TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Elektrotechnik und Informationstechnik Junior Professorship of Haptic Communication Systems

Real-Time Synchronization of Multimedia Streams for Teleoperation

Yixin Yuan

Matriculation number: 4975933

Diploma Thesis

to achieve the academic degree

Diplomingenieur (Dipl.-Ing.)

Supervisor

Dipl.-Ing. Mingyu Ma

M.Sc. Yushan Yang

Supervising professor

Jun.-Prof. Dr.-Ing. Giang T. Nguyen

Submitted on: 15th November 2024

TECHNISCHE UNIVERSITÄT DRESDEN
FAKULTÄT ELEKTROTECHNIK UND INFORMATIONSTECHNIK

Task for diploma thesis

For Mr. Yixin Yuan (Matr.-No: 4975933) ET/IT 2020

Topic: Real-Time Synchronization of Multimedia Streams for Teleoperation

Task description: With the development of science and technology and the application of AR, VR, and other technologies, multimedia data streams, such as video, audio, and haptic, are increasingly prevalent in networks. These applications have higher requirements for latency and synchronization. To address the latency problem, network slicing has been proposed. However, with differentiated services, the multimedia streams can be out of sync. Furthermore, the expansion of network capacities is accompanied by increased buffer sizes at each network node. Excessive buffer occupancy can lead to buffer bloat, a condition where the increased latency and reduced throughput undermine network performance. To tackle these multifaceted challenges, a novel scheme will be proposed for addressing the critical synchronization issues present in modern network operations.

To complete this thesis, the student must implement and assess the proposed scheme. This involves providing a comprehensive understanding and evaluating performance in various network scenarios.

Expected outcomes:

1. Conduct a literature review on synchronization for multimedia streams, especially for video and haptic streams.
2. Build a testbed and record the traces of multimedia streams.
3. Propose an algorithm for synchronization of video and haptic streams in OpenWrt.
4. Implement the proposed algorithm.
5. Assess system performance, including synchronization rate and latency.
6. Write a report on this synchronization topic.

Requirements: Preferred programming languages are C++ and Python, along with skills in benchmarking and analyzing network systems.

Supervisors: Mingyu Ma, Yushan Yang

Reviewer: Jun.-Prof. Dr.-Ing. Giang T. Nguyen
Second Reviewer: Prof. Dr.-Ing. Dr. h.c. Frank H.P. Fitzek

Started at: 03.06.2024
To be submitted by: 11.11.2024

This diploma thesis will be written in English.

Prof. Dr. rer. nat. habil. Hans Georg Krauthäuser
Vorsitzender des Prüfungsausschusses

Jun.-Prof. Dr.-Ing. Giang T. Nguyen
Verantwortlicher Hochschullehrer

Statement of authorship

I hereby certify that I have authored this document entitled *Real-Time Synchronization of Multimedia Streams for Teleoperation* independently and without undue assistance from third parties. No other than the resources and references indicated in this document have been used. I have marked both literal and accordingly adopted quotations as such. There were no additional persons involved in the intellectual preparation of the present document. I am aware that violations of this declaration may lead to subsequent withdrawal of the academic degree.

Dresden, 15th November 2024

Yixin Yuan

Abstract

In teleoperation systems, the real-time synchronization of multimedia streams, including video and haptic feedback, is crucial for accurate and responsive interactions between operators and remote environments. This thesis presents a framework designed to achieve low-latency and synchronized multimedia streams, thereby enhancing the Quality of Experience (QoE) in teleoperation applications. We implement an eXpress Data Path (XDP) algorithm on an OpenWrt router and integrate it with Active Queue Management (AQM) techniques—specifically, Controlled Delay (CoDel) and Common Applications Kept Enhanced (CAKE)—to manage network congestion effectively. This approach optimizes frames per second (FPS) and the haptic-video synchronization offset under varying network conditions, particularly in low-bandwidth environments where latency and packet loss pose significant challenges.

Experimental results indicate that XDP, in conjunction with AQM, increases FPS by up to 10 in low-bandwidth scenarios and reduces the haptic-video synchronization offset by up to 20 ms, meeting the Quality of Service (QoS) requirements for teleoperation tasks. We compare algorithm performance with and without XDP across different AQM setups, showing that XDP with CoDel achieves the best balance between frame rate stability and latency, while XDP with CAKE offers a lower synchronization offset with a slight reduction in frame rate. This thesis highlights the trade-offs between latency and frame rate when selecting AQM strategies, providing valuable insights for the design of teleoperation systems that require real-time video quality and precise haptic feedback.

Key Words: Real-time synchronization, XDP, AQM, QoS

Zusammenfassung

In Teleoperationssystemen ist die Echtzeitsynchronisation von Multimedia-Streams, einschließlich Video und haptischem Feedback, entscheidend für eine präzise und schnelle Interaktion zwischen Bedienern und entfernten Umgebungen. Diese Arbeit stellt ein Framework vor, das auf die Bereitstellung von latenzarmen und synchronisierten Multimedia-Streams abzielt, um die Benutzererfahrung (QoE) in Teleoperationsanwendungen zu verbessern. Dafür implementieren wir einen eXpress Data Path (XDP)-Algorithmus auf einem OpenWrt-Router und kombinieren ihn mit Methoden des Active Queue Management (AQM), insbesondere Controlled Delay (CoDel) und Common Applications Kept Enhanced (CAKE), um Netzwerküberlastungen effektiv zu steuern. Unser Ansatz optimiert sowohl die Bildrate (FPS) als auch die Synchronisation zwischen haptischen und Video-Streams unter verschiedenen Netzwerkbedingungen, insbesondere in Szenarien mit geringer Bandbreite, in denen Latenz und Paketverlust große Herausforderungen darstellen.

Die experimentellen Ergebnisse zeigen, dass XDP in Verbindung mit AQM die FPS in Szenarien mit niedriger Bandbreite um bis zu 10 steigern und die Synchronisationsabweichung zwischen haptischen und Video-Streams um bis zu 20 ms verringern kann, was die Anforderungen an die Dienstgüte (QoS) für Teleoperationen erfüllt. Wir vergleichen die Leistung des Algorithmus mit und ohne XDP über verschiedene AQM-Konfigurationen hinweg und zeigen, dass XDP mit CoDel die beste Balance zwischen stabiler Bildrate und Latenz bietet, während XDP mit CAKE eine geringere Synchronisationsabweichung erreicht, jedoch mit einem leichten Rückgang der Bildrate. Diese Arbeit verdeutlicht die Abwägungen zwischen Latenz und Bildrate bei der Auswahl von AQM-Strategien und liefert wertvolle Erkenntnisse für die Entwicklung von Teleoperationssystemen, die sowohl eine hohe Videoqualität in Echtzeit als auch präzises haptisches Feedback erfordern.

Schlüsselwörter: Echtzeitsynchronisation, XDP, AQM, QoS

Contents

1 Introduction

Teleoperation refers to the control of a robot over a network by an operator at a distant location to complete specific tasks [1]. This setup is essential in fields ranging from industrial automation to remote survey and immersive virtual reality (VR) environments, where multimedia communication enables effective and precise remote interaction. Both visual and tactile information are crucial in these scenarios, as the stable and low-latency transmission of these signals directly impacts task accuracy, operator satisfaction, and overall experience.

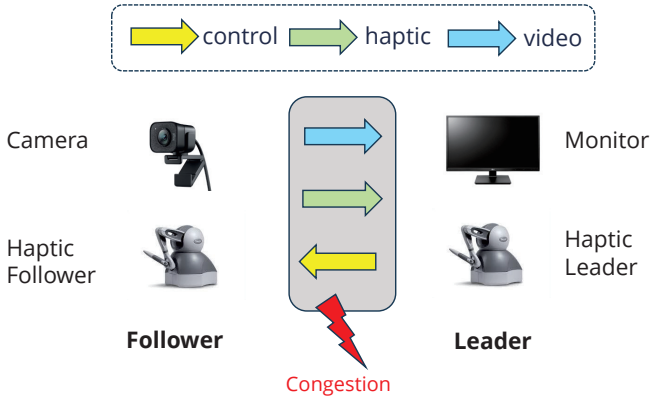


Figure 1.1: Teleoperation Model with Control, Haptic and Video Stream.

While teleoperation relies on the stable and low-latency transmission of both visual and tactile signals, as shown in Figure 1.1, a major challenge arises from network congestion or fluctuations, which can disrupt the synchronization of these signals. Variations in network latency, packet loss, leading to a mismatch between the operator's actions and the robot's responses. For example, if the timing between haptic and video signals is not synchronized, the operator might see actions that don't match the tactile feedback they're experiencing, making tasks more difficult and potentially unsafe. This mismatch can reduce accuracy in precise tasks like remote surgery. Solving these timing issues is essential for ensuring smooth, responsive teleoperation, especially in applications where accuracy and immersion are critical.

The signal types involved in remote operations have unique characteristics and specific requirements for transmission quality, which further complicates the challenge. Tactile signals, which require relatively low bandwidth but operate at a high frequency of 1 kHz to provide real-time haptic feedback [2], allow the operator to accurately perceive touch and resistance. This capability is essential in applications like remote surgery, where precise tactile feedback enables the delicate manipulation of instruments. In contrast, video signals demand higher

bandwidth to maintain a consistent at least 30 frames per second, ensuring smooth visual perception and helping the operator interpret the remote environment effectively [3]. In VR applications, high-quality video at stable frame rates is crucial to maintain immersion, as disruptions in visual data can disorient users and degrade the experience [4].

These varied transmission needs highlight the importance of a network approach that dynamically adapts to the specific requirements of each signal type. However, network congestion remains a persistent challenge, particularly in traditional networks that route all traffic through a single channel and frequently employ the first-come, first-served (FCFS) strategy. In such configurations, tail packets are often discarded when congestion occurs, leading to potential loss of important data. Active Queue Management (AQM) techniques like Controlled Delay (CoDel) [5] and Common Applications Kept Enhanced (CAKE) [6] have been introduced to manage congestion more effectively. By prioritizing critical packets and minimizing queue length, these AQM methods help reduce latency and improve overall network efficiency under moderate congestion.

This prioritization strategy, supported by AQM techniques such as CoDel and CAKE, offers a significant advantage: it enables the system to support both accuracy and responsiveness by allocating network resources according to the specific demands of each task. As a result, operators experience improved control and task performance, while the network remains resilient and adaptable, even when demands are high.

Despite these advantages, traditional Active Queue Management (AQM) techniques within the Traffic Control (TC) layer still encounter challenges. While AQM methods, such as CoDel and CAKE, work well to manage congestion by prioritizing signals and reducing delay at the queue level, they could be limited in their responsiveness. These methods rely on managing traffic within the main network stack, which can lead to delays in high-throughput or latency-sensitive environments as packets still need to traverse several layers of the stack.

