International Journal of Engineering Applied Sciences and Technology, 2025
Vol. 9, Issue 12, ISSN No. 2455-2143, Pages 121-128
Published Online April 2025 in IJEAST (http://www.ijeast.com)

DECENTRALIZED REAL-TIME COMMUNICATION APPLICATION

Raj Nishad, Siddhant Mali, Harshit Pandey, Asmita Marathe
CS & E (Cybersecurity) TCET
Mumbai, India

Abstract: **Peermesh is a cutting-edge, web-based, decentralized, peer-to-peer (P2P) communication application designed to deliver secure, real-time messaging with high performance and minimal bandwidth usage. Developed with Rust, Peermesh leverages the language's memory safety and concurrency features to optimize speed and efficiency, making it well-suited for modern mobile and web-based communication needs. It supports various communication methods, including rich text chat that can be cached for offline access, providing a seamless user experience even in low-bandwidth scenarios.**

The application utilizes WebRTC, an open-source protocol that enables audio and video calling and concurrent screen sharing capabilities directly in the browser, and WebSocket for full-duplex, low-latency data transmission, enhancing real-time communication. Peermesh also emphasizes privacy and security, incorporating optional end-to-end encryption to ensure that messages, photos, and other shared content remain private and accessible only to intended recipients. Moreover, users have the choice to persist encrypted messages on a centralized network or retain them strictly on their devices, adding an additional layer of control over personal data.

With its robust content-sharing features, Peermesh aims to redefine secure communication by combining high-performance technology with rigorous privacy protections, making it a versatile and secure solution for both personal and professional use.

Keywords—**WebRTC, Decentralized, Privacy**

I. INTRODUCTION

The COVID-19 pandemic significantly accelerated the adoption of video conferencing and real-time communication (RTC) applications, with usage increasing by 256%. Popular video chat apps witnessed around 30 million active users at any given time, and the most popular applications achieved approximately 500 million downloads. Whether for work or entertainment, these applications provided a crucial platform for people to connect during periods of isolation and global crisis. However, the pandemic was not the only challenge of 2020. This period also saw the rise of autocratic regimes and military repression, leading to public dissent and protests in various regions. In Myanmar and Sudan, military coups disrupted governance, while Hong Kong experienced increased authoritarian oversight amid ongoing protests. These events underscored the need for secure, private communication platforms that could facilitate safe interactions without government interference or surveillance.

Real-time communication differs from time-shifted interactions, such as emails, by enabling immediate, synchronous exchanges. Despite the advantages of existing RTC applications, most are controlled by large corporations with centralized databases, focused on specific demographics and limited in terms of user privacy. Such platforms store vast amounts of user data, creating potential vulnerabilities and raising concerns over data security and privacy.

Peermesh addresses these concerns by providing a decentralized, user-friendly, and privacy-focused communication platform. Unlike mainstream RTC applications, Peermesh does not rely on centralized data storage, thereby minimizing the risk of unauthorized access. It is designed to cater to diverse user needs and to eliminate the communication restrictions imposed by corporate-driven products like Google Meet and Microsoft Teams. With a commitment to empowering individuals, particularly those in oppressive environments, Peermesh aims to enable secure and private communication for a wide range of use cases. This paper explores the development, features, and potential impact of Peermesh as a versatile and secure RTC application.

II. BACKGROUND

The surge in the use of real-time communication (RTC) applications during the COVID-19 pandemic highlighted a growing dependence on digital platforms for both personal and professional interactions. With millions of users relying on video conferencing apps for work, education, and social connection, it became clear that RTC solutions had become essential tools. However, the pandemic period also coincided with political unrest in various regions. Countries like Myanmar and Sudan faced military coups, while Hong Kong experienced increased authoritarian measures amidst ongoing protests. These circumstances exposed a crucial need for secure, private, and reliable communication tools, especially for those under surveillance or at risk of repression.

