks. Our experiences developing and using XRShark, a real-time network visualizer, has excited us by enabling new classes of insights, while also highlighting future research challenges. We hope community research efforts in this direction will someday bring the dream of perceptual awareness of our immediate wireless environment to (mixed) reality — rendering the invisible visible.

ACKNOWLEDGMENTS

The authors would like to thank Neal Jackson for his help instrumenting the OpenThread sensors and routers to enable network monitoring; Gabe Fierro for authoring some of the WiFi packet sniffing code; Will Huang and Noah Klugman for editing paper drafts; and the many demo users who gave amazing feedback. This work was supported in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

REFERENCES

[1] Amazon. 2015. Slack Case Study | Amazon Web Services (AWS). https://aws.amazon.com/solutions/case-studies/slack/.
[2] Amazon. 2020. Amazon Echo & Alexa Devices. https://amazon.com/Amazon-Echo-And-Alexa-Devices/b?node=9818047011.
[3] M. Andersen, S. Kumar, M. AbdelBaky, G. Fierro, J. Kolb, H. Kim, D. Culler, and R. Popa. 2019. {WAVE}: A Decentralized Authorization Framework with Transitive Delegation. In *28th USENIX Security Symposium (USENIX Security 19)*. USENIX Association, Santa Clara, CA, 1375–1392.
[4] Apple. 2019. HomeKit - Apple Developer. https://developer.apple.com/homekit.
[5] Apple. 2019. Ultra Wideband | iOS and iPadOS - Feature Availability. https://www.apple.com/ios/feature-availability/#ultra-wideband.
[6] BadVR. 2020. SeeSignal. https://www.badvr.com/seesignal.html.
[7] Eric A Bier, Maureen C Stone, Ken Pier, William Buxton, and Tony D DeRose. 1993. Toolglass and magic lenses: the see-through interface. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. 73–80.
[8] E. Biersack, Q. Jacquemart, F. Fischer, J. Fuchs, O. Thonnard, G. Theodoridis, D. Tzovaras, and P. Vervier. 2012. Visual analytics for BGP monitoring and prefix hijacking identification. *IEEE Network* 26, 6 (November 2012), 33–39. https://doi.org/10.1109/MNET.2012.6375891
[9] Leonard D Brown and Hong Hua. 2006. Magic lenses for augmented virtual environments. *IEEE Computer Graphics and Applications* 26, 4 (2006), 64–73.
[10] Decawave. 2020. MDEK1001 Development Kit - Decawave. https://www.decawave.com/product/mdek1001-deployment-kit.
[11] G. Dublon, L. S. Pardue, B. Mayton, N. Swartz, N. Joliat, P. Hurst, and J. A. Paradiso. 2011. DoppelLab: Tools for exploring and harnessing multimodal sensor network data. In *SENSORS, 2011 IEEE*. Institute of Electrical and Electronics Engineers, New York, NY, USA, 1612–1615.
[12] Rodrigo Fonseca, George Porter, Randy H Katz, and Scott Shenker. 2007. X-trace: A pervasive network tracing framework. In *4th {USENIX} Symposium on Networked Systems Design & Implementation ({NSDI} 07)*.
[13] L. Harrison and A. Lu. 2012. The future of security visualization: Lessons from network visualization. *IEEE Network* 26, 6 (November 2012), 6–11. https://doi.org/10.1109/MNET.2012.6375887
[14] V. Heun, S. Kasahara, and P. Maes. 2013. Smarter Objects: Using AR Technology to Program Physical Objects and Their Interactions. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems* (Paris, France) *(CHI EA '13)*. ACM, New York, NY, USA, 961–966. https://doi.org/10.1145/2468356.2468528
[15] Benjamin Kempke, Pat Pannuto, and Prabal Dutta. 2016. Harmonium: Asymmetric, bandstitched UWB for fast, accurate, and robust indoor localization. In *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 1–12.
[16] Larry Dignan. 2019. Splunk aims to use augmented reality to monitor server racks, equipment to bring data to multiple screens. https://www.zdnet.com/article/splunk-aims-to-use-augmented-reality-to-monitor-server-racks-equipment-to-bring-data-to-multiple-screens.
[17] Patrick Lazik, Niranjini Rajagopal, Oliver Shih, Bruno Sinopoli, and Anthony Rowe. 2015. ALPS: A bluetooth and ultrasound platform for mapping and localization. In *Proceedings of the 13th ACM conference on embedded networked sensor systems*. 73–84.
[18] LIFX. 2020. Wi-Fi enabled LED smart lights - LIFX. https://www.lifx.com.
[19] W. McGrath, J. Warner, M. Karchemsky, A. Head, D. Drew, and B. Hartmann. 2018. WiFröst: Bridging the Information Gap for Debugging of Networked Embedded Systems. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology* (Berlin, Germany) *(UIST '18)*. Association for Computing Machinery, New York, NY, USA, 447–455. https://doi.org/10.1145/3242587.3242668
[20] MQTT. 2019. MQTT. http://mqtt.org.
[21] Netflix. 2020. Watch TV Shows Online, Watch Movies Online. https://www.netflix.com.
[22] OpenWrt Project. 2020. Welcome to the OpenWrt Project. https://openwrt.org.
[23] Yongtae Park, Sangki Yun, and Kyu-Han Kim. 2019. When IoT met augmented reality: Visualizing the source of the wireless signal in AR view. In *Proceedings*

of the 17th Annual International Conference on Mobile Systems, Applications, and Services. 117–129.
[24] Quuppa. 2020. Real-Time Locating System (RTLS) | powered by Quuppa. https://quuppa.com.
[25] Slack Technologies. 2020. Where work happens | Slack. https://slack.com.
[26] Martin Striegel, Carsten Rolfes, Johann Heyszl, Fabian Helfert, Maximilian Hornung, and Georg Sigl. 2019. EyeSec: A Retrofittable Augmented Reality Tool for Troubleshooting Wireless Sensor Networks in the Field. *arXiv preprint arXiv:1907.12364* (2019).
[27] The Tcpdump Group. 2019. TCPDUMP/LIBPCAP public repository. https://www.tcpdump.org.
[28] Unity. 2020. Unity Real-Time Development Platform | 3D, 2D, VR & AR Visualizations. https://unity.com.
[29] Megan Wachs, Jung Il Choi, Jung Woo Lee, Kannan Srinivasan, Zhe Chen, Mayank Jain, and Philip Levis. 2007. Visibility: A new metric for protocol design. In *Proceedings of the 5th international conference on Embedded networked sensor systems*. 73–86.
[30] Wireshark Foundation. 2020. Wireshark: Go Deep. https://www.wireshark.org.