To overcome these limitations, further enhancements in network efficiency and latency reduction can be achieved with early traffic filtering and processing by using eXpress Data Path (XDP) [7]. XDP handles packet processing directly within the network driver, allowing it to filter and manage packets even before they reach the main network stack. By processing packets at an early stage, XDP minimizes delays that typically occur as packets move through the network stack, providing benefits crucial for remote operations, like low latency and high throughput. When combined with active queue management (AQM) techniques, such as CoDel and CAKE, XDP becomes even more effective at handling congestion. AQM techniques allow the network to prioritize critical signals at the queue level, actively managing delays and reducing the chance of packet loss. Together, XDP and AQM enable more reliable, stable, and efficient performance, ensuring that critical signals are delivered without significant delays.

The thesis aims to improve communication quality in remote control operations, specifically focusing on enhancing video quality and synchronizing video and haptic signals. To achieve these goals, a remote control testbed was designed and built, providing a controlled environment to evaluate network optimizations. Using this setup, a method was proposed to drop packets at the XDP layer to reduce congestion before packets enter the network stack. This approach enhances video communication quality and improves the synchronization of video and haptic streams, which are essential for effective remote control.

The remainder of this thesis is organized as follows: Section 2 introduces background knowledge and related work. Section 3 details the testbed's implementation and the principles of the algorithm. Section 4 evaluates the performance metrics of both the testbed and the algorithm. Section 5 discusses limitations and suggests directions for future work. Finally, Section 6 concludes the thesis with a summary of the findings.

2 Background and Related Work

2.1 Multimedia synchronization

Multimedia communication typically refers to the method of transmitting multiple types of signals, including visual, auditory, haptic, and others, which are often used simultaneously in applications like teleoperation, virtual reality, and interactive simulations. These signals are captured and transmitted with specific temporal relationships at the sender side, creating a synchronized experience for the user. Ideally, these temporal relationships should be preserved at the receiver side to ensure smooth and realistic interactions. However, due to factors such as network congestion and processing delays at either the sender or receiver end, time delays may occur between these signals, disrupting their synchronization. When signals arrive out of sync, it can lead to a degraded Quality of Experience (QoE), making interactions feel unnatural or disjointed [8].

Research on human sensitivity to media asynchrony, such as the study in [9], has highlighted the perceptual thresholds for detecting asynchronization, specifically in scenarios where temporal shifts exist between haptic and video signals. The statistical results from this study indicate that users generally begin to perceive asynchrony around a delay of 100 milliseconds. In tests involving video and haptic signals, the average threshold for detecting asynchronization was found to be 125 milliseconds when the haptic signal lagged behind the video. Conversely, asynchrony was noticeable at 87 milliseconds when the haptic signal led the video. These findings underscore the importance of maintaining close synchronization between different signal types, as noticeable delays can disrupt the user's experience. Additional studies [10, 11] further demonstrate that when temporal delays between haptic and video signals are minimized, users experience improved performance and greater task accuracy. Thus, maintaining synchronization not only supports a more smooth multimedia experience but also directly impacts user performance and satisfaction.

Haptic-video time synchronization can be achieved at various points in the transmission chain: on the sender side, at the receiver side, or within the network.

On the **sender** side, synchronization is commonly implemented through protocols or multiplexing (mux) techniques that package video and haptic data into a single data packet. By packaging video, audio, and haptic signals together as they are captured, temporal synchronization is maintained from the outset. For example, a study proposes that video, audio, and haptic data captured at the same moment be placed into a single packet to ensure temporal synchronization [12]. Multiplexing tools, such as the Admux, integrate these signals with timestamps, ensuring that multiple data streams are synchronized during transmission [13–15]. This approach has been practically applied in systems like Slingshot 3D [16], where the encapsulation of multiple signals in a single data packet allows for consistent timing across media types.

On the **receiver** side, synchronization is often managed through buffering techniques designed to handle variations in packet arrival times. Timestamp-based methods are frequently employed here, tagging each packet so that the receiver can reorder and synchronize playback accurately [17]. A timestamp-independent synchronization method has also been proposed, which allows synchronization of haptic and visual signals by analyzing key moments in interactions, such as object collisions, through machine learning [18]. This approach allows for real-time adjustments, optimizing synchronization for the receiver.

For the approach implemented in the **network** part, synchronization is facilitated through advanced packet prioritization and scheduling techniques. For example, the SynCoDel algorithm, an enhanced version of the CoDel algorithm implemented on the P4 platform, achieves synchronization by dynamically prioritizing video packets based on their requirements for alignment with haptic streams [19]. By utilizing a synchronization window to match event pairs and reallocating bandwidth originally reserved for haptic data, SynCoDel minimizes asynchrony, even in congested network environments. This network-level synchronization assists in maintaining consistent timing for haptic-video interactions despite potential network delays.

There are several key advantages to implementing synchronization at the network layer. First, synchronization algorithms on the sender and receiver sides are often affected by network conditions, and many rely on network measurements or predictions to achieve synchronization. This reliance on network feedback introduces additional overhead and latency, which can reduce efficiency. By implementing synchronization directly at the network layer, we can reduce or eliminate the need for separate measurement and prediction steps, streamlining the synchronization process. This approach enables faster and more responsive synchronization, as it addresses timing and latency issues directly within the network, thereby improving performance for applications where precise timing is critical.

Currently, few studies specifically address the synchronization of video and haptic data at the network layer. While SynCoDel optimizes the CoDel algorithm in the Traffic Control layer, it does not extend to the XDP layer, which could further enhance performance. Additionally, SynCoDel's evaluation was limited to emulation using a Raspberry Pi and a P4 switch, without the establishment of a data collection testbench or validation of performance on actual network hardware.

2.2 Quality of Service (QoS)

Quality of Service (QoS) is crucial in modern network services due to the growing demand on high-speed data transmission and the need for high Quality of Experiences(QoE). It involves evaluating the overall effectiveness of services like mobile phone systems, computer networks, or cloud computing by examining key performance metrics such as transmission delay, throughput, bit rate, packet loss, availability, and jitter. These metrics directly impact the end user experience, making reliable QoS essential for improving Quality of Experience (QoE), as better network performance ensures smoother, disruption-free usage, ultimately enhancing user satisfaction and efficiency.

To further enhance Quality of Service (QoS), the Internet Engineering Task Force (IETF) has developed numerous protocols aimed at improving the reliability, stability, and flexibility of data transmission. These protocols are essential in ensuring that network services can meet the growing demands for higher performance and better user experiences. For instance, the Differentiated Services (DiffServ) protocol uses a 6-bit Differentiated Services Code Point (DSCP) within the IP header for packet classification. This classification system allows network traffic to be prioritized and managed efficiently across the network, ensuring that more critical data receives the necessary resources [20]. Additionally, the Resource Reservation Protocol (RSVP) enables hosts and routers to request and provide specific levels of Quality of Service for application data streams, thereby helping to maintain performance consistency for high-priority

data in demanding network environments [21]. These IETF protocols are practical solutions to the challenge of maintaining high QoS, directly addressing the network's need to prioritize essential data. This improves the overall Quality of Experience (QoE) for users by reducing disruptions, minimizing latency, and enhancing the smoothness of real-time applications such as video conferencing and online gaming.

In haptic video communication systems, the Quality of Service (QoS) requirements differ significantly between haptic and video components due to their distinct roles and sensitivity to delay. Haptic feedback, where precise, real-time interaction is crucial, has a strict jitter tolerance of less than 5 ms to prevent any noticeable delay in response. In contrast, video can tolerate jitter up to 30 ms without significantly impacting visual continuity. Similarly, the delay requirement for haptics is stringent, needing to be below 30 ms to maintain an immersive and realistic user experience, while video has a more lenient threshold of up to 400 ms. This contrast reflects the need for immediate responsiveness in haptic feedback, as even slight delays can disrupt the sense of touch and immersion. On the other hand, video can handle slightly higher latencies, as visual information is less sensitive to minor delays. Additionally, the update rate for haptic signals must be over 1000 Hz to ensure smooth, continuous feedback, whereas an update rate of around 30 Hz suffices for video, aligning with standard frame rates for visual clarity [3].

To further optimize haptic data transmission, many studies have explored ways to reduce the data load by adjusting sampling frequency based on the perceptual limitations of human haptic sensation [22–26]. This method, known as adaptive sampling, leverages the "Just Noticeable Difference" (JND) metric, which identifies haptic samples that are imperceptible to human users. By selectively transmitting only those samples that exceed the JND threshold, the system avoids sending data that would be indistinguishable to the user. This results in a compressed haptic signal that maintains the user's perception of quality without transmitting unnecessary data. Such selective transmission significantly reduces the data rate needed for teleoperation, enabling more efficient use of network resources while maintaining a high-quality user experience.

In addition to haptic data optimization, studies have also examined how frame rate (FPS) and quantization parameter (QP) affect perceived quality in video streams, particularly within haptic video applications. As shown in Figure 2.1, Mean Opinion Score (MOS) values were analyzed across various source sequences at different QP values and resolutions (480p and 1080p), revealing the critical roles FPS and QP play in balancing video quality and resource efficiency. A key finding from these analyses is the substantial improvement in MOS values when the frame rate increases from 5fps to 30fps. This increase in FPS provides a high marginal benefit by significantly enhancing user experience at lower frame rates, leading to smoother motion and clearer detail. However, as FPS rises beyond 30fps, the rate of improvement slows, indicating a point of diminishing returns where additional frame rate increases offer minimal perceptual gains. This suggests that while higher frame rates can further enhance quality, they may not always justify the added bandwidth and computational resources required [27].

Supporting these findings, a comprehensive review of over 50 studies on FPS and human performance demonstrates that 15 FPS is often sufficient for many perceptual tasks, balancing visual quality and efficiency. Acceptable performance can still be achieved at 10 FPS, suggesting this rate as a potential threshold for applications where resource constraints are high. Even 5 FPS, though generally marginal, is found to provide a baseline for watchability and basic video quality in certain applications, making it a feasible option for low-bandwidth or low-power scenarios [28].

