

A Privacy-Focused Cross-Platform Peer-to-Peer Communication and Synchronization Framework Using WebRTC and WebSocket

Aakash Poudel

Department of Software Engineering
Nepal College of Information Technology
Lalitpur, Nepal
aakash.211604@ncit.edu.np

Aradhya Parajuli

Department of Software Engineering
Nepal College of Information Technology
Lalitpur, Nepal
aradhya.211616@ncit.edu.np

Bishwas Khanal

Department of Software Engineering
Nepal College of Information Technology
Lalitpur, Nepal
bishwas.211625@ncit.edu.np

Niroj Danai

Department of Software Engineering
Nepal College of Information Technology
Lalitpur, Nepal
niroj.211632@ncit.edu.np

Rudra Nepal

Department of Software Engineering
Nepal College of Information Technology
Lalitpur, Nepal
rudra.nepal@ncit.edu.np

Abstract—Most contemporary file synchronization systems rely on centralized cloud infrastructure, which raises concerns regarding data privacy, service availability, dependence on external infrastructure, and user control over personal data. As users increasingly operate across multiple devices, providing secure cross-platform synchronization without persistent centralized storage remains a significant challenge. This paper presents PocketBridge, a privacy-focused cross-platform peer-to-peer communication and file synchronization framework that enables communication and synchronization between authenticated personal devices. The proposed framework adopts a layered architecture that separates connection coordination from application data transfer. WebSocket is used for device discovery, temporary pairing, and signaling, while WebRTC DataChannels are used for peer-to-peer data transfer. Temporary pairing credentials and cryptographic device identities provide controlled device authorization, while application-level encryption protects synchronization content when supporting infrastructure is involved. The framework operates within a web browser, enabling synchronization across heterogeneous desktop and mobile platforms without requiring separate platform-specific applications. The implementation was evaluated through integrity and security tests, cryptographic processing benchmarks, end-to-end file-transfer experiments, synchronization reliability measurements, and cross-platform browser compatibility tests. The results demonstrate the feasibility of combining browser-based peer-to-peer communication, temporary device authorization, and application-level protection for privacy-focused cross-platform synchronization.

Keywords —peer-to-peer communication, file synchronization, WebRTC, WebSocket, cross-platform systems, privacy-preserving systems, decentralized communication, browser-based applications.

I. INTRODUCTION

The growing use of smartphones, tablets, laptops, and desktop computers has increased the need for reliable synchronization and access to data across multiple devices. Conventional

synchronization systems depend heavily on centralized cloud infrastructure to store and share user data. These systems provide easy access across devices but raise issues of data privacy, service reliability, continuous Internet connectivity and user control over the synchronized data. To address these concerns, PocketBridge utilizes a cross-platform peer-to-peer architecture that allows devices to communicate and synchronize data without the need for persistent centralized storage. The framework combines WebRTC-based peer-to-peer communication [13], WebSocket-based signaling and session coordination, and authenticated device pairing and security to protect synchronization data. The system is implemented as a browser-based self-hosted framework for personal multi-device synchronization across heterogeneous platforms.

A. Motivation

The primary motivation for PocketBridge is the increasing dependence on centralized synchronization services and the privacy and control limitations associated with storing personal data on third-party infrastructure. Cloud-based systems can simplify synchronization between devices, but require users to trust the service provider with their data and depend on the availability of the provider's infrastructure. This dependency can be reduced by applying a peer-to-peer approach where devices are able to directly communicate and have more control over synchronized data. However, direct device-to-device transfer alone does not provide sufficient security. Devices must be authenticated before they are allowed to synchronize, communication sessions must be secured against unauthorized access and replay, and synchronization content should be kept confidential even when supporting infrastructure is involved. Thus, PocketBridge is motivated by the need for a system that allows for cross-platform synchronization,

peer-to-peer communication, trusted device authentication, and privacy from the coordinating infrastructure. The browser-based approach also provides a common runtime and security API across supported platforms while reducing the need for separate platform-specific implementations.

B. Problem Statement

Existing synchronization approaches do not provide an effective combination of cross-platform usability, user control over data, and the security requirements of decentralized communication. While centralized solutions require the users to depend on third-party infrastructure, pure peer-to-peer solutions suffer from challenges regarding device discovery, authentication, session establishment, network traversal and heterogeneous platform support. Additionally, if a server is used for signaling or connectivity fallback, synchronized information could be exposed to the infrastructure if application-level protection is not provided. Thus, the problem addressed in this work is to design and implement a privacy-focused, cross-platform synchronization framework that enables authorized personal devices to securely communicate, while preventing the coordinating infrastructure from recovering plaintext synchronization contents. The framework is designed to support device identity, ephemeral pairing, authenticated session establishment, protected data transfer, integrity verification, replay protection, and reliable data transfer under the evaluated network conditions. This implementation is currently limited to personal multi-device synchronization, not multi-user group collaboration.

