2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops): Demos

2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops) | 978-1-6654-5381-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/PERCOMWORKSHOPS56833.2023.10150343

Demo: VELOX: Enhancing P2P Real-Time Communication in Browsers

Justin Parsons*, Jackson Winslow, Lewis Tseng
{parsonjj, winslojc, lewis.tseng}@bc.edu
Boston College
Boston, MA

***Abstract*—Peer-to-peer (P2P) real-time communication is a fundamental building block for modern pervasive systems and applications, to name a few, mobile edge clouds, device-to-device coordination, and intelligent transportation. This paper introduces Velox, a suite of tools to enhance P2P real-time communication in browsers. It consists of a customized JavaScript compiler, a control room for managing peer connections and tracking resources using the WebSocket protocol, and client-side scripts for maintaining P2P communication channels using WebRTC and fetching necessary resources and contents.**

Two design priorities of our design are extensibility and ease of adoption. First, our tools are modular so that each component can be easily replaced and extended without touching other parts. Hence, others can build on top of Velox by only modifying the critical component(s). Second, we choose to develop our tools such that they work directly in web browsers. Browsers run smoothly on most personal computing devices; hence, a direct support for browsers will make the adoption and development simpler.

Finally, we discuss our plan to demonstrate a P2P caching application using Velox to showcase the effectiveness and functionalities of our tools. In particular, a client can fetch resources cached at other peers.

***Index Terms*—P2P, WebRTC, JavaScript, WebSocket**

I. INTRODUCTION

A. Motivation: Why P2P Real-Time Communication?

Imagine a scenario in which a group of friends visit a restaurant with an online menu and attempt to access the menu simultaneously. Depending on how and where the menu is hosted and how request messages are routed on the web, it is entirely possible that one person in the group has the menu loaded on his or her smartphone, while it takes a noticeable amount of time for others' smartphones to load it. This is mainly because of the complexity of modern networking and storage/caching infrastructure, which can result in poor user experience. Peer-to-peer (P2P) real-time communication can fix this issue by allowing the person who fetches the menu first to share it with others. This observation motivated us to build tools to support P2P communication in browsers.

B. Applications in Pervasive Computing

International Data Corporation (IDC) predicts that by 2025 there will be 55.7B connected devices, together generating 80ZB amount of data [1]. Ericsson predicts that the total global mobile data traffic (excluding traffic generated by Fixed Wireless Access) will grow to 325 EB[1] per month in 2028 [2]. To better support emerging pervasive systems that are communication-intensive (such as ML, AR, VR and MR applications), we will need to increase the capacities of current infrastructure for networking, storage, and computation.

P2P communication is an effective approach in addressing the challenges. First, a computing device can rely on P2P channels to offload computing and communication to a "surrogate" or "proxy" [3]. Second, P2P communication can be used to reduce clouds' workload, as the central servers or network services do not need to be involved in every communication step, nor handle all of the data. Third, as mentioned earlier, P2P communication can also increase user experience by allowing peers to share data and resources directly.

There are numerous pervasive applications that could benefit from real-time P2P communication. For example, intelligent video acceleration and Internet-of-Things gateway both utilize P2P communication in the mobile edge clouds paradigm [4], and vehicle-to-vehicle communication is essential for intelligent transportation systems [5]. In short, providing tools to simplify the development and adoption of P2P real-time communication has large-scale impacts.

C. Contributions

Towards our goal, we introduce VELOX, a suite of tools for implementing P2P real-time communication. We choose to develop our tools such that they work directly inside web browsers. That is, no extra app needs to be loaded on the clients. Browsers work smoothly on most personal computing devices; hence, a direct support for browsers will make the adoption and development simpler. Furthermore, our P2P communication is based on WebRTC data channel [6], which uses DTLS-SRTP to ensure end-to-end encryption.[2] For the demonstration, we implement a P2P caching application to show that caching effectively reduces the workload at the web server. Section III presents our plan.

Due to space constraints, this paper only presents the core features and design of VELOX. We believe it enables us to build emerging pervasive systems. For one of our projects, we are using VELOX to build a peer-assisted content delivery network (CDN). VELOX allows us to focus on the innovation

*First two authors contribute equally. The authors are partially funded by Schiller SIGEC grant.

[1] 1 EB (Exabyte) = 1,000 PB = 1,000,000 TB.

[2] DTLS stands for Datagram Transport Layer Security, whereas SRTP stands for Secure Real-time Transport Protocol.