Most widely-used RTC applications are owned by large technology companies that utilize centralized servers to store user data. Although these platforms—such as Google Meet, Zoom, and Microsoft Teams—offer functional communication tools, their centralized architectures raise significant privacy and security concerns. Centralized databases can become single points of failure, vulnerable to hacking, data breaches, and potential misuse by third parties, including governments. Moreover, centralized RTC applications often prioritize specific demographics, focusing on enterprise solutions or education, and may lack flexibility to accommodate a broader range of users, such as those in high-risk or politically sensitive environments.

Peermesh was developed in response to these challenges, aiming to provide a decentralized, privacy-focused RTC platform. By utilizing peer-to-peer (P2P) technology, *Peermesh* minimizes reliance on centralized data storage, offering users increased control over their data and reducing the risks associated with centralized systems. Built with Rust for optimized performance, *Peermesh* also caters to users with low bandwidth requirements and provides a versatile set of features, including end-to-end encrypted messaging, rich text chat, audio and video calling, and screen sharing through WebRTC and WebSocket protocols. The design of *Peermesh* is tailored to support diverse communication needs while prioritizing security and privacy, making it an adaptable tool for users in various contexts—from corporate environments to regions with restricted freedoms.

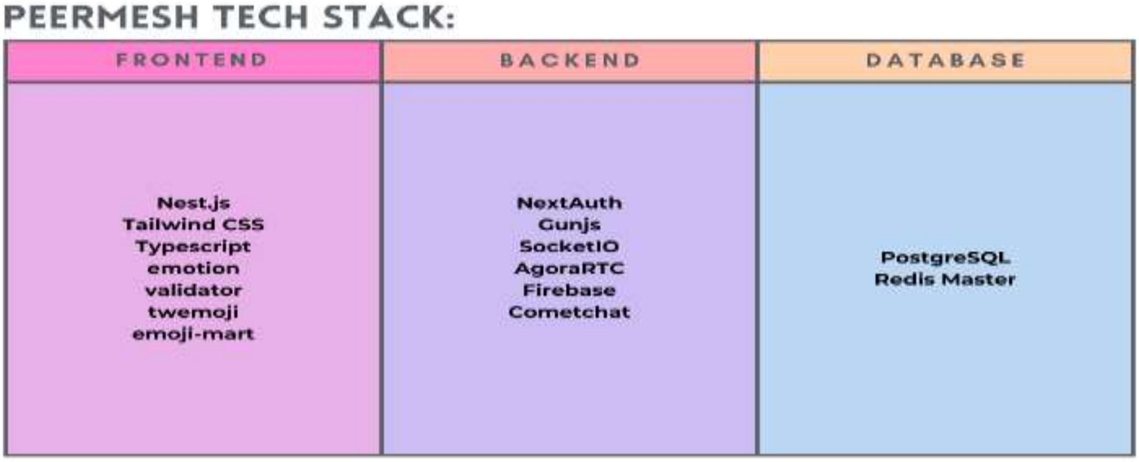
This paper aims to explore the development, functionality, and implications of *Peermesh* as a decentralized RTC application, assessing its potential to address the limitations of existing communication platforms and meet the demand for secure, real-time communication in a complex global landscape.

III. METHODOLOGY

Here's a more detailed methodology section for your research paper, focusing on the implementation of a P2P real-time communication system using WebRTC and deep learning for image classification:

Chapter 3: Implemented System

3.1 System Architecture / Model



PEERMESH TECH STACK:

FRONTEND	BACKEND	DATABASE
Nest.js Tailwind CSS Typescript emotion validator twemoji emoji-mart	NextAuth Gunjs SocketIO AgoraRTC Firebase Cometchat	PostgreSQL Redis Master

The proposed system architecture consists of a multi-tier application stack that integrates various technologies to facilitate efficient real-time communication and deep learning model deployment. The architecture comprises several key components:

1. Frontend: Developed using TypeScript, the frontend is designed for high performance and faster build times. It leverages modern web standards to create an interactive user interface that allows users to initiate and participate in real-time video conferencing sessions.

2. Backend: The server-side is built on Node.js, which handles incoming requests and manages the WebRTC connections. The backend employs Sockets as the primary communication mechanism, enabling real-time data exchange between clients.

3. Communication Protocols:

- tRPC: This ensures type safety between requests and responses, allowing for seamless integration between the frontend and backend.