C. Objectives

- To design and develop a secure, cross-platform peer-to-peer device synchronization system that enables direct device-to-device communication using WebSocket for signaling and WebRTC for low-latency, encrypted data transfer, supporting reliable file sharing and data consistency.
- To develop a privacy-focused synchronization mechanism incorporating secure device authentication and temporary pairing, thereby reducing dependence on centralized cloud storage while enhancing user privacy, data control, and cross-platform interoperability.

II. LITERATURE REVIEW

The increasing demand for cross-device data access has led to research into decentralized storage and synchronization architectures that reduce dependence on centralized cloud infrastructures. Peng et al. [2] proposed a peer-to-peer file storage and sharing system based on consortium blockchain, using encrypted file fragments, identity authentication, and erasure coding to improve integrity and fault tolerance. Zhan et al. [3] presented a file-sharing framework supporting bidirectional folder synchronization, version control, and access permissions. Rozen Berg et al. [9] investigated WebRTC-based decentralized communication with minimal latency, demonstrating the feasibility of direct peer communication

without relying on conventional centralized communication paths. Although these approaches demonstrate the benefits of decentralized architectures, they either introduce additional infrastructure and processing requirements or focus primarily on communication and storage rather than privacy-focused synchronization across heterogeneous personal devices.

WebRTC provides a framework for real-time peer-to-peer communication between web browsers, including direct data transfer and mechanisms for NAT traversal and secure communication [13]. Recent research has also explored WebRTC and WebSocket as complementary technologies for browser-based peer-to-peer communication. Nachiappan et al. [10] demonstrated a WebRTC and WebSocket architecture for decentralized real-time file sharing, where WebSocket is used for coordination and WebRTC provides direct peer communication. Shaikh et al. [11] proposed DropBridge, which uses a lightweight signaling server to establish peer connections before transferring files through WebRTC DataChannels. Geetha Rani et al. [8] investigated peer-to-peer file streaming using WebSockets without relying on intermediate storage. Ishola and Mailler [6] extended the WebRTC-WebSocket model to collaborative applications by combining peer communication with WebSocket-based signaling and adaptive state synchronization. These studies establish WebRTC and WebSocket as practical technologies for browser-based P2P communication; however, their primary focus is on file transfer, streaming, or real-time collaboration rather than secure, persistent synchronization between a user's heterogeneous personal devices.

Security and privacy have received increasing attention in decentralized communication research. Arnes and Karlsen [1] proposed Swirlwave, a cloud-independent middleware for smartphone communication that supports direct communication and reconnectable channels across changing network conditions. Jindal et al. [7] presented P2P-SFS, incorporating authentication and encryption mechanisms to protect peer-to-peer file sharing against unauthorized access and man-in-the-middle attacks. Urmila and Shubham [4] developed a cross-platform P2P file-sharing framework using hybrid communication and cryptographic mechanisms to secure file exchange across different devices. Jain et al. [12] also investigated encrypted socket-based peer communication for decentralized file sharing. While these approaches demonstrate the importance of authentication and encryption, they primarily address secure communication or file exchange and provide limited treatment of temporary device authorization, cryptographically established synchronization sessions, replay protection, and protection of synchronization content when coordination infrastructure remains involved.

Another important challenge concerns the role of supporting infrastructure in P2P systems. Although peer-to-peer data transfer can eliminate the need for centralized storage, practical systems frequently require signaling servers, peer-discovery services, or relay mechanisms for connection establishment and network traversal. Mahmoud and Abozariba [5], in their systematic review of WebRTC applications beyond multimedia communication, identified interoperability,

scalability, standardization, and distributed communication management as important challenges in applying WebRTC to broader application domains. Similarly, Ishola and Mailler [6] highlighted challenges associated with maintaining synchronization and communication across distributed peers. These findings indicate that a practical P2P synchronization framework should not only provide direct communication but should also clearly separate the control plane from the data plane and protect application data when supporting infrastructure is required.

A. Research Gap

Although existing studies have made significant progress in decentralized communication, WebRTC, WebSocket, and secure peer-to-peer networking, several challenges remain unresolved. Existing solutions primarily address file transfer, collaborative applications, browser-based streaming, or distributed storage, while only limited work combines these capabilities into a browser-native framework for secure personal multi-device synchronization. Furthermore, practical P2P architectures often rely on centralized signaling, discovery, or relay services, creating additional privacy and trust considerations. However, comparatively limited attention has been given to integrating temporary device pairing, authenticated synchronization sessions, and application-level protection against supporting infrastructure within a lightweight browser-native framework. These limitations highlight the need for a lightweight, privacy-focused framework that combines cross-platform browser-based synchronization with secure device authorization and peer-to-peer communication while minimizing dependence on centralized cloud storage. Motivated by these challenges, this paper proposes PocketBridge, which combines WebSocket-based signaling with WebRTC peer communication and incorporates temporary pairing and authenticated communication to provide privacy-focused device synchronization.