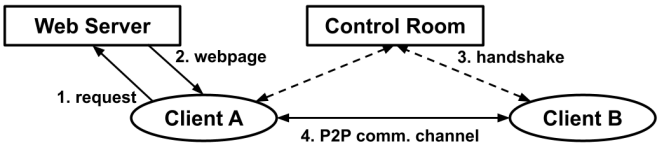

Fig. 1: **Architecture of VELOX.** Dashed lines denote communication among client A, client B, and Control Room. Solid lines denote communication between two respective entities.

without worrying about the engineering behind computer networking and P2P communication.

D. Related Work

WebRTC is an open-source project initiated by Google, which provides the foundation for real-time communication in browsers. While powerful and versatile, WebRTC is non-trivial to adopt and use, especially for developing a system with large-scale peer interactions. PeerJS [7] and Peer5 [8] are two recent open-source projects that aim to solve this issue. PeerJS simplifies the peer connection part, whereas Peer5 builds a CDN for streaming applications. However, as opposed to PeerJS, VELOX is optimized for large interconnected peer-networks so that more computation is offset to clients. Peer5 handles large interconnected networks but is not adaptable for usage other than in video CDNs. In comparison, VELOX is designed with a wide range of pervasive systems in mind. As it will become clear later, VELOX supports an easier approach to keep track of connections and peers, as well as message handling. This allows developers to take care of peer interactions with ease.

II. DESIGN AND IMPLEMENTATION

VELOX consists of the following key components: clients, a control room, and a web server. Figure 1 presents the high-level architecture and communication flow. For brevity, only two clients are depicted, and we assume that each client may serve as a peer; thus, we use the two terms interchangeably. However, we stress that VELOX is designed to support a large number of clients, and it is possible to separate clients and peers in our design.

A. Architecture

- **Web Server**: It serves the content of a website, which includes HTML files, JavaScript files, and other media contents (e.g., images or videos). The Web Server is what hosts the necessary scripts for clients to use VELOX. These scripts can be fetched remotely by clients.
- **Control Room**: It hosts several services: (i) *Connection management*: it maintains and manages the connection with peers using the WebSocket protocol; (ii) *Signaling*: it acts as a WebRTC signaling server so that clients can build a WebRTC-based P2P communication channel with other clients; and (iii) *Service discovery*: it provides a unique identifier to clients and works as a centralized source to record and share peer metadata.
- **Clients**: In our design, clients should have a minimum requirement to participate as peers (which is to be able to communicate with others through a JavaScript run-time environment that supports WebRTC). Each client requires at least two client-side JavaScript files which would typically be fetched from the Web Server. This would consist of the script associated with the VELOX library, and another script that imports the library's features for browser-based use.
  The VELOX library script works with the Control Room to coordinate a client's connection to other peers in the peer network. This is done when calling for the creation of a VELOX object in the second script. This simple behavior removes the complexity of establishing such connections manually from the developer. The second client script utilizes functionality from VELOX to then communicate on this network. In the case for our testing, we created browser-based peer-to-peer resource caching. This browser-based peer-to-peer resource caching is accomplished using a smart fetching protocol. It first attempts to load the resource from a locally stored Binary Large Object (Blob) URL. If this fails, the VELOX object is called upon to access the peer network and send messages across data channels requesting the given resource. While this is likely to succeed and provide users with a fast resource-fetching experience, a catch-all for any errors is implemented which requests resources from the centralized web server as a last resort.

B. Communication Flow

We present the steps that client A needs to take in order to fetch a webpage with some contents cached at client B.

1) Client A requests a specific webpage from the Web Server as it normally does.
2) Web server sends the base HTML of the webpage (which contains special tags that force the browser to fetch and run the client side scripts) to client A.
3) Client A contacts the Control Room to initialize the *handshake protocol* between client A and client B. This step may involve multiple communication steps so that the two clients agree to connect to each other and share details on their network topologies. This part involves three parties: client A, client B, and the Control Room.
4) The WebRTC-based P2P communication channel between client A and client B is established. The two clients can then use the channel to fetch and exchange necessary resources, as specified by the webpage.

Handshake: There are several alternative designs for the handshake protocol, depending on the application needs and client constraints. The protocol is for clients to know which peers to connect to and how to connect to them (i.e., WebRTC signaling). The handshake is built on top of SDP (session description protocol). Exact details are omitted due to space constraint.

C. Design Principles

Two principles that guide our design are extensibility and ease of adoption. To make it extensible, our tools are designed to be modular: (i) each component can be easily replaced and extended without touching other parts, and (ii) communication steps (e.g., steps 1, 2, 3) can be replaced as long as they conform with the interface. As a result, others can build on top of VELOX by only modifying the necessary component(s).