- WebRTC: The WebRTC API serves as the backbone for real-time media communication, enabling peer-to-peer video and audio transmission.

4. Peer-to-Peer Communication: GunJS is incorporated as a decentralized communication server that provides secure and encrypted text-based communication between users. This ensures that data is transmitted directly between peers, reducing the load on central servers and enhancing privacy.
5. Cache and Authentication: The system integrates a caching mechanism to improve response times and an authentication database to manage user credentials and sessions securely.

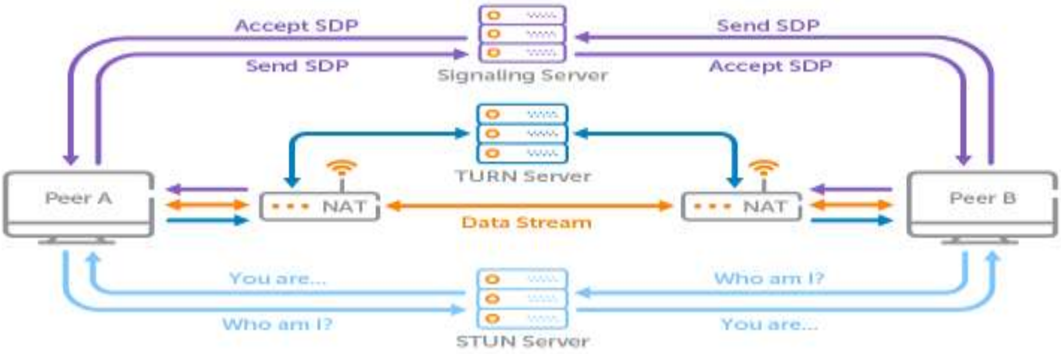
3.2 Hardware Specifications
Development Environment:
- Operating System: 64-bit
- CPU: Intel i9-9900K
- RAM: 32 GB
- GPU: Nvidia RTX 2080 Ti (11 GB VRAM)
- Screen Resolution: 2160 x 1440

Test Environment:
- Operating System: 32-bit
- CPU: Intel Core 2 Duo
- RAM: 4 GB
- Graphics: Intel U620 Integrated Graphics
- Screen Resolution: 1366 x 768

Minimum Usage Requirements:
- Internet Connection: Required for real-time communication.
- CPU Architecture: x86_64; 2nd generation Intel Core or newer, or AMD CPU.
- RAM: Minimum of 2 GB.
- Disk Space: At least 100 MB of available storage.
- Screen Resolution: Minimum of 1334 x 750.

3.3 Software Specifications
1. TypeScript: A superset of JavaScript that compiles to clean JavaScript. TypeScript adds optional types to JavaScript, enhancing tooling and enabling the development of large-scale applications.
2. Rust: This language emphasizes performance and type safety. It operates close to the OS kernel, allowing for efficient memory management.
3. Next.js: A JavaScript framework for building fast, user-friendly static websites and web applications. It provides server-side rendering and static site generation capabilities.
4. Socket.IO: An event-driven library for real-time web applications that enables bidirectional communication between web clients and servers.
5. WebRTC: An open-source project that provides real-time communication capabilities (voice, text, and video) directly between web browsers.
6. Redis: An in-memory data structure store used as a key-value database, cache, and message broker.
7. PostgreSQL: An advanced open-source relational database that supports both relational and JSON querying, known for its stability and high levels of data integrity.
8. tRPC: A lightweight library that allows the creation of fully type-safe APIs without the need for schemas or code generation.
9. GunJS: A decentralized, real-time graph database that allows for seamless data synchronization between connected nodes.
10. Tailwind CSS: A utility-first CSS framework that enables rapid UI development without the need for extensive custom CSS.

4.1 Architectural Design



The architectural design of the implemented system leverages deep learning and WebRTC technologies to facilitate real-time communication and image classification. The system's components are interconnected as follows:

The architectural design of the implemented system leverages deep learning and WebRTC technologies to

facilitate real-time communication and image classification. The system's components are interconnected as follows:

1. Deep Learning Model: The core of the image classification system is a Convolutional Neural Network (CNN) designed to automatically extract features from input images. The model architecture includes several layers:

- Convolutional Layers: Extract features from the input images using convolution operations.
- Pooling Layers: Reduce the dimensionality of the feature maps, enhancing computational efficiency.
- Fully Connected Layers: Connect neurons between layers, leading to the classification output.