These insights into FPS and perceived quality highlight the importance of setting QoS requirements that match the specific needs of haptic video applications. By limiting FPS to levels that balance user satisfaction and efficient network use, systems can save bandwidth without lowering the perceived quality of the video. Combined with adaptive sampling for haptic data,

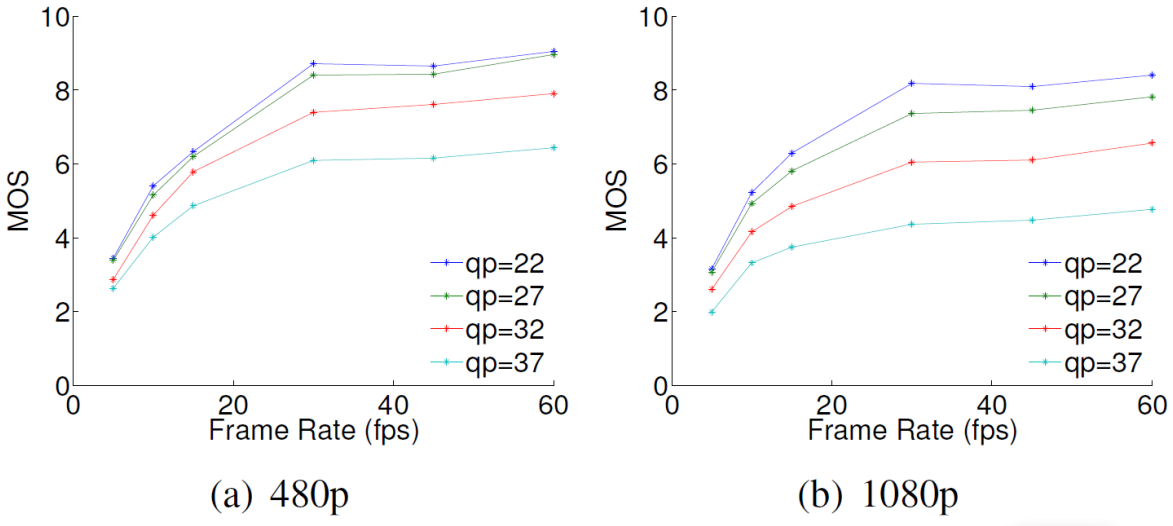


Figure 2.1: Relationship Between Video Frame Rate and MOS Scores at Different Resolutions and QP in a Study of 25 Participants (taken from [27]).

which reduces unnecessary data, this approach allows for more efficient network use while still providing a high-quality experience for users.

This diverse exploration of QoS across different application scenarios sets the stage for the next section, which will delve into Active Queue Management (AQM). AQM is an essential network strategy that directly addresses latency and throughput challenges, specifically by mitigating issues like buffer bloat, where excessive buffering can lead to high latency and poor QoS. By managing network queues effectively, AQM techniques aim to reduce latency and improve overall network performance, enhancing the stability and responsiveness of real-time applications. The next section will discuss how AQM can complement existing QoS strategies, like adaptive sampling and FPS optimization, by providing additional control over data flow, thereby improving the end-user experience in haptic video communication systems.

2.3 Active Queue Management (AQM)

The exploration of Quality of Service (QoS) in the previous section laid the foundation for understanding how various factors like delay, packet loss, and throughput impact user experience across different applications. These factors are particularly important in real-time systems, where delays or lost packets can disrupt the flow of communication and degrade the Quality of Experience (QoE). To further enhance QoS, especially in terms of reducing latency and minimizing packet loss, Active Queue Management (AQM) plays a crucial role in managing network congestion proactively.

Active Queue Management (AQM) techniques are designed to mitigate network congestion by improving network utilization, limiting packet loss, and reducing delay. AQM achieves these goals by controlling the flow of data packets through network queues, thereby promoting a balanced and efficient use of network resources. By keeping queue sizes manageable, AQM helps prevent excessive buffering, which can lead to increased latency, or *buffer bloat*, a common issue in congested networks that negatively impacts QoS.

The development of Active Queue Management (AQM) began with the introduction of the Random Early Detection (RED) algorithm [29], a foundational approach to congestion control that uses queue length as an indicator of congestion. RED monitors the average queue size and selectively drops packets before the queue becomes full, signaling the sender to reduce

its transmission rate. This early intervention helps avoid sudden congestion and keeps data flowing smoothly. By dropping packets based on queue length, RED controls congestion more dynamically than traditional tail-drop methods, which only drop packets once the queue is completely full. This proactive approach reduces the risk of packet bursts and sudden bottlenecks, leading to a more stable flow of data across the network.

Over time, several variants of the RED algorithm were developed to address its limitations and improve performance in diverse network environments. Among these, a widely used variant is Adaptive Random Early Detection (ARED) [30], which dynamically adjusts its parameters to adapt to changing network conditions. ARED improves upon RED by tuning key parameters, such as packet drop rates, in real time, making it more responsive to fluctuations in network traffic. This adaptability helps ARED perform more effectively in real-time applications where network conditions vary frequently, allowing for smoother, more consistent data transmission.

In addition to RED-based schemes, entirely new AQM approaches were introduced to handle congestion through different mechanisms, expanding the toolkit for managing network performance. For example, CoDel (Controlled Delay) [5] is a delay-based AQM algorithm specifically designed to minimize queue latency. Rather than focusing on queue length alone, CoDel aims to keep latency low by adjusting packet drops based on delay targets, making it particularly effective in applications where low latency is critical. An enhanced version, known as fq-CoDel (FlowQueue-CoDel) [31], further improves performance by ensuring fairness across multiple data flows, distributing bandwidth more equitably among users.

Another innovative approach is the BLUE algorithm [32], which controls packet loss rates rather than queue length or delay. BLUE dynamically adjusts packet marking and dropping probabilities based on observed packet loss and queue behavior, providing a different perspective on congestion management that focuses on reducing packet loss to improve overall throughput. Additionally, CAKE (Common Applications Kept Enhanced) [6] is a comprehensive AQM solution designed specifically for home Internet gateways. CAKE integrates multiple features, such as bandwidth shaping, per-user fairness, and support for DiffServ, to enhance performance in consumer networks where multiple applications and users compete for bandwidth.

These alternative AQM algorithms, from RED and its variants to entirely new methods like CoDel, BLUE, and CAKE, offer diverse strategies for managing network congestion. Each approach provides unique advantages in specific scenarios, allowing network administrators to adjust congestion control to the requirements of various applications. For instance, CoDel and fq-CoDel are particularly effective in minimizing latency-sensitive applications, while BLUE focuses on controlling packet loss to maintain throughput. By leveraging these varied AQM tools, network administrators can more effectively maintain network performance and Quality of Service (QoS), supporting a smoother and more reliable user experience, even in demanding network environments.

While Active Queue Management (AQM) plays a critical role in mitigating congestion and enhancing QoS by reducing delay and packet loss, it is not a standalone solution for all network performance challenges. AQM primarily focuses on managing queues to prevent congestion, but it does not directly control the rate at which data is transmitted. In many cases, this is not enough to fully manage network flow, as sudden bursts of data can still overload the network. This is where the Token Bucket (TB) mechanism becomes essential. TB is a bandwidth management method that regulates traffic by setting an average transmission rate and permitting controlled bursts of data within a predefined limit [33]. By ensuring that traffic flows adhere to bandwidth limits, TB complements AQM by adding an additional layer of control over traffic rates, which helps prevent excessive bursts that could otherwise lead to congestion and network instability.

The integration of AQM and TB creates a more comprehensive approach to traffic management, with AQM addressing latency and congestion issues and TB providing rate-limiting and

fairness. This combined strategy is particularly valuable in environments where traffic prioritization and bandwidth allocation are crucial for sustaining optimal network performance. For example, in real-time applications like video conferencing or online gaming, this dual approach helps prevent bottlenecks while ensuring critical data streams receive prioritized transmission. Together, AQM and TB offer a balanced framework for traffic management that can lead to significant improvements in both network efficiency and user experience.

As network requirements continue to evolve, newer technologies are emerging to offer even more granular and flexible traffic management solutions. One such technology is the extended Berkeley Packet Filter (eBPF), which enables dynamic and programmable control over network traffic at the kernel level. By allowing custom scripts to be loaded into the kernel, eBPF extends traditional AQM capabilities and offers a powerful tool for handling network performance challenges in real time. In the next section, we will explore how eBPF can be leveraged to create sophisticated traffic management strategies that complement AQM and TB, providing network administrators with unprecedented control over data flow and QoS.

2.4 Extended Berkeley Packet Filter (eBPF)

The Berkeley Packet Filter (BPF) is a packet filtering system introduced in 1992, designed to capture and filter network packets at the operating system level [34]. By enabling user-space processes to supply filter programs, BPF allows these processes to specify which packets they want to receive. This targeted approach reduces system overhead by focusing only on selected packets, making BPF an efficient tool for network monitoring and diagnostics.

Building on this foundational technology, extended BPF (eBPF) represents a significant advancement. eBPF retains BPF's original packet filtering capabilities but goes further by enabling the execution of small programs within privileged contexts, such as the operating system kernel. This ability to modify kernel functionality dynamically and safely at runtime—without needing to alter kernel source code or compile new kernel modules—makes eBPF a flexible and powerful tool for enhancing kernel behavior [35]. eBPF's ability to extend kernel functionality efficiently has opened up a range of new applications in areas such as security, traffic management, and system monitoring.

One of eBPF's primary advantages is its ease of development. Unlike traditional kernel modules, eBPF programs can be written, tested, and deployed independently of the kernel itself, following a development process similar to that of user-space applications. This separation simplifies the workflow, allowing developers to implement and iterate on kernel-level solutions without the need to recompile or update the kernel. As a result, eBPF supports faster, more flexible development cycles.

In terms of performance, eBPF achieves high throughput by leveraging just-in-time (JIT) compilation, which allows eBPF programs to be executed natively on the target hardware. This capability is particularly advantageous for packet processing at the eXpress Data Path (XDP) level, where eBPF programs operate at near hardware-level speeds, providing performance that rivals traditional user-level solutions [36]. By processing packets at such a low level, eBPF can handle high-speed data flows effectively, making it ideal for real-time network applications.

eBPF also stands out for its efficient resource usage. Unlike polling-based methods, such as DPDK (Data Plane Development Kit), which consume CPU resources continuously, eBPF programs only use CPU cycles when there is actual traffic to process. This on-demand resource usage leads to lower overall system consumption, freeing up resources for other processes and enabling more sustainable scaling for high-traffic environments.

Furthermore, eBPF integrates seamlessly with the Linux kernel's TCP/IP stack and other subsystems, ensuring compatibility with legacy applications and existing debugging tools. This integration minimizes the need for extensive reconfiguration, making eBPF adaptable for existing systems while providing modern functionality. The seamless Linux integration also means that

administrators can use familiar monitoring and debugging tools to manage and analyze eBPF programs, reducing the learning curve.

Security is another critical advantage of eBPF. Each eBPF program must pass through an in-kernel verifier that checks for safety and correctness, preventing faulty or potentially harmful operations from being executed. This in-built verification process ensures that eBPF programs do not compromise the stability or security of the system, making it a safe solution for extending kernel functionality.

Together, these advantages make eBPF a highly efficient, flexible, and secure solution for modern network environments. Its capabilities enable network administrators and developers to optimize traffic management, monitor system performance, and enhance security in real-time, all within the kernel's privileged context. This high level of control and integration makes eBPF an ideal choice for tasks that require rapid and reliable system-level data processing.