III. CONTRIBUTIONS

The main contribution of this work is the design and implementation of PocketBridge as a browser-based, privacy-focused framework for cross-platform personal device synchronization. The framework separates connection coordination from application data transfer by using WebSocket for signaling and WebRTC DataChannels for peer-to-peer communication, while temporary pairing and cryptographic device identities provide controlled device authorization. PocketBridge further combines Ed25519 authentication, ephemeral P-256 ECDH, HKDF-SHA256 key derivation, and AES-256-GCM application-level encryption to protect synchronization data even when supporting signaling or relay infrastructure is involved. In addition, the framework provides both direct WebRTC and encrypted relay-based communication paths and was evaluated through security, integrity, cryptographic processing, end-to-end transfer, reliability, and cross-platform compatibility experiments.

IV. METHODOLOGY

This section describes the proposed PocketBridge framework, including its system architecture, communication model, device pairing mechanism, peer connection establishment, file synchronization workflow, and security considerations. The methodology illustrates how secure and decentralized cross-platform file synchronization is achieved using WebSocket and WebRTC.

A. System Architecture

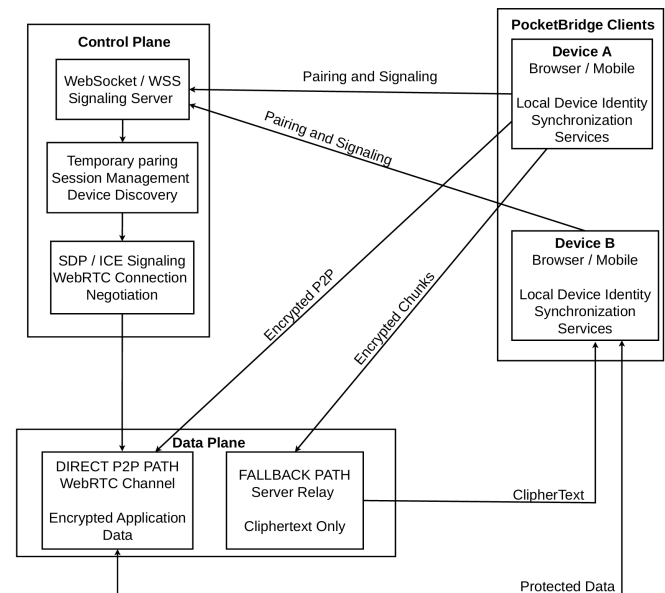


Fig. 1. System Architecture

PocketBridge adopts a server-assisted peer-to-peer architecture that separates connection coordination from application data transfer. The architecture is defined in two main communication planes, a control plane and a data plane. The control plane manages the device discovery, temporary pairing, session management and peer connection negotiation, while the data plane is responsible for the transfer of protected application data between authorized devices. Such separation allows the signaling infrastructure to coordinate the communication without having access to the plaintext content that is being synchronized.

The control plane is implemented via WebSocket communication between PocketBridge clients and the signaling server [14]. During session establishment, the server manages temporary pairing information and coordinates the exchange of signaling information required for WebRTC connection establishment. The signaling layer thus provides connection coordination rather than a centralized store for synchronized files and application content. Temporary session information is only stored for pairing and connection management purposes.

The data plane allows authenticated devices to communicate after a peer connection is established. If direct peer connectivity is available, application data is transferred over

an encrypted WebRTC DataChannel between the participating devices. If direct peer connectivity is not possible, PocketBridge also supports an encrypted relay path. In this case, the application data is encrypted before being sent via the supporting infrastructure and the relay can process the ciphertext without having to access the plaintext application content.

The framework is designed for synchronization of heterogeneous devices that belong to the same user. Device identity is based on locally generated cryptographic credentials and does not require a permanent account issued by the provider. Temporary pairing determines which devices are allowed to participate in a synchronization session. This architecture reduces reliance on centralized cloud storage while preserving the signaling infrastructure for practical peer discovery and connection establishment. This separation of control coordination and protected data transfer ultimately lays the foundation for the privacy-preserving cross-platform synchronization model of PocketBridge.