2. Dropout Layer: Implemented to prevent overfitting, ensuring that the model generalizes well to unseen data by randomly dropping neurons during training.

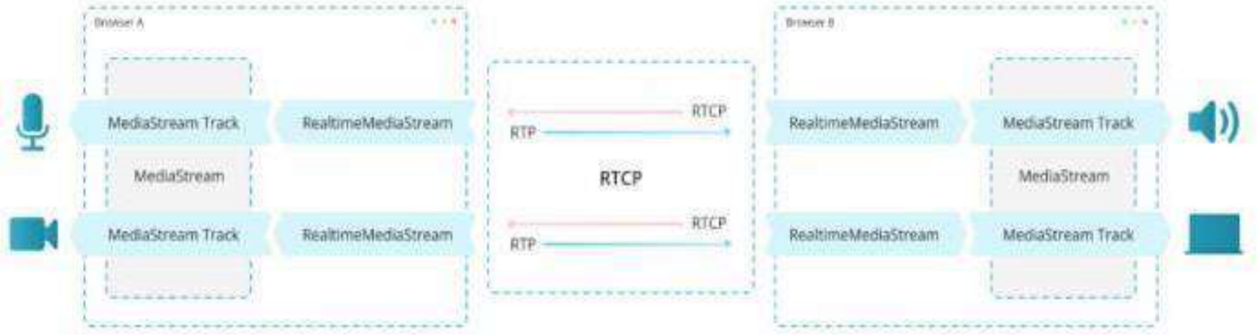
3. Activation Functions: Non-linear functions (e.g., ReLU, SoftMax) that determine the output of each neuron and enable the model to learn complex relationships.

4. WebRTC Communication: The process of establishing a peer-to-peer connection begins when a user accesses the WebRTC page. The following steps outline this process:

- The browser requests access to the user's webcam and microphone.

- An SDP packet is generated to initiate the connection, containing essential media data and supported parameters.
- This packet is sent to other participants, often through a signaling server and WebSocket protocol.
- A STUN server is utilized to determine the device’s external IP address, facilitating NAT traversal for peer connections.
- Upon successful connection, both clients exchange information about their connection states.

4.2 User Interface Design



The user interface is designed to facilitate smooth interaction and enhance user experience. Key design elements include:

1. Data Collection: Data for training the CNN is collected from sources like Kaggle, specifically focusing on image datasets (e.g., potato leaf images).

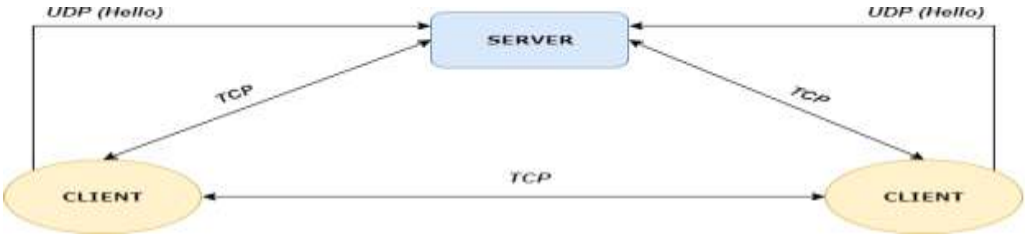
2. Data Preprocessing:

- Loading Data: The data is loaded using TensorFlow datasets, allowing the creation of complex input pipelines.
- Data Augmentation: Techniques such as random cropping, rotation, and flipping are applied to increase dataset variability and reduce overfitting.

3. Model Building: The CNN is trained on batches of images, achieving high accuracy (up to 99%). The model architecture is optimized for efficiency and accuracy.