The workflow of eBPF is illustrated in Figure 2.2, which outlines the process from program creation to execution within the kernel. Typically, the workflow begins with writing an eBPF program in C. This code is then compiled using Clang, producing an ELF (Executable and Linkable Format) or object code file that is suitable for kernel execution. A loader then inserts this compiled code into the Linux kernel, where it passes through an in-kernel verifier that ensures the program's safety and correctness. Once verified, the program undergoes Just-In-Time (JIT) compilation, translating it into executable code for the target platform [36]. This JIT compilation step allows eBPF programs to run with near-native performance, making them both efficient and highly responsive to network and system events.

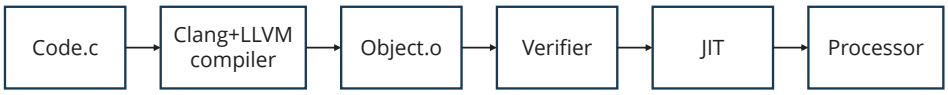


Figure 2.2: eBPF Workflow: From Program Creation to Kernel Execution.

Despite its flexibility, eBPF programs have certain constraints due to their operation within the kernel. For example, eBPF programs can only use a limited set of C libraries, must avoid non-static global variables, are restricted to bounded loops, and have a stack size limit of 512 bytes [36]. These limitations are necessary to maintain system stability and prevent excessive resource consumption within the kernel. However, ongoing efforts to improve eBPF's usability are being pursued by various projects, including IOVisor [37] and VMware's P4C-XDP [38], which aim to extend eBPF's capabilities while retaining its safety features. Additionally, tools like the BPF Compiler Collection (BCC) [39] and Cilium [40] offer APIs for programming eBPF in languages such as Python 3 and Go, making eBPF development more accessible and lowering the entry barrier for new developers.

An important feature of eBPF is its use of *maps*, which are generic key-value data structures that facilitate data exchange between the kernel and user-space processes. eBPF maps handle keys and values as raw binary data, making them adaptable for storing a wide variety of custom data structures, as long as their sizes are specified when the map is created. These maps allow user-space programs to create and interact with multiple storage containers, which can then be accessed by both user-space applications and eBPF programs running in the kernel. When an eBPF program is loaded, the required maps are instantiated through a system call, with file descriptors passed to the eBPF program to enable efficient data sharing between user space and the kernel. Different types of eBPF maps are available, each designed to resemble various data structures (e.g., hash tables, arrays) to meet specific application needs [35,36,41]. This flexibility in data handling allows eBPF programs to store and retrieve state information, making it possible to manage real-time data for tasks such as monitoring, traffic control, and security enforcement.

The process of compiling and loading an eBPF program involves several components work-

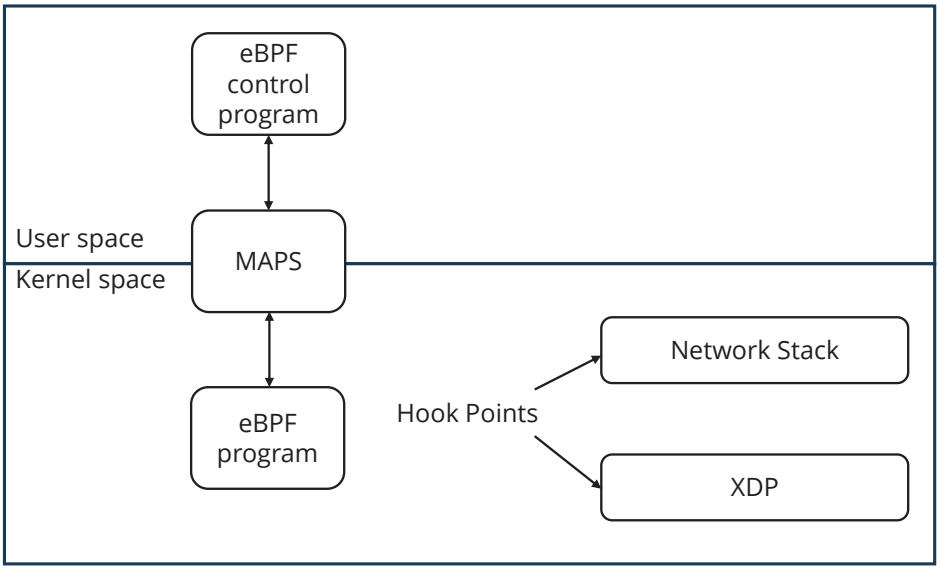


Figure 2.3: Architecture of eBPF Programs in Linux.

ing in tandem. As shown in Figure 2.3, an eBPF program compiled with Clang and LLVM is verified and then loaded into the kernel. It can be attached to various hook points, such as XDP or TC, and it can communicate with user-space programs via eBPF maps. These maps provide the foundational mechanism for data exchange, enabling both real-time kernel instrumentation and effective interaction with user-space programs. User-space programs can be written in high-level languages, further simplifying the integration between kernel-level monitoring and application-level logic.

2.5 Express Data Path (XDP)

Starting from version 4.8, the Linux kernel introduced eXpress Data Path (XDP) [42], a high-performance data path positioned at the lowest layer of the kernel. Integrated directly within the network driver to manage RX (receive) data, XDP enables ultra-fast packet processing by intercepting data even before it reaches the traditional network stack. This early-stage intervention provides several substantial benefits, including high throughput, reduced latency, and optimized CPU usage, making XDP ideal for high-performance and low-latency networking scenarios [7].

One of the core advantages of XDP is its integration with eBPF (extended Berkeley Packet Filter), which provides a programmable interface that allows developers to customize packet handling without altering the core kernel. Through eBPF, developers can write small, efficient programs that define how packets are processed, enabling flexible and powerful packet handling that can adapt to a range of networking needs. This flexibility makes XDP not only powerful but also adaptable, as eBPF can evolve to meet new use cases without requiring kernel modifications.

XDP offers a variety of actions for handling incoming packets, allowing developers to tailor packet processing based on application requirements. For example, they can choose to drop packets with an exception (XDP_ABORTED), silently discard packets (XDP_DROP), redirect packets to other network interfaces (XDP_REDIRECT), allow packets to proceed up the network stack for conventional processing (XDP_PASS), or retransmit packets immediately through the same interface (XDP_TX) [36]. This level of control means developers can use XDP to finely tune network performance based on their specific workload, enhancing both security and efficiency.

XDP has already demonstrated its value in several high-impact scenarios. Cloudflare, for instance, employs XDP to mitigate Distributed Denial of Service (DDoS) attacks by filtering malicious traffic at the earliest possible point, reducing the load on subsequent network layers and preventing potential overloads [43]. Similarly, Facebook leverages XDP for Layer 4 load balancing, distributing incoming traffic across servers with minimal latency, which is critical for maintaining high availability and scalability in large data centers [44]. These applications showcase how XDP can handle intensive, large-scale network demands efficiently.

In addition to Linux, the versatility of XDP has led to its adoption in other environments as well. Notably, starting in 2022, Windows began integrating an XDP module, signaling the growing recognition of XDP's benefits in diverse operating systems and its potential to become a cross-platform standard for high-performance packet processing [45]. This expansion demonstrates that the principles behind XDP—early packet processing, programmability, and efficient resource usage—are applicable beyond Linux and can bring similar benefits to other systems.

In summary, XDP represents a significant advancement in network processing technology, combining high performance, flexibility, and extensibility. Through its integration with eBPF, XDP enables developers to manage network traffic with unprecedented efficiency and control, making it a powerful tool for modern, performance-intensive networking environments.

2.6 WebRTC

WebRTC (Web Real-Time Communication) is a free, open-source project that enables real-time communication (RTC) capabilities directly in web browsers and mobile applications through a set of standardized application programming interfaces (APIs) [46]. By offering a way for developers to incorporate audio and video communication, as well as data streaming, directly within web pages, WebRTC allows for direct peer-to-peer connections without the need for external plugins or native application downloads. This setup not only simplifies the user experience by operating seamlessly within the browser but also ensures low latencya and efficient communication, a key requirement for real-time interactions.

Supported by leading tech companies such as Apple, Google, Microsoft, and Mozilla, WebRTC's specifications are established by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF) [47,48]. These organizations set rigorous standards, ensuring that WebRTC remains a reliable and widely compatible solution for developers seeking to create interactive, real-time web and mobile applications. The strong backing and standardized approach make WebRTC a dependable choice across platforms.

One of the main advantages of WebRTC is its ability to deliver low latency in real-time applications, thanks to its direct, peer-to-peer communication framework. This is especially beneficial for use cases like video calls, live streaming, and online gaming, where even minor delays can disrupt the user experience. Additionally, WebRTC is designed for broad compatibility, supported by all major browsers (such as Chrome, Firefox, Safari, and Edge) and across various operating systems [49]. This compatibility allows seamless integration across a wide range of devices, which is critical in a diverse technology ecosystem. Security is another significant advantage; WebRTC enforces encryption by default on all media and data streams, ensuring secure communication without requiring additional configurations.

Furthermore, WebRTC enhances user accessibility by eliminating the need for plugins or native applications, allowing users to connect directly through their browsers. This simplicity and ease of use make it accessible to a wider audience, even those unfamiliar with installation processes. Adding to its versatility, WebRTC supports data channels in addition to audio and video streams, which opens the door to features like file sharing, chat, and real-time data transfer. This flexibility makes WebRTC suitable for a wide array of applications beyond traditional audio and video calls.

To support high-quality communication across devices, WebRTC incorporates several video codecs. The primary codec, VP8, strikes a balance between performance and compression efficiency, making it well-suited for most real-time applications. For those needing higher quality at lower bitrates, WebRTC also supports VP9, an advanced codec offering superior compression. Additionally, the widely adopted H.264 codec is supported, which offers hardware acceleration on many devices. This hardware support improves performance and compatibility across platforms, allowing WebRTC to operate smoothly on a diverse range of systems [50, 51].

Overall, the combination of broad compatibility, robust security, ease of use, and flexibility in codec support makes WebRTC a powerful and accessible tool for real-time, interactive communication in both web and mobile applications.

3 Method

3.1 Problem analyze

In traditional networks, all packets are treated equally, with network queue management typically following a first-come, first-served (FCFS) approach, as shown in Figure 3.1. When the queue reaches its capacity, any newly incoming packets are simply dropped behavior known as tail drop. This default approach presents several significant challenges for time-sensitive applications, especially in scenarios like remote operations, where different types of data packets have varying levels of importance.