B. Hybrid Communication Model

PocketBridge uses a hybrid communication model that separates connection coordination from application data transfer. WebSocket: This is the control channel between clients and the signaling server for device discovery, temporary session management, and exchange of WebRTC signaling information [13]. This includes SDP offers and answers and ICE candidates required for the peer connection establishment [14]. The data plane includes the application data after negotiation. When direct connectivity is available, application data is transferred over an encrypted WebRTC DataChannel [15], [16], whose underlying WebRTC security mechanisms provide confidentiality and integrity for the transport [17]. In the absence of direct connectivity, PocketBridge also supports an encrypted relay path in which application data is protected before crossing the supporting infrastructure. Thus the signaling server coordinates the setup of the connection but is not the primary storage or plaintext data-transfer layer. The separation enables PocketBridge to coordinate connections practically while protecting the application data when synchronizing across devices in heterogeneous network environments.

C. Device Pairing and Authentication

PocketBridge enables devices using temporary pairing and locally generated cryptographic identities, removing the need for permanent user accounts issued by a provider. Each device group has an Ed25519 public key that is used as the `user_id` and serves as the identity anchor for device authorization.

If a device wants to start synchronization, it creates a temporary pairing session that can be shared using a six-digit pairing code or a QR code. The signaling server validates the pairing information and its expiration before allowing the receiving device to join the session. Pairing credentials are time-limited and single-use, reducing the risk of unauthorized reuse.

After successful pairing, the devices involved can use their cryptographic identities to establish an authenticated commu-

nication session. This mechanism enables controlled device enrollment with low dependency on persistent accounts and centralized identity management.

D. Peer Connection Establishment

The signaling layer then establishes the communication session once the device has been successfully paired. Both participating clients connect to the signaling server over WebSocket and exchange SDP offer, SDP answer and ICE candidates [13], [16]. This negotiation is orchestrated by the signaling server. This server does not carry the application payload.

The exchanged SDP and ICE information enables the WebRTC peers to identify and establish a suitable communication path [13] [16]. Peers establish an encrypted WebRTC DataChannel to send application data if direct connectivity is possible. In the absence of direct connectivity, the framework can set up an encrypted relay path while maintaining confidentiality at the application level.

The resulting connection thus separates signaling from application communication. After the peer connection is established, the synchronization data is sent over the secured data path instead of traversing the signaling server.

E. Security Architecture and Authenticated Key Exchange

PocketBridge uses several layers of security to authenticate devices and secure synchronization sessions. The message flow of the authenticated key-exchange protocol is inspired by the session-establishment structure of MTProto 2.0 [18], while the implementation uses standardized cryptographic primitives including P-256 ECDH, Ed25519, HKDF-SHA256, and AES-256-GCM. Each device maintains a locally generated Ed25519 identity key pair. For each synchronization session, the device generates an ephemeral P-256 key pair and a fresh 256-bit handshake nonce. The exchanged ephemeral public keys are authenticated using Ed25519 signatures, which bind them to the identity of the participating device.

The two devices then perform an ephemeral ECDH to establish a shared secret, and use HKDF-SHA256 to derive session-specific cryptographic keys. Direction-specific key derivation isolates keys used for each communication direction. Fresh handshake nonces contribute to session freshness is provided by the use of fresh handshake nonces, and replay protection is provided by monotonically increasing device sequence numbers. For signaling communication PocketBridge uses WSS, and for peer communication uses WebRTC transport security [17]. Application payloads are encrypted with AES-256-GCM for confidentiality and integrity. Application data remains encrypted when it enters the supporting infrastructure, so a relay path will not enable the relay to recover plaintext content if needed.

F. Secure Data Synchronization

After an authenticated session is established, PocketBridge synchronizes application data between authorized devices via the available protected data path. For file synchronization,

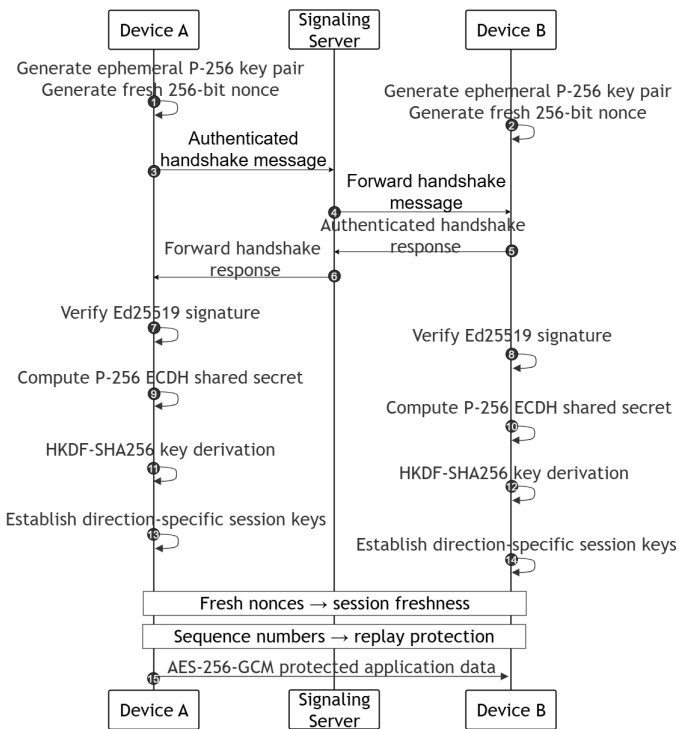


Fig. 2. Authenticated key exchange and session key establishment in PocketBridge.