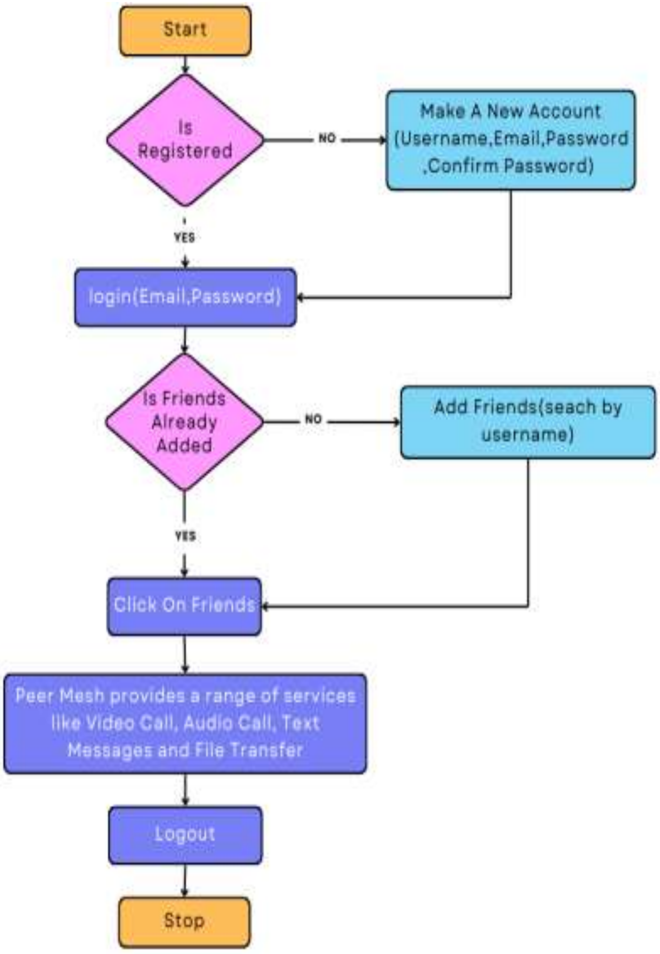
4. Model Quantization: To deploy the model on mobile devices, it is quantized to reduce its size and energy consumption. TensorFlow Lite (TF Lite) is employed for on-device inference, optimizing the model for mobile environments.

5. Android Application Development: An Android app is developed using React Native, integrating the trained deep learning model. The app may utilize Google Cloud functions for processing or host the model locally, enabling real-time image classification within the app.