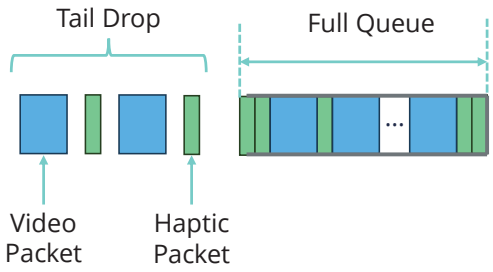


Figure 3.1: Tail Drop Occurring in a Single FIFO Queue Under Congestion.

One major limitation of the FCFS method is that it treats all packets equally, regardless of their importance to the user. This lack of differentiation means that critical packets, such as those carrying haptic feedback or control signals, may face unnecessary delays if queued alongside less time-sensitive data. For latency-sensitive applications, these delays can severely impact task performance and user experience. Furthermore, when the queue is at full capacity, any packets that manage to enter experience substantial queuing delays due to network congestion, which further impacts the timeliness of data delivery.

As a way to control congestion, tail drop is often ineffective, frequently causing bursts of packet loss during high levels of network traffic. This packet loss can increase delays, cause irregular data streams, and reduce overall network performance, especially in situations where reliable packet delivery is crucial. This issue is especially challenging for multimedia applications, where packet loss in a video stream can interrupt the experience, and for haptic feedback, where even small delays can break the sense of real-time control and quick response.

To address these limitations, modern networks have adopted a differentiated approach to traffic management, prioritizing packets based on their specific requirements. For instance, in scenarios where both haptic and video streams are involved, these streams are separated into

different priority queues, as illustrated in Figure 3.2. The haptic stream, which is highly sensitive to latency, is assigned to a high-priority queue, ensuring minimal delay and reducing latency to below 5 milliseconds, thereby providing a responsive and immersive user experience. Video data, on the other hand, is treated as low-priority traffic since it can tolerate slightly higher latencies without critically impacting the overall task.

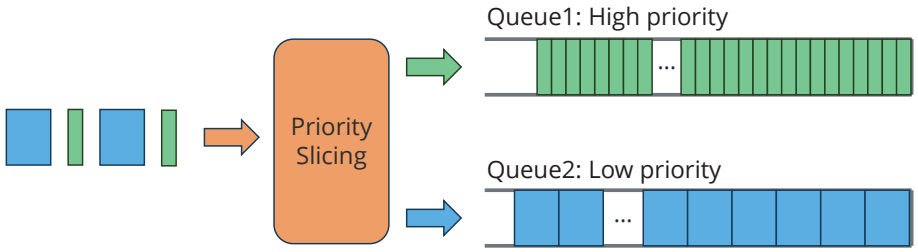


Figure 3.2: Segmentation of High and Low Priority Traffic via Priority Policing.

In our setup, video data transmission leverages WebRTC and is encoded using the VP8 codec, where each frame is divided into multiple packets. This structure, however, introduces an added challenge in maintaining video quality, as the loss of even a single packet renders the entire frame undecodable, as shown in Figure 3.3. Consequently, the frame loss rate can quickly exceed the packet loss rate, making it challenging to ensure consistent video quality, particularly under lower priority conditions in a busy network.

To address this, we can leverage specific properties within the RTP (Real-time Transport Protocol) stream, specifically the marker bit, which is set to 1 in the last packet of each frame, while all other packets in the frame have a marker bit set to 0. By using this information to identify complete frames, we can develop more effective packet drop strategies. Instead of losing isolated packets within a frame, which causes the entire frame to be lost, we can selectively discard all packets associated with a frame when necessary. This approach optimizes network performance while preserving the quality of other critical data streams, such as haptic feedback.

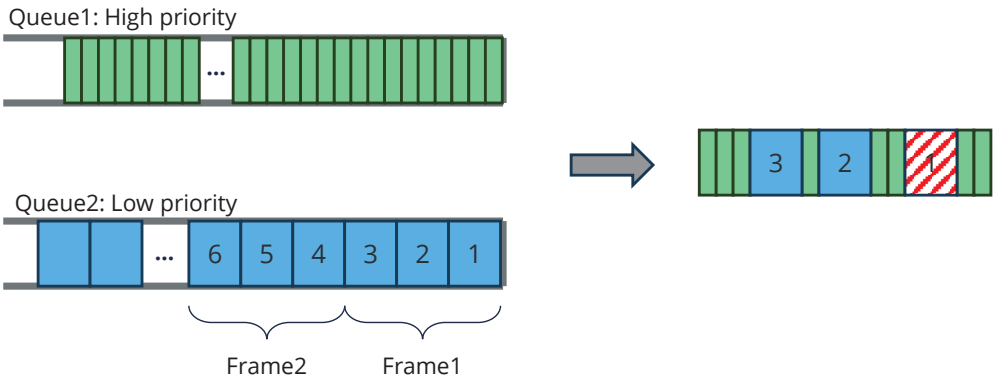


Figure 3.3: Impact of Packet Loss in VP8 Encoding: Entire Frame Becomes Undecodable Upon Loss in Any Packet.

To implement this selective dropping mechanism, we plan to deploy an algorithm via eBPF on XDP to discard entire video frames, specifically a sequence of consecutive RTP packets, when network conditions make it advantageous. XDP, positioned at the lowest level of the kernel's network stack, offers significant performance benefits and low CPU utilization, making it an ideal choice for deployment on routers and other network devices. As shown in Figure

3.4, when packets are received from the Network Interface Card (NIC) and enter the device driver, XDP examines whether to drop all packets associated with a specific video frame. If the algorithm determines that the frame should be dropped, none of its RTP packets proceed to the network stack, thus conserving bandwidth and reducing unnecessary processing.

For frames that are allowed to proceed, packets move up to the network stack, where they are subject to further processing and management. The packets are organized into separate queues according to their priority and managed using Active Queue Management (AQM) techniques to handle congestion effectively. To adapt to changing network conditions dynamically, the eBPF control logic in user space monitors the packet drop rate and continuously adjusts parameters within the XDP program using eBPF maps. These real-time adjustments influence the type and volume of packets entering the network stack, as shown by the red arrow in Figure 3.4, thus forming a closed-loop control system. This adaptive strategy ensures that XDP can optimize packet handling dynamically, balancing the needs of high-priority haptic feedback with efficient video frame management to maintain an optimal user experience in remote operation scenarios.

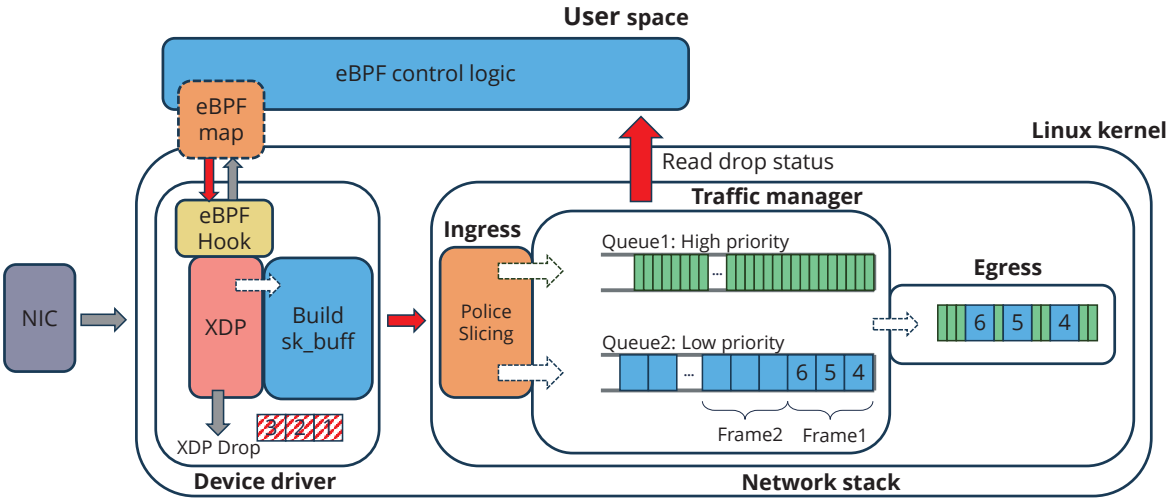


Figure 3.4: Packet Processing Flow in XDP and TC Layers with Adaptive eBPF Control.

3.2 Test bench setup for data collection

To analyze and address the issues, the first step is to set up a test environment for data collection. The devices used in this setup are listed in Table 3.1. Four Fujitsu PCs are used, each assigned a specific role within the system: one PC, called the camera client, captures video of the haptic follower's movement and sends it to another PC, called the camera server, for real-time display. At the same time, the camera server captures video of the haptic leader's movement and sends it back to the camera client for playback. This setup uses WebRTC to enable real-time video exchange, much like a video call, allowing the two PCs to simultaneously send and receive video.

In addition to these PCs, the setup includes two 3D Systems Touch™ devices that act as the haptic leader and follower, each operating at an update rate of 1 kHz to provide continuous, real-time feedback. The camera client uses a Logitech StreamCam, which records at 60 fps, to capture the haptic follower's movements, while the camera server's camera captures the actions of the haptic leader. This arrangement allows both the haptic devices' movements to be visually monitored and provides synchronized video playback on both ends. The haptic

code used in this setup is based on code provided by the TU Munich Chair of Media Technology, ensuring precise haptic feedback control and synchronization across devices.

For network connectivity, the setup uses three TP-Link TX20U Plus external WiFi6 adapters and an ASUS TUF-AX4200 router running OpenWRT to handle data transmission. A Raspberry Pi is configured as a Precision Time Protocol (PTP) server to keep all devices precisely synchronized, with Ethernet cables used to improve the accuracy of PTP synchronization across the network.

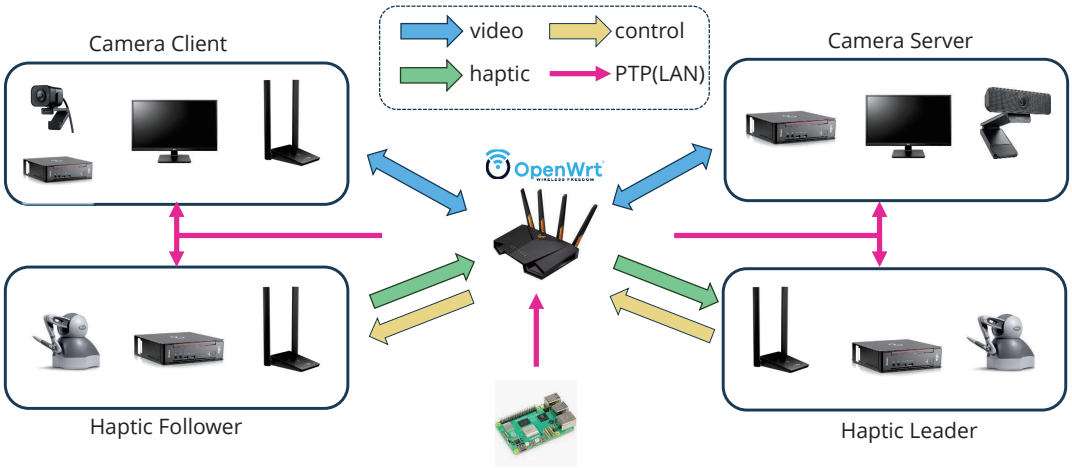


Figure 3.5: Topology of the Data Collection Testbench.