the sender selects the files and gathers the required file metadata. Large files are divided into manageable chunks. This incremental processing reduces the memory requirements and enables synchronization of large files on heterogeneous devices. A sequence number is attached to each transmitted chunk, which is sent via the encrypted WebRTC DataChannel [15], when direct connectivity is available. If a direct connection is not available, the encrypted relay path is used and application data is protected before being sent over the relay. The receiving device processes the chunks in the order of their sequence information, and sends acknowledgments for successfully received chunks, so that the sender can monitor the transfer progress and detect incomplete transmissions. Once all chunks have been received, the receiving device reassembles the original file and perform an integrity check before making it available to the user. The same protected communication mechanism also supports other application data such as chat and shared scratchpad updates. PocketBridge achieves reliable and privacy-preserving cross-device synchronization by combining incremental transfer, sequencing and acknowledgment, and integrity verification.

G. Threat Model

PocketBridge assumes an honest-but-curious signaling and relay infrastructure that can see communication metadata but must not see plaintext application data. The threat model includes adversaries that monitor, modify, replay, or inject messages on a network, try to access a device without permission, or launch man-in-the-middle attacks during the establish-

ment of a session. The implemented security mechanisms are designed to mitigate these threats through temporary one-time pairing, Ed25519 authentication, ephemeral P-256 ECDH, HKDF-SHA256 key derivation, AES-256-GCM authenticated encryption, fresh nonces, and monotonically increasing device sequence numbers. The model also includes the unauthorized access to the signaling or relay infrastructure, where application data must be kept protected as ciphertext. However, PocketBridge does not provide complete metadata privacy. An observer may still infer source addresses, timestamps, device identifiers, sequence numbers, and ciphertext sizes. Traffic-analysis resistance, denial-of-service attacks, compromised endpoints, malicious authenticated users, and post-compromise security are not part of the primary threat model.

V. PERFORMANCE EVALUATION

A. Experimental Setup

PocketBridge was evaluated using independent test harnesses covering cryptographic integrity, file-processing correctness, authentication and key-establishment properties, and standard and structural attack scenarios. The cryptographic experiments were conducted on an Apple M1 system with 16 GB RAM running macOS 26.6 and Node.js v24.14.1, using the WebCrypto API.

The evaluation included chunk-level integrity tests, whole-file reconstruction tests, cryptographic attack simulations, nonce-generation tests, and SHA-256 processing benchmarks. Integrity and reconstruction experiments used 5 MiB chunks, while the SHA-256 processing benchmark evaluated 1 MiB, 5 MiB, and 10 MiB inputs. The reported experiments used different numbers of repetitions depending on the test, including 1,000 trials for clean integrity and ciphertext-tampering evaluation, 200 trials for file-reconstruction and corruption experiments, and 500 trials for several security attack evaluations.

For the end-to-end performance evaluation, physical sender and receiver devices were used to measure connection establishment time, transfer duration, throughput, resource utilization, transfer success, and cross-platform compatibility. The primary configuration used a MacBook Air with an Apple M1 processor and 16 GB RAM running macOS 26.6 with Safari 26 as the sender, and an Oppo F17 Pro with 8 GB RAM running Chrome as the receiver. The devices communicated over a WorldLink Wi-Fi network using WebSocket signaling and a WebRTC DataChannel with 64 KiB chunks. File sizes of 10 MB, 50 MB, 100 MB, 500 MB, and 1 GB were each transferred 10 times, with the reported transfer measurements averaged across the repetitions.

B. Security and Integrity Evaluation

PocketBridge uses AES-256-GCM authentication tags for authenticated integrity and SHA-256 file digests as a complementary mechanism for detecting corruption during file reconstruction. In 1,000 clean-integrity trials, all chunks were successfully recovered with matching SHA-256 digests, while

all 1,000 ciphertext-tampering attempts were rejected by AES-GCM. In 200 post-decryption corruption trials, all corruptions were detected by SHA-256.

Whole-file reconstruction used 25 MiB files divided into five 5 MiB chunks. All 200 trials were successfully reconstructed, including out-of-order delivery, while all missing-chunk conditions were detected. These results demonstrate detection of both transmission-level and post-decryption corruption, as well as correct chunk reconstruction.

A total of 100,000 96-bit nonces were generated, and no collisions were observed. A controlled nonce-reuse demonstration further confirmed the importance of nonce uniqueness for AES-GCM security.