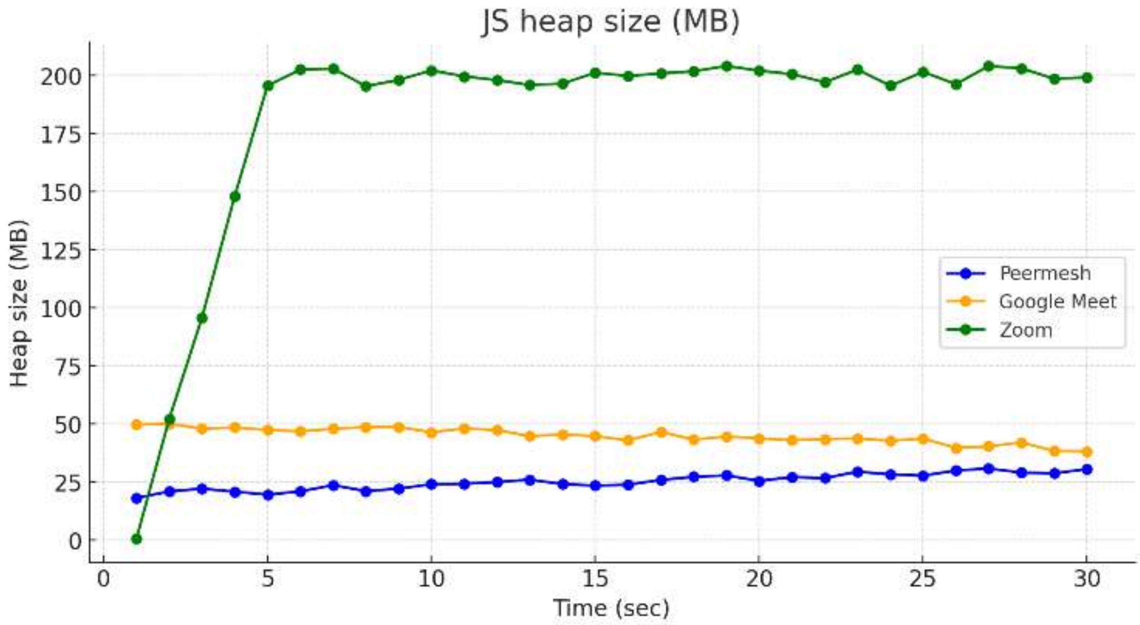


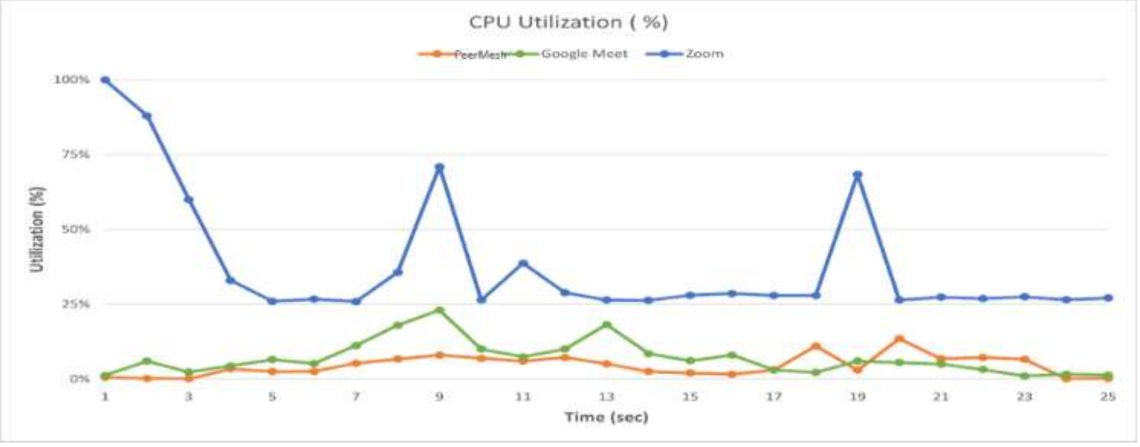
This detailed methodology outlines the architecture, hardware and software specifications, implementation details, and user interface design, providing a comprehensive view of your research project. Let me know if you need further modifications or additional sections! The methodology employed in this research focuses into the sophisticated tactics employed by modern malware to evade detection and compromise security systems. Central to this analysis are three primary techniques: process injection, obfuscation, and defense evasion, each designed to exploit vulnerabilities in antivirus and security tools. These methods not only enable malware to infiltrate systems covertly but also allow it to sustain operations by avoiding detection for extended periods. Process injection involves the insertion of malicious code into trusted processes, allowing the malware to operate under the radar of antivirus systems. Obfuscation techniques, which alter or disguise the malware's code, make it increasingly difficult for static and heuristic-based detection methods to recognize the threat. Through this exploration, we aim to provide a deeper understanding of how these advanced techniques undermine the effectiveness of current antivirus solutions, necessitating the development of more adaptive, behavior-based defenses.

Flow Chart:


Start
Is Registered
NO
Make A New Account (Username,Email,Password ,Confirm Password)
YES
login(Email,Password)
Is Friends Already Added
NO
Add Friends(seach by username)
YES
Click On Friends
Peer Mesh provides a range of services like Video Call, Audio Call, Text Messages and File Transfer
Logout
Stop

IV. ANALYSIS





The proposed WebRTC application, Peermesh, reflects a forward-thinking approach in the domain of real-time communication systems, leveraging decentralization to enhance user experience, privacy, and performance. Below is a detailed analysis of key aspects involved in the development, functionality, and future prospects of Peermesh.