Table 3.1: Devices Used in the Data Collection Testbench.

Device	Description	Quantity
Fujitsu PCs	Camera client: control camera and transmit video data	1
	Camera server: receive video data and play on monitor (with Internal WiFi Adapter)	1
	Haptic Leader: control the movement of haptic stylus	1
	Haptic follower: follow the movement of haptic stylus and return the force feedback to haptic leader	1
3D Systems Touch™	Haptic leader and follower	2
Logitech StreamCam	Camera capture the movement of haptic follower	1
Logitech C925e Webcam	Camera capture the movement of haptic leader	1
TP-Link TX20U Plus	External WiFi6 adapters	3
ASUS TUF-AX4200 Router	Router running on OpenWRT	1
Raspberry Pi	PTP server	1
Ethernet Cables	Ethernet cables for PTP synchronization	Several

Using these devices, the testbench model is illustrated in Figure 3.5. In the upper left section, the camera client captures the movement of the haptic follower and transmits video data over WiFi to the camera server, where it is displayed in real time on a monitor in the upper right. The camera server uses its built-in WiFi hardware, eliminating the need for external adapters.

In the lower left, the haptic follower's movement is guided by the haptic leader on the lower right, which performs tasks and sends tactile feedback back to the follower.

This setup, with each component precisely configured to capture, transmit, and display video and haptic data in real time, enables detailed monitoring and assessment of data flow and device interactions within the network. This configuration not only supports effective real-time feedback but also helps to evaluate the network's ability to handle data-intensive, low-latency applications.

The actual physical setup of the testbench is shown in Figure 3.6. For simplicity, some PC devices and routers are not visible in the image; however, the four primary terminal devices are clearly labeled: the haptic leader, the haptic follower, the camera on the camera client side, and the monitor on the camera server side. In this scenario, the haptic leader remotely controls a stylus to perform a fine-motor task of removing and reattaching a rubber band, as shown in Figure 3.7. Since each action is performed manually, variability is inherent in the data due to differences in execution. To reduce the impact of this variability and to capture a more reliable dataset, each action was repeated 30 times, providing a robust foundation for analysis.

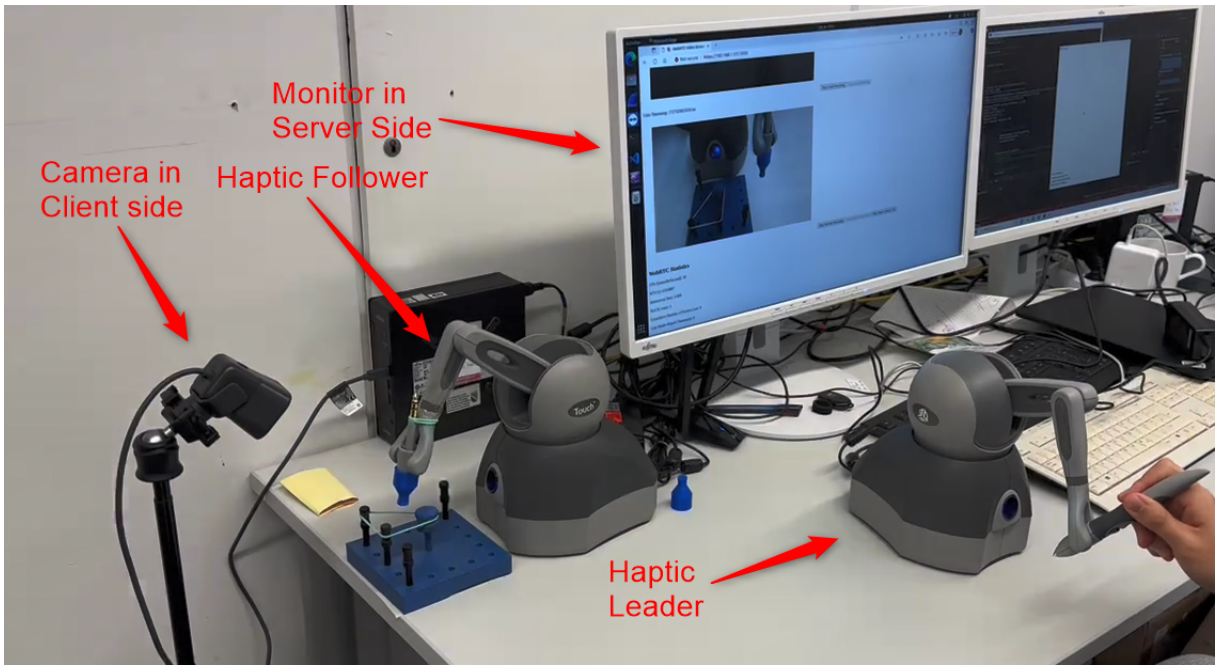
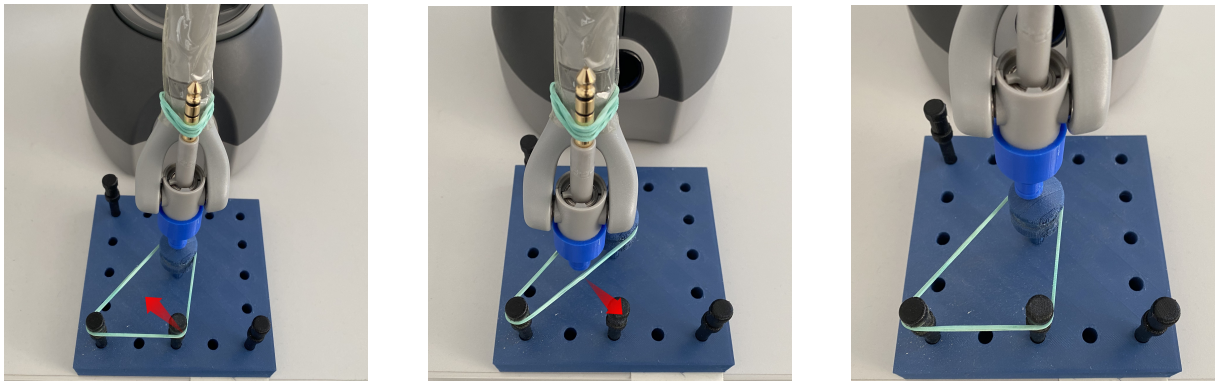


Figure 3.6: Testbench Setup Displaying Only Terminal Devices (Router and Host Omitted).

Table 3.2: Statistics of Collection Packets and Packet Types.

Component	Duration (s)	Total Packets	Packet Count	Packet Type Description
Camera Client	289	153080	52820 86728 13532	Video to Camera Server Video from Camera Server Other Packets
Camera Server	280	142841	81340 49002 12499	Video to Camera Client Video from Camera Client Other Packets
Haptic Leader	381	202020	80303 121067 650	Haptic Data Control Data Other Packets
Haptic Follower	348	201673	121017 80315 341	Control Packets Haptic Data Other Packets



(a) Position 1 (b) Position 2 (c) Position 3

Figure 3.7: Move the rubber band from the position in Figure 3.7a to the position in Figure 3.7b, then move the rubber band back to the position in Figure 3.7c, which is the same as in Figure 3.7a. This sequence is considered one task.

Table 3.2 provides a comprehensive breakdown of the data collected by each device in the testbench, highlighting packet types, counts, and the duration of data capture. The **Camera Client** device recorded a total of 153,080 packets over a duration of 289 seconds. Of these, 52,820 packets represent video data sent from the Camera Client to the Camera Server, while 86,728 packets correspond to video data received from the Camera Server, representing a bidirectional video communication between the two devices. Additionally, 13,532 packets were categorized as other types, primarily consisting of voice data, WebRTC control packets, and PTP packets to facilitate video transmission and synchronization.

The **Camera Server**, over 280 seconds, recorded a total of 142,841 packets, with a similar distribution between video transmission and reception. Specifically, 81,340 packets were transmitted from the Camera Server to the Camera Client, while 49,002 packets were received from the Camera Client, mirroring the bidirectional flow seen in the Camera Client data. The remaining 12,499 packets were classified as other types, mainly including voice data, WebRTC control packets, and PTP (Precision Time Protocol) packets, likely serving supportive roles in maintaining the stability and synchronization of the video feed between the Camera Client and Server.

For the haptic devices, the **Haptic Leader** captured a substantial amount of data, totaling 202,020 packets over 381 seconds. This data includes 80,303 packets identified as haptic feedback data generated for the Haptic Follower, enabling force feedback in the haptic control system. Additionally, 121,067 packets were identified as control data, which likely contain control commands or synchronization data necessary for maintaining the haptic feedback loop. An additional 650 packets consist mainly of PTP (Precision Time Protocol) packets.

Similarly, the **Haptic Follower** recorded a total of 201,673 packets during 348 seconds of operation. Of these, 121,017 packets represent haptic data received from the Haptic Leader, ensuring that the follower accurately mirrors the leader's movements in real-time. Meanwhile, 80,315 packets are designated as control data, enabling the follower to provide feedback and maintain synchronization with the leader device. An additional 341 packets consist mainly of PTP (Precision Time Protocol) packets.

The haptic-video synchronization algorithm is primarily designed around the interaction between the haptic data and the video stream transmitted from the camera client to the camera server. Figure 3.8 illustrates the characteristics of these two data streams, focusing on a 60-second segment to observe their behavior in detail.

From this segment, we can observe that the haptic data exhibits an average rate of approximately 600 packets per second (pps), with variations ranging from 0 to 850 pps. This variability reflects the dynamic nature of haptic interactions, likely driven by user input or environmen-

tal factors affecting haptic response. In contrast, the video data demonstrates a more stable transmission rate, oscillating around an average of 200 pps, suggesting a relatively steady stream with minimal fluctuation. This stability in video data rates supports consistent visual feedback, essential for maintaining a smooth and synchronized user experience in real-time applications.