C. Security Attack Evaluation

The security mechanisms of PocketBridge were evaluated against attacks targeting authentication, confidentiality, message integrity, and key establishment. Wrong-key, MITM key-substitution, replay, chunk-substitution, downgrade/tag-stripping, and known-plaintext tests produced no successful attacks in the tested configurations. Additional tests covering ciphertext non-malleability, authentication-tag truncation, and per-file key independence were also completed successfully. The evaluation identified two implementation-level issues requiring further hardening: a small timing difference in an administrative authentication path and a transcript-canonicalization ambiguity under variable-length field conditions. The latter was not exploitable in the tested configuration because the relevant implementation call sites use fixed-length fields.

D. Cryptographic Processing Overhead

The computational overhead of SHA-256 was evaluated for 1 MiB, 5 MiB, and 10 MiB inputs using 20 measurements per size. The mean processing times were 0.509 ms, 2.364 ms, and 4.934 ms, with corresponding throughputs of 1961.9 MB/s, 2114.3 MB/s, and 2026.5 MB/s, respectively. For the 5 MiB chunk size, the measured p95 processing time was 2.586 ms.

Each protected chunk adds 32 bytes for the SHA-256 digest and 16 bytes for the AES-GCM authentication tag, resulting in 48 bytes of integrity metadata. For a 5 MiB chunk, this represents approximately $9.16 \times 10^{-4}\%$ of the payload. These results indicate relatively low cryptographic processing and metadata overhead on the evaluation platform.

The measured throughput represents cryptographic processing performance and should not be interpreted as end-to-end network or file-transfer throughput.

E. End-to-End Transfer Performance

End-to-end performance was evaluated using physical devices to measure connection establishment, file-transfer performance, resource utilization, and synchronization reliability. The primary test configuration used a MacBook Air with an Apple M1 processor and 16 GB RAM running macOS 26.6 with Safari 26 as the sender, and an Oppo F17 Pro with 8

GB RAM running Chrome as the receiver. The devices communicated over a WorldLink Wi-Fi network using WebSocket signaling and a WebRTC DataChannel with 64 KiB chunks. Each file size was transferred 10 times, and the reported values represent the average application-level measurements.

The mean connection establishment time over 10 trials was 0.951 s. File-transfer experiments using 10 MB, 50 MB, 100 MB, 500 MB, and 1 GB files achieved average transfer times of 0.26, 1.31, 2.58, 12.95, and 26.35 s, respectively. The reported throughput represents application-level file-transfer throughput calculated from transferred file size and measured transfer duration. The relatively stable throughput across file sizes indicates consistent transfer performance for the tested workload, as shown in Fig. 3.

TABLE I
END-TO-END FILE TRANSFER PERFORMANCE

File Size	Avg. Transfer Time (s)	Avg. Throughput (MB/s)
10 MB	0.26	38.46
50 MB	1.31	38.17
100 MB	2.58	38.76
500 MB	12.95	38.61
1 GB	26.35	38.86

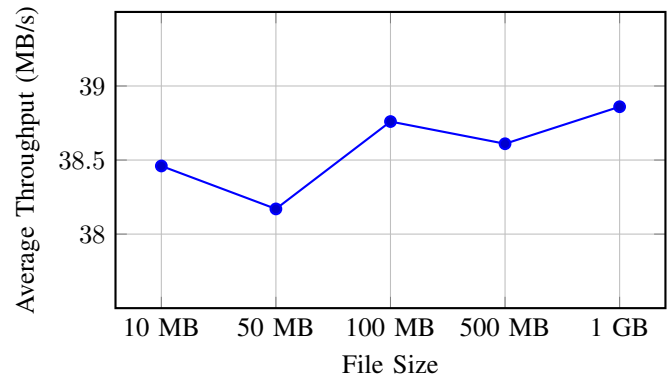


Fig. 3. Average end-to-end throughput for different file sizes.

Resource utilization was also measured during the transfers. Peak CPU usage increased from 12.5% for 10 MB to 48.2% for 1 GB, while peak memory increased from 145 MB to 265 MB. Peak memory usage remained substantially below the corresponding file sizes, consistent with the chunked transfer design.

TABLE II
RESOURCE UTILIZATION DURING FILE TRANSFER

File Size	Avg. CPU (%)	Peak CPU (%)	Peak Memory (MB)
10 MB	8.2	12.5	145
50 MB	14.5	19.2	162
100 MB	21.3	28.4	185
500 MB	34.1	42.6	220
1 GB	41.5	48.2	265

Across 50 transfer attempts, 49 completed successfully and one failed, resulting in an overall success rate of 98%. The

failure occurred during a 500 MB transfer under the tested Wi-Fi condition.