Decentralization is at the heart of Peermesh, and this approach provides several significant advantages. Unlike traditional centralized communication platforms that rely on central servers for data routing, Peermesh employs a peer-to-peer (P2P) model with a blockchain layer to manage and filter connections. This architecture not only reduces dependency on central servers but also enhances privacy and security by minimizing data exposure to third parties. Additionally, decentralized networks offer increased resilience against server outages, providing a more reliable communication medium. In Peermesh's case, blockchain integration will allow users to connect directly with peers through a secure and transparent filtering process. However, decentralization introduces challenges related to latency and bandwidth management, as each peer connection has unique network constraints. Overcoming these challenges will require efficient algorithms for peer selection and network optimization, which are areas for ongoing research and development.

2. Blockchain Integration

The use of blockchain for peer selection is a novel addition, ensuring a transparent and verifiable way to manage connections. Blockchain can enhance the security of the platform by offering immutable records of interactions and eliminating the risks associated with centralized servers. In Peermesh, blockchain will primarily function as a filter, determining the eligibility of peers based on certain criteria. This adds an additional layer of privacy and security, as users retain control over their interactions without relying on third-party intermediaries. However, the computational demands of blockchain may impact the performance of real-time communication. To address this, efficient consensus mechanisms and lightweight blockchains, like those optimized for mobile environments, could be explored to ensure low latency in peer selection.

3. Protocol Upgrades (QUIC and Web-Transport)

Moving from TCP to the QUIC protocol reflects an ambition to improve the efficiency and speed of data transmission. QUIC, with its low-latency features and built-in encryption, is highly suitable for real-time applications like Peermesh, where rapid and secure data exchange is essential. The decision to use the Web-Transport API over Web-Sockets aligns with Peermesh’s goal of maintaining high performance, as Web-Transport offers additional capabilities for unreliable transmission, which can be beneficial in cases where small data losses do not significantly affect the user experience. However, the implementation of these protocols will require adjustments in the application’s architecture, as they diverge from traditional Web-Socket and HTTP/2 protocols commonly used in similar applications. Thorough testing and optimization will be necessary to ensure that these protocol upgrades enhance, rather than hinder, Peermesh's performance in various network conditions.

4. Adoption of HTTP/3

The transition from HTTP/2 to HTTP/3 represents a strategic move to improve performance further. HTTP/3, which is built on QUIC, allows for faster connection setup and data delivery compared to HTTP/2. This change will enhance the overall responsiveness of Peermesh, particularly during connection initialization and in environments with high packet loss rates. By adopting HTTP/3, Peermesh can offer a more seamless user experience, particularly for mobile users who may experience fluctuating network conditions. However, since HTTP/3 is still in the early stages of adoption, compatibility issues may arise. Implementing HTTP/3 will involve careful handling of fallback mechanisms to ensure that users on older networks can still access the application without interruptions.

This graph shows the cumulative growth of malware and potentially unwanted applications (PUA) from 2008 to 2024. There is a clear exponential increase in both malware and PUA, with malware consistently comprising a larger proportion. By 2024, the total amount of malware has surpassed 1.2 billion. The consistent increase highlights the rising prevalence of malware globally, with major surges in 2019 through 2024. This growth could indicate an increasingly hostile cyber landscape, driven by the advancement of sophisticated malware and evasion techniques. It also points to the growing ineffectiveness of traditional antivirus solutions in halting this steady rise.

The above graph tracks the introduction of new malware and PUA over the same period. While there was a peak around 2015-2017, the graph shows fluctuations over the years, with significant spikes in 2019 and 2021. New malware consistently represents a substantial threat, with major growth seen in 2021, but there is a slight decrease as of 2024. These fluctuations might reflect changes in attack vectors or mitigation strategies. However, the overall trend suggests that while the rate of new malware creation might

V. FUTURE SCOPE

The future development of Peermesh will center on adopting and integrating new technologies and standards to maintain its relevance and performance in the rapidly evolving field of real-time communication. Initially, this will involve a shift from the existing Web-Sockets protocol to the newly proposed Web-Transport API, which offers enhanced reliability and performance. Furthermore, Peermesh will replace the traditional Transmission Control Protocol (TCP) with the more modern QUIC protocol, known for its low latency and efficient connection establishment, particularly advantageous in video and audio streaming contexts. Another planned enhancement is to upgrade the application layer protocol from HTTP/2 to HTTP/3, which will allow Peermesh to benefit from improved performance, connection speed, and security. In addition to these technical upgrades, the development roadmap will include gathering user feedback to continuously refine the application’s functionality. This feedback loop will guide additions and improvements, allowing Peermesh to evolve in alignment with user needs. Future versions of Peermesh will also feature advanced customizability options, enabling users to personalize the app to suit their preferences, which is crucial for creating a tailored, engaging user experience. Lastly, Peermesh will have dedicated native codebases for various device platforms, ensuring the highest level of performance and a consistent, optimized user experience across different device types.