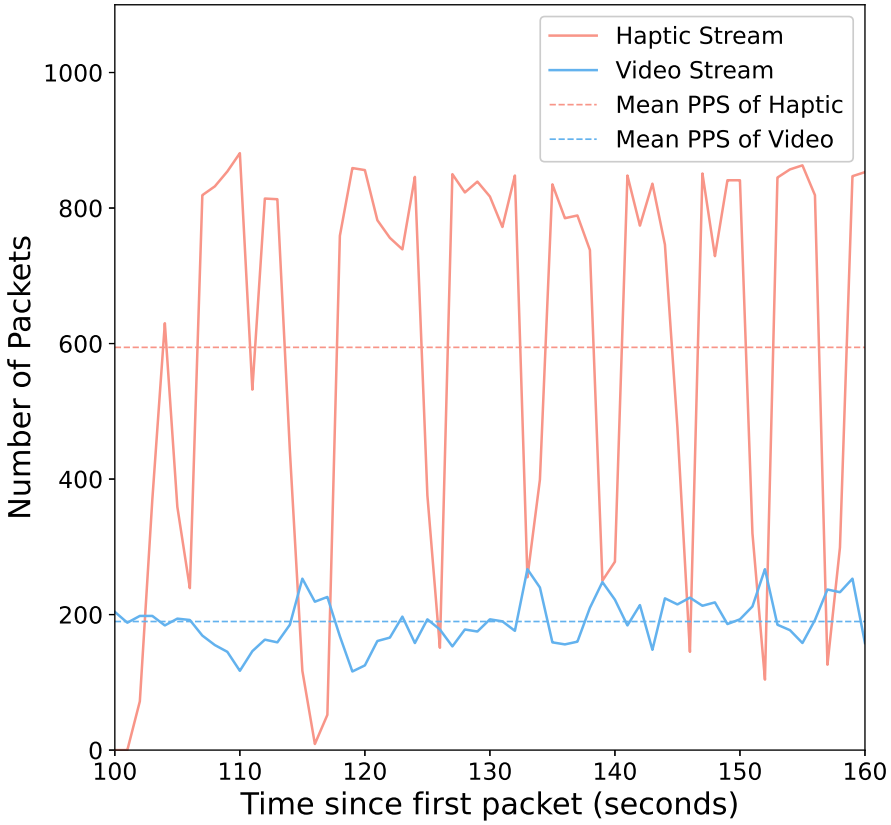


Figure 3.8: Traffic Characteristics of 60-Second Excerpt from Haptic and Video Streams.

3.3 Test bench setup for algorithm Implementation

To facilitate algorithm development and validation, an additional testbench using Raspberry Pis was set up. This approach eliminates the need for manual operation of the haptic stylus during each test, ensuring consistent data across trials and removing any randomness introduced by human interaction. Additionally, by using pre-recorded data for replay, this setup prevents WebRTC from dynamically adjusting its transmission behavior based on round-trip time (RTT) measurements, thereby avoiding the variability that would otherwise affect the data. As a result, this configuration provides a more stable, repeatable, and streamlined environment for validation.

Once data collection on this testbench is complete, the collected data undergoes filtering to prepare it for replay. The data that needs to be filtered is highlighted in red in Table 3.2. Source and destination MAC and IP addresses are modified before the data is replayed on the corresponding Raspberry Pi using tcpreplay. The statistical results of the filtered data are shown in Table 3.3. To maintain the correct timing relationships across different media streams, times-

tamps for the filtered data are carefully aligned, with an error margin kept within 10 ms to preserve temporal accuracy.

The Raspberry Pi topology, illustrated in Figure 3.9, assigns each Raspberry Pi a specific role in the testbench to accurately replicate data replay and collection processes. The Raspberry Pi designated as the camera client replays video data initially captured by the camera client in Figure 3.5, simulating the transmission of video information to the server just as it occurred in the original setup. Meanwhile, the haptic client Raspberry Pi replays the haptic feedback data generated by the haptic follower, allowing for consistent testing of haptic data transmission without the variability introduced by live manual control.

At the core of the testbench setup, the server Raspberry Pi receives both video data from the camera client and haptic feedback from the haptic client. In addition, it also sends control data generated by the haptic leader in Figure 3.5 back to the haptic follower, enabling a full simulation of the original interaction. By assigning each Raspberry Pi a dedicated role, the testbench replicates the actual data flow and interactions of the original setup within a controlled and consistent environment, which is ideal for algorithm testing and validation.

Table 3.3: Statistics of filtered packets.

Component	Total Packets	Duration (s)
Camera Client	43715	241
Haptic Leader	120851	241
Haptic Follower	80075	241

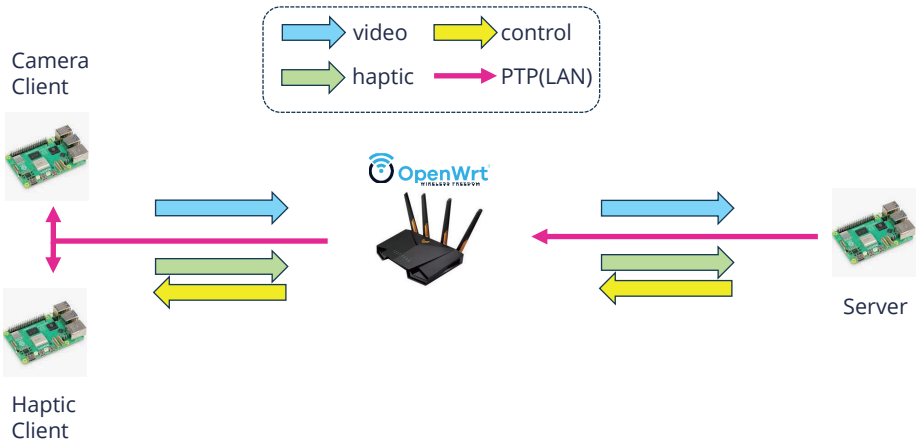


Figure 3.9: Topology of the algorithm implementation testbench

3.4 Algorithm

Building upon the above setup, the algorithm is designed to manage data handling efficiently in real time. It consists of two main components: the XDP (eXpress Data Path) part, which operates at the kernel level for high-speed data processing, and the userspace part, which provides additional control and flexibility. These two components work in tandem, with each part dynamically adjusting specific parameters of the algorithm to respond to current network conditions, ensuring optimal data flow and minimal latency.

The XDP component focuses on low-level packet processing, leveraging its position within the network stack to make fast, critical decisions on whether to pass or drop packets based on real-time measurements. Meanwhile, the userspace component complements XDP by monitoring broader network conditions and fine-tuning parameters over time. Together, these

components enable a responsive and adaptive algorithm, well-suited for handling dynamic network environments.

To clarify the roles and functions of each parameter within the algorithm, Table 3.4 presents key symbols and definitions. For instance, B_{max} represents the maximum measured bandwidth, which helps the algorithm gauge the network's current capacity and adjust the rate limit accordingly. The parameter k, as the proportional control parameter in the P-controller, is essential for regulating adjustments and maintaining stability. Another important parameter, R_{limit}, sets a rate limit per second, controlling data flow to prevent network congestion. C_{byte} tracks the total processed byte count over a given period, which serves as a measure of throughput within the system.

Additionally, the table defines F_{drop}, a flag that indicates when a packet should be dropped in XDP, and M_{RTP}, which identifies markers within the RTP header and helps manage sequences within video frames. The variable P_{size} captures the size of the current packet, a factor influencing rate-limiting decisions, while D_{prev} and D_{curr} track packet drop counts over time, allowing the system to adapt dynamically to packet loss conditions.

Together, these parameters allow the XDP and userspace components to coordinate adjustments in real time, resulting in a highly flexible and adaptive algorithm. This design enables the system to respond smoothly to network fluctuations, optimizing data transmission and ensuring stable, high-quality performance in varying conditions.

Table 3.4: Key variables used in the pseudocode and their definitions.

Symbol	Definition
B_{max}	Maximum measured bandwidth
k	Proportional control parameter (P-controller)
R_{limit}	Rate limit per second
C_{byte}	Total processed byte count
F_{drop}	Drop indicator flag
M_{RTP}	Marker in RTP header, Marker is 1 indicates that this is the last packet of a video frame
P_{size}	Size of current packet
$D_{\text{prev}}, D_{\text{curr}}$	Previous and current dropped packet counts

3.4.1 Algorithm in XDP layer

In Algorithm 1, the XDP section is responsible for processing each incoming packet, deciding whether it should be passed to the network stack or dropped. This decision-making process results in one of two possible outcomes: **PASS** (allowing the packet to continue through the network stack) or **DROP** (discarding the packet). The algorithm starts by initializing key variables on line 3, where $C_{byte} = 0$, $F_{drop} =$ False, and $R_{limit} = Inf$ are set based on values from userspace. The variable C_{byte} is used to keep track of the cumulative size of processed packets and is updated by both XDP and userspace. XDP increments C_{byte} with the size of each new packet, while userspace resets C_{byte} to zero periodically every interval T to control data flow over time. Meanwhile, F_{drop}, which indicates whether packets should be dropped, is exclusively managed by XDP, and R_{limit}, the maximum allowable rate, is adjusted by userspace to control packet throughput.

When a new packet arrives, the algorithm first checks if it meets a predefined filter condition (lines 6-7). If the packet does not match this filter, it is immediately passed to the network stack, minimizing processing overhead for packets that do not require special handling. However, if the packet matches the filter, the algorithm proceeds to evaluate whether F_{drop} is set to **True**

(lines 9-17). When F_{drop} = True, the packet is dropped unless the cumulative byte count C_{byte} plus the incoming packet's size P_{size} remains within the rate limit R_{limit}, and the packet's RTP marker M_{RTP} equals 0. This condition allows the algorithm to override the drop flag if certain criteria are met, ensuring that essential packets still have the opportunity to pass through. If the conditions are met, F_{drop} is reset to **False**, and this updated state is stored in the BPF map. If the conditions are not met, the packet is dropped.

If F_{drop} = False, indicating that the packet is not currently flagged for dropping, the algorithm checks whether the cumulative byte count, $C_{byte} + P_{\text{size}}$, exceeds the rate limit R_{limit} while the RTP marker M_{RTP} is set to 1 (lines 18-20). If this condition is met, F_{drop} is set to **True**, effectively signaling that subsequent packets should be dropped to manage congestion, and this updated state is recorded in the BPF map. Finally, C_{byte} is incremented by P_{size} to reflect the addition of the new packet's size to the cumulative count, and this new value is also updated in the BPF map (lines 22-24). If all conditions allow, the packet is passed to the network stack, ensuring smooth data flow without unnecessary packet loss.

These steps enable the algorithm to dynamically respond to network conditions, balancing data throughput and packet management to achieve efficient processing under varying loads.