Second, to reduce the effort of adopting VELOX, we choose to develop our tools such that they work directly in web browsers. Right now, we have only tested VELOX with the exchange of static files, but the same design can be extended to support modern JavaScript applications that provide visualization and interaction. P2P communication channel (step 4) is a WebRTC-based channel. In particular, it enables in-browser applications to stream audio and/or video, and exchange arbitrary data without requiring a third party. Since WebRTC is an open standard, one can easily adopt it using an existing framework that runs on top of WebRTC.

D. Implementation

For webpage rendering, we need to implement a layer to govern where a resource is fetched from in client-side scripts. We call our approach the "blob-oriented model." Blob is a JavaScript data type used to represent large binary data. In our approach, blob objects can be encoded as URLs which are referenceable in HTML and JavaScript. This property of the binary large object makes it ideal for creating a layer below content rendered on the DOM (Document Object Model). We designed a compiler to prevent the browser from automatically fetching resources embedded in HTML so that the DOM will point to a blob URL that is generated by a client-side script.

We developed any backend functionality in Go[3] due to its extensive libraries and ease of use without the need to compromise on control and speed. The client-side scripts are implemented in JavaScript, as this is naturally the best language for supporting in-browser rendering and interaction. In particular, one of the scripts controls the request waterfall (for rendering an HTML page) so that it can fetch necessary resources and contents from peers, without contacting the Web Server for all referenced resources. Our implementation is publicly available at https://github.com/parleon/velox-songbird.

E. Future Work

We implement all the necessary framework for demonstrating a minimum viable product (MVP), as detailed in the next section. There are several tasks that we plan to add to VELOX: (i) checksums for extra security, (ii) support of data chunking and fetching, (iii) more efficient fetching (e.g., smart prefetching), and (iv) decentralizing control room functionality.

III. SHOWCASE: P2P CACHING FOR PEER-ASSISTED CDN

We will build a website that runs with both the Web Server and the Control Room from Figure 1. The website is publicly available so that visitors can directly use their own personal device (with Internet access) to connect to our website.

Our website will include a page that has a toggle for disabling P2P caching so visitors can observe its effect in real-time. We will also include a chatroom to display server-less communication between devices. Moreover, we will display visualizations of overall bandwidth to help visitors observe that the amount of upstream/downstream data being distributed by the Web Server is less than the cumulative data being downloaded by clients.

Recall that the main reason that we develop VELOX for browsers is for ease of adoption; hence, we believe most personal devices with browsers (that support JavaScript) can have the same experience. In particular, VELOX is implemented in a way that services such as P2P caching work without clients needing to do anything else that they would not normally do (for accessing a webpage). That being said, we developed and tested the project on Chrome and Firefox.

Requirement: To simplify the process, we will ask all the visitors to connect to the venue's WiFi to use P2P caching. While VELOX can work around most network address translation or firewall issues, the user experience may possibly be affected.

IV. CONCLUSION

The main take-away of our demo is that even with recent advancement in communication and cloud tools, it is still surprisingly difficult to implement direct P2P communication between devices. A direct support in the browser is even more challenging. This paper introduces VELOX as a standalone service, which enhances the P2P real-time communication between browsers on different devices. We believe it will be useful for many pervasive applications.

REFERENCES

[1] IDC. Future of industry ecosystems: Shared data and insights. https://blogs.idc.com/2021/01/06/future-of-industry-ecosystems-shared-data-and-insights/.
[2] Ericsson. Mobile data traffic outlook. https://www.ericsson.com/en/reports-and-papers/mobility-report/dataforecasts/mobile-traffic-forecast.
[3] Mahadev Satyanarayanan. Pervasive computing: vision and challenges. *IEEE Wirel. Commun.*, 8(4):10–17, 2001.
[4] Min Qin, Li Chen, Nan Zhao, Yunfei Chen, F. Richard Yu, and Guo Wei. Computing and relaying: Utilizing mobile edge computing for p2p communications. *IEEE Transactions on Vehicular Technology*, 69(2):1582–1594, 2020.
[5] Marco Meucci, Marco Seminara, Tassadaq Nawaz, Stefano Caputo, Lorenzo Mucchi, and Jacopo Catani. Bidirectional vehicle-to-vehicle communication system based on vlc: Outdoor tests and performance analysis. *IEEE Transactions on Intelligent Transportation Systems*, 23(8):11465–11475, 2022.
[6] WebRTC. Webrtc. https://www.webrtc.org/.
[7] PeerJS. Peerjs. https://www.peerjs.com/.
[8] Peer5. Peer5. https://www.peer5.com/.

[3]The Go Programming Language go.dev