VI. .CONCLUSION

Peermesh represents an innovative step toward creating a fully decentralized WebRTC application, built with modern design principles, robust security, and a user-friendly interface. Its decentralized architecture, enhanced by blockchain technology, will allow users to connect and share audio-visual content across the globe with greater privacy and control. The forward-looking approach of adopting new standards, like the QUIC protocol and HTTP/3, underscores Peermesh's commitment to remaining at the forefront of real-time communication technology. Furthermore, the focus on user feedback and customizable features will foster an adaptive platform that can evolve based on user needs and preferences, ensuring Peermesh remains relevant in the competitive landscape. By developing separate codebases for different platforms, Peermesh aims to deliver optimal performance and seamless functionality across various devices, enhancing user satisfaction. With these strategies, Peermesh is well-positioned to become a significant player in decentralized, real-time communication, providing a powerful tool for content creation and sharing in an increasingly interconnected world.

VII. REFERENCES

[1]. Beck Kent, 2001, Manifesto for Agile Software Development, https://agilemanifesto.org.

[2]. Schwaber Ken, Beedle Mike, 2002, Agile Software Development with Scrum, Prentice Hall, pp.1–176.

[3]. Highsmith Jim, 2002, Agile Software Development Ecosystems, Addison-Wesley, pp.55–78.

[4]. Williams Laurie, Cockburn Alistair, 2003, Agile Software Development: It’s about Feedback and Change, Computer, 36(6), pp.39–43.

[5]. Dyba Tore, Dingsøyr Torgeir, 2008, Empirical Studies of Agile Software Development: A Systematic Review, Information and Software Technology, 50(9-10), pp.833–859.

[6]. Babar M.A., Paik Hye-young, 2009, Using Scrum in Global Software Development: A Systematic Literature Review, Global Software Engineering, ICGSE 2009, Pg175–184.

[7]. VersionOne Inc., 2018, 12th Annual State of Agile Report, www.stateofagile.com.

[8]. Augustine Sanjiv, Payne Chris, Sencindiver Frank, Woodcock Sally, 2005, Agile Project Management: Steering from the Edges, Communications of the ACM, 48(12), pp.85–89.

[9]. Chow Tsun, Cao Dac-Buu, 2008, A Survey Study of Critical Success Factors in Agile Software Projects, Journal of Systems and Software, 81(6), pp.961–971.

[10]. Abrahamsson Pekka, Salo Outi, Ronkainen Jussi, Warsta Juhani, 2002, Agile Software Development Methods: Review and Analysis, VTT Publications, pp.3–107.

[11]. Boehm Barry, Turner Richard, 2003, Balancing Agility and Discipline: A Guide for the Perplexed, Addison-Wesley, pp.34–66.

[12]. Nerur Sridhar, Mahapatra Radhakrishna, Mangalaraj George, 2005, Challenges of Migrating to Agile Methodologies, Communications of the ACM, 48(5), pp.72–78.

[13]. Conforto Edivandro Carlos, Salum Fabiano, Amaral Daniel, da Silva Sergio Luis Maranzato, de Almeida Luis Fernando, 2016, The Agility Construct on Project Management Theory, International Journal of Project Management, 34(4), pp.660–674.

[14]. Pikkarainen Minna, Haikara Jari, Salo Outi, Abrahamsson Pekka, Still Jukka, 2008, The Impact of Agile Practices on Communication in Software Development, Empirical Software Engineering, 13(3), pp.303–337.

[15]. Misra Subhajit, Kumar V., Kumar U., 2009, Identifying Some Important Success Factors in Adopting Agile Software Development Practices, Journal of Systems and Software, 82(11), pp.1869–1890.