TABLE III
END-TO-END SYNCHRONIZATION RELIABILITY

Total Transfers	Successful	Failed	Success Rate
50	49	1	98%

Under the tested network conditions, same-LAN transfers achieved an average connection time of 1.44 s and throughput of 38.8 MB/s, while transfers between different networks achieved 2.85 s and 14.2 MB/s, respectively. Both configurations achieved a 100% success rate in the reported tests.

TABLE IV
PERFORMANCE UNDER TESTED NETWORK CONDITIONS

Network	Conn. Time (s)	Throughput (MB/s)	Success
Same LAN	1.44	38.8	100%
Different Networks	2.85	14.2	100%

Cross-platform compatibility was additionally verified across macOS, Windows, Android, and iOS browser combinations. All tested combinations successfully established connections and completed file transfers.

TABLE V
CROSS-PLATFORM COMPATIBILITY RESULTS

Sender	Receiver	Connection	File Transfer
macOS Chrome	macOS Chrome	Pass	Pass
macOS Chrome	Windows Chrome	Pass	Pass
macOS Chrome	Android Chrome	Pass	Pass
macOS Chrome	iPhone Safari	Pass	Pass
Windows Chrome	Android Chrome	Pass	Pass
Windows Chrome	iPhone Safari	Pass	Pass

F. Discussion of Results

The end-to-end experiments demonstrated practical transfer performance across physical devices. The tested configuration achieved an average throughput of approximately 38 MB/s across file sizes from 10 MB to 1 GB, with a mean connection establishment time of 0.951 s. Resource measurements showed increasing CPU and memory usage with file size, while peak memory remained substantially below the transferred file size. The network-condition comparison also showed lower throughput and higher connection time across different networks than on the same LAN. However, the current experiments do not isolate the selected WebRTC connectivity path or provide a controlled direct-versus-relay comparison. Consequently, the measured throughput should be interpreted as performance under the tested network and browser configuration rather than as a general characterization of WebRTC performance.

The SHA-256 benchmarks showed processing throughput above 1.9 GB/s across the tested 1 MiB, 5 MiB, and 10 MiB input sizes on the evaluation platform, while the integrity metadata overhead remained small relative to the protected

payload. These results support the practicality of the evaluated cryptographic and integrity mechanisms for the tested workloads.

VI. LIMITATIONS AND FUTURE WORK

The end-to-end evaluation was conducted using a limited set of physical devices, browser configurations, and network conditions. Although transfer performance, resource utilization, reliability, and cross-platform compatibility were evaluated, the experiments do not yet provide broad coverage of heterogeneous hardware, browsers, mobile devices, or network environments. The WebCrypto benchmarks were also executed in a Node.js environment rather than directly within browser-based clients.

The security evaluation also identified areas for further hardening. A small timing difference was observed in the administrative authentication path, while a transcript-canonicalization issue was identified under variable-length field conditions. The latter was not exploitable in the tested configuration because the relevant implementation call sites currently use fixed-length fields. Future work will address these issues and extend the evaluation to real multi-device deployments, diverse network conditions, larger file sets, and longer synchronization sessions. Additional work will investigate performance differences between direct WebRTC and encrypted relay transfers, mobile execution, and broader interoperability across heterogeneous platforms.

VII. CONCLUSION

This paper presented PocketBridge, a privacy-focused cross-platform peer-to-peer framework for secure device communication and file synchronization. The primary contribution is a browser-native architecture that separates connection coordination from application data transfer while combining temporary device authorization, cryptographic device identity, authenticated key exchange, and application-level protection with direct or encrypted relay-based communication.

The security evaluation demonstrated successful detection of tested ciphertext and post-decryption corruption, reliable chunk reconstruction, and successful rejection of the evaluated replay, key-substitution, downgrade, and other tested attack scenarios. The measured SHA-256 processing performance also showed low processing overhead for the tested chunk size. These results indicate that the proposed security and integrity mechanisms provide a practical foundation for privacy-preserving device synchronization.

Further evaluation is required to characterize performance across a broader range of heterogeneous devices, browsers, network conditions, and WebRTC connectivity paths.

REFERENCES

- [1] J. I. Arnes and R. Karlsen, "Cloudless Friend-to-Friend Middleware for Smartphones," in *Proc. Int. Conf. E-Business Telecommun. (ICETE 2018)*, M. Obaidat, Ed., Communications in Computer and Information Science, vol. 1118. Cham, Switzerland: Springer, 2019, pp. 199–218, doi: 10.1007/978-3-030-34866-3_10.

- [2] S. Peng, W. Bao, H. Liu, X. Xiao, J. Shang, L. Han, S. Wang, X. Xie, and Y. Xu, "A peer-to-peer file storage and sharing system based on consortium blockchain," *Future Generation Computer Systems*, vol. 141, pp. 197–204, 2023, doi: 10.1016/j.future.2022.11.010.
- [3] D. Zhan, X. Lin, W. Wang, X. Gan, and S. Bi, "A Design of File Sharing System," in *Proc. IEEE 13th Int. Conf. CYBER Technol. Autom., Control, Intell. Syst. (CYBER)*, pp. 1235–1240, 2023, doi: 10.1109/CYBER59472.2023.10256639.
- [4] Urmila and Shubham, "A Secure and Efficient Cross-Platform P2P File Sharing System," *AIJR Proc.*, vol. 7, no. 6, pp. 171–178, Nov. 2025, doi: 10.21467/proceedings.7.6.20.
- [5] H. Mahmoud and R. Abozariba, "A systematic review on WebRTC for potential applications and challenges beyond audio video streaming," *Multimedia Tools and Applications*, vol. 84, no. 6, pp. 2909–2946, Feb. 2025, doi: 10.1007/s11042-024-20448-9.
- [6] P. Ishola and R. Mailler, "Hybrid WebRTC-WebSocket Communication and Adaptive State Synchronization for Scalable Real-Time 3D Collaborative Whiteboard," in *Proc. 16th Int. Conf. Inf. Commun. Syst. (ICICS)*, pp. 1–6, 2025, doi: 10.1109/ICICS65354.2025.11072969.
- [7] K. Jindal, H. Kharb, and R. K. Nirala, "P2P-SFS: Secured Peer-to-Peer File Sharing with End-to-End Encryption and Authentication," in *Proc. Int. Conf. Intell. Innov. Technol. Comput., Electr. Electron. (IITCEE)*, pp. 1–6, 2026, doi: 10.1109/IITCEE67948.2026.11394461.
- [8] G. R. E, R. J. S, J. T. Chacko, J. P. C, and J. K. K, "Peer-to-Peer File Streaming Using Web Sockets Protocol," in *2023 Int. Conf. Recent Trends Electron. Commun. (ICRTEC)*, pp. 1–6, 2023, doi: 10.1109/ICRTEC56977.2023.10111896.
- [9] D. Rozen Berg, M. Tharunraj, B. Raj Kumar, M. R. Sumalatha, L. H. Palivela, and P. V. Karthikeyaa, "WebRTC-based Decentralized Chat Application with Minimal Latency," in *Proc. Int. Conf. Intell. Innov. Eng. Technol. (ICIET)*, pp. 210–215, 2022, doi: 10.1109/ICI-IET55458.2022.9967597.
- [10] N. C, N. M, S. K. K. L, and R. D. Bharathi, "Decentralized Real-Time Communication: WebRTC and WebSockets for Peer-to-Peer File Sharing," in *Proc. Int. Conf. Smart Technol. Sustain. Develop. Goals (IC-STDG)*, pp. 1–5, 2024, doi: 10.1109/ICSTDG61998.2024.11026207.
- [11] R. N. Shaikh, A. Poojary, Suraj, S. Bhandary, and Sumanth, "DropBridge-A-WebRTC-Based-P2P-File-Transfer-Tool," in *Proc. Int. Conf. Future Technol. (ICFT)*, pp. 1–6, 2025, doi: 10.1109/ICFT66708.2025.11336315.
- [12] A. Jain, L. Arora, A. Raj, and V. Pathak, "Decentralized Peer-to-Peer File Sharing System," *Int. J. Res. Anal. Rev. (IJRAR)*, vol. 11, no. 4, pp. 379–385, Nov. 2024, Art. no. IJJAR24D1789.
- [13] C. Jennings, T. Hardie, and M. Westerlund, "Real-time communications for the web," *IEEE Communications Magazine*, vol. 51, no. 4, pp. 20–26, Apr. 2013, doi: 10.1109/MCOM.2013.6495756.
- [14] I. Fette and A. Melnikov, "The WebSocket Protocol," RFC 6455, RFC Editor, Dec. 2011, doi: 10.17487/RFC6455.
- [15] R. Jesup, S. Loreto, and M. Tüxen, "WebRTC Data Channels," RFC 8831, RFC Editor, Jan. 2021, doi: 10.17487/RFC8831.
- [16] A. Keränen, C. Holmberg, and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal," RFC 8445, RFC Editor, Jul. 2018, doi: 10.17487/RFC8445.
- [17] E. Rescorla, "WebRTC Security Architecture," RFC 8827, RFC Editor, Jan. 2021, doi: 10.17487/RFC8827.
- [18] Telegram, "MTProto 2.0: Cloud chats," Telegram Core Documentation, 2021. [Online]. Available: <https://core.telegram.org/mtproto/AJiEAWIYFoAsBGJBjZwYoQIwFMAm/2-0>.