Algorithm 1 XDP Algorithm

```
1:  Output: pkt DROP or pkt PASS
2:  while a new packet pkt arrives do
3:      Update C_byte, F_drop, and R_limit from BPF map
4:      Calculate incoming packet size P_size
5:      Check filtering condition for pkt
6:      if packet does not match filter condition then
7:          return pkt PASS
8:      else
9:          if F_drop == True then
10:             if C_byte + P_size > R_limit or M_RTP = 0 then
11:                 return pkt DROP
12:             else
13:                 Set F_drop ← False
14:                 Update F_drop in BPF map
15:                 return pkt DROP
16:             end if
17:         end if
18:         if C_byte + P_size > R_limit and M_RTP = 1 then
19:             Set F_drop ← True
20:             Update F_drop in BPF map
21:         end if
22:         C_byte ← C_byte + P_size
23:         Update C_byte in BPF map
24:         return pkt PASS
25:     end if
26: end while
```

3.4.2 Algorithm in Userspace layer

In Algorithm 2, the userspace algorithm primarily controls the rate limit value in XDP by dynamically adjusting parameters based on network conditions. The algorithm begins by initializing key variables: $C_{byte} = 0$, F_{drop} = False, and $R_{limit} = Inf$, and then updates these values to the

XDP layer for initial setup.

The first function, **estimate_max_bandwidth**, defined in lines 2-12, is used to observe and record the maximum bandwidth over a specified duration. This function initializes B_{max} to zero, then monitors the bandwidth during each interval within the duration. For each interval, the current bandwidth is estimated, and if this value exceeds B_{max}, it is updated as the new maximum observed bandwidth. This recorded maximum bandwidth is returned at the end of the function, providing an estimate of the peak capacity during the monitoring period.

The next function, **packet_loss_increasing_count**, defined in lines 13-21, checks for an increase in packet loss compared to the previous count stored in D_{prev}. It first retrieves the current dropped packet count D_{curr}. If D_{curr} exceeds D_{prev} by more than a specified loss threshold, the difference is calculated as **packet_loss**, D_{prev} is updated to D_{curr}, and the function returns the observed packet loss. Otherwise, D_{prev} is simply updated to D_{curr} without any calculated loss, and the function returns zero.

In line 22, a proportional controller k is initialized. This controller is crucial for adapting the rate limit dynamically based on packet loss trends in the network. In lines 23-25, the algorithm reads the packet loss count from the traffic control (tc) statistics, calculates the maximum observed bandwidth B_{max} over a defined duration using the **estimate_max_bandwidth** function, and checks if packet loss has increased using the **is_packet_loss_increasing** function. If packet loss increases, it indicates that B_{max} represents the upper capacity of the channel, and the algorithm applies this limit in XDP. If no packet loss increase is detected, it suggests that the estimated bandwidth reflects stream characteristics rather than channel capacity limitations, and no rate limit is applied.

In lines 27-31, the value of k is adjusted dynamically based on the packet drop state in the network stack. If packet loss continues to increase, k is decreased slightly to reduce the rate limit, whereas if there is no loss and k is below a specified threshold (e.g., 1.05), it is gradually increased to allow more data throughput. Finally, in line 32, the rate limit R_{limit} is updated to $k \times B_{max}$ in the BPF map, allowing the XDP layer to operate with this dynamically adjusted limit. The algorithm then waits for the next update interval before repeating the process, thus continuously adapting to current network conditions.

Additionally, a separate thread, described in Algorithm 3, is responsible for resetting the byte count C_{byte} at a constant frequency. This thread continuously runs in the background, ensuring that C_{byte} is reset to zero at regular intervals. The reset operation helps maintain accurate tracking of cumulative packet size over each interval, which is essential for controlling the rate limit in the XDP layer based on recent data flow.

In this thread, the algorithm starts by initializing a loop that runs indefinitely. Within each loop iteration, C_{byte} is updated to zero in the BPF map. After performing the reset, the thread waits for a predefined reset interval before repeating the process. By consistently resetting C_{byte} at a fixed rate, this thread enables real-time monitoring of data flow rates, allowing the algorithm to dynamically adapt to network conditions.

As highlighted in lines 30 and 32 of Algorithm 2, the optimization focuses on refining the logic for how k is adjusted, including parameters such as the adjustment step size. This adjustment process is crucial for adapting the rate limit dynamically in response to network congestion. Drawing inspiration from TCP's AIMD (Additive Increase, Multiplicative Decrease) strategy, which manages congestion by reducing the window size quickly when congestion is detected and gradually increasing it afterward, a similar approach is applied here for controlling k. By mimicking this congestion control mechanism, we aim to create a balance between responsiveness to congestion and recovery to optimal data flow.

In this approach, two distinct strategies for adjusting k were tested:

The first strategy, **Dynamic Adjustment Based on Packet Loss (DYNA)**, modifies k in proportion to the packet loss observed over a specified interval. Higher packet loss triggers a larger decrease in k, while lower packet loss results in a smaller decrease, allowing the algorithm to

Algorithm 2 Userspace Algorithm Main Thread

```
Initialize BPF map value C_byte ← 0, F_drop ← False, R_limit ← Inf
function estimate_max_bandwidth(duration)
    Initialize B_max ← 0
    Monitor bandwidth over duration
    for each interval in duration do
        current bandwidth ← estimate bandwidth in interval
        if current bandwidth > B_max then
            B_max ← current_bandwidth
        end if
    end for
    return B_max
end function
function packet_loss_increasing_count(D_prev)
    Get current dropped packet count D_curr
    if D_curr > D_prev + loss threshold then
        Packet_loss = D_curr − D_prev
        D_prev ← D_curr
        return Packet_loss
    end if
    D_prev ← D_curr
    return 0
end function
Initialize P-controller k ← 1
Read dropped number D_prev from tc statistic
B_max ← estimate_max_bandwidth(duration)
if is_packet_loss_increasing(D_prev) then
    while True do
        packet_loss_increasing = packet_loss_increasing_count(D_prev)
        if packet_loss_increasing > 0) then
            k ← k − 0.01
        else if k < 1.05 then
            k ← k + 0.01
        end if
        Update rate limit R_limit ← k × B_max in BPF map
        Wait update interval
    end while
end if
```

Algorithm 3 Userspace Algorithm Bytecount Reset Thread

```
Create a thread to reset C_byte
Thread Bytecount Reset:
while True do
    Update C_byte ← 0 in BPF map
    Wait Reset Interval
end while
```

respond proportionally to varying levels of congestion. This method seeks to provide finer control over rate adjustments by scaling the decrease in k to the severity of congestion, thereby reducing rate limits precisely when necessary.

The second strategy, **AIMD (Additive Increase, Multiplicative Decrease)**, leverages the traditional AIMD approach from TCP. Here, when congestion is detected, k decreases multiplicatively, providing a rapid response to prevent further congestion. When congestion subsides, k gradually increases additively, allowing a smooth recovery. This AIMD method is particularly effective in environments with fluctuating congestion, as it combines a swift reaction to congestion with a controlled increase, balancing stability and adaptability [52].

To identify the most effective parameters, we tested different variation factors for both strategies, as summarized in Table 3.5. For the dynamic strategy, various proportional factors were applied to the decrease in k based on packet loss. In the AIMD strategy, multiple multiplicative factors were tested to examine how sharply k should reduce during congestion. Each variation offers specific advantages and trade-offs: while a higher reduction factor provides faster relief from congestion, a more gradual decrease can maintain steadier throughput.

This table organizes the variations in k adjustments for both strategies, comparing their respective behaviors during congestion and their potential benefits in achieving balance between responsiveness and data throughput in the network.

In summary, the two strategies for adjusting k, Dynamic Adjustment Based on Packet Loss and AIMD, offer distinct methods for adapting to network conditions. The Dynamic Adjustment approach enables finer control by scaling the reduction in k relative to observed packet loss, making it responsive to varying congestion levels. On the other hand, the AIMD approach emulates TCP's established congestion control model, using a balanced method to handle network fluctuations through multiplicative decreases and additive increases.

These strategies were tested across different variation factors, as shown in Table 3.5, to identify optimal parameters that balance responsiveness and throughput. By evaluating each strategy's effectiveness, we aim to understand their impact on network performance, specifically how each approach influences data flow stability and congestion management.

The following chapter presents a detailed evaluation of these strategies, analyzing their performance under various network conditions to determine the most suitable approach for adaptive rate control.

Table 3.5: Strategies for Adjusting Parameter k in the P-Controller.

Strategy	K increase	K decrease
DYNA1	$k = k + 0.01$	$k = k - 0.001 \times$ *packet loss increasing*
DYNA2	$k = k + 0.01$	$k = k - 0.002 \times$ *packet loss increasing*
DYNA3	$k = k + 0.01$	$k = k - 0.003 \times$ *packet loss increasing*
DYNA4	$k = k + 0.01$	$k = k - 0.004 \times$ *packet loss increasing*
DYNA5	$k = k + 0.01$	$k = k - 0.005 \times$ *packet loss increasing*
AIMD1	$k = k + 0.01$	$k = 0.9 \times k$
AIMD2	$k = k + 0.01$	$k = 0.8 \times k$
AIMD3	$k = k + 0.01$	$k = 0.7 \times k$
AIMD4	$k = k + 0.01$	$k = 0.6 \times k$
AIMD5	$k = k + 0.01$	$k = 0.5 \times k$

4 Evaluation

In this section, we will first evaluate the performance of the data collection testbench, focusing on the latency characteristics of both haptic and video components. Understanding the latency in these elements is essential, as it directly impacts the responsiveness and synchronization of user interactions with the system. Following this, we will assess the effectiveness of our algorithm on the algorithm implementation testbench, concentrating on the latency of each data stream, the algorithm's impact on video quality, and how it influences the Video-Haptic offset, which is key to achieving a cohesive user experience.

4.1 Test bench for data collection

This section focuses on evaluating the performance of the data collection testbench, particularly the screen-to-screen latency in video streams and the latency between the application layers of connected haptic devices. Measuring these latencies provides insight into the system's response time and reveals areas where latency can be minimized to improve synchronization. By analyzing the components that contribute to these latencies, we can better understand how each stage affects overall performance.

For instance, screen-to-screen video latency refers to the delay from when a visual input is displayed on one screen to when it is replicated on another. This metric is crucial in real-time applications, where even slight delays can affect the user's perception of continuity and responsiveness. Similarly, latency between haptic device application layers captures the time taken for a haptic signal to travel from one device's application layer to another. This measurement is essential in scenarios where two users interact through haptic devices or when precise feedback is required, as any delay can disrupt the sensation of real-time feedback.

4.1.1 Latency of Haptic

Measuring the device-to-device delay of haptic feedback is challenging, but it can be addressed by adding timestamps into the code. This allows us to measure the delay from the haptic leader's application layer to the haptic follower's application layer, providing a meaningful metric for assessing the overall system delay, as illustrated in Figure 4.1.

For haptic (touch) device communication, the total delay, often referred to as *End-to-End Latency* or *Perceptual Latency*, represents the time from when a haptic input (such as pressing or sliding) is generated on one device until the remote device receives and processes the signal to provide feedback to the user. This latency is crucial to creating a haptic experience that feels smooth and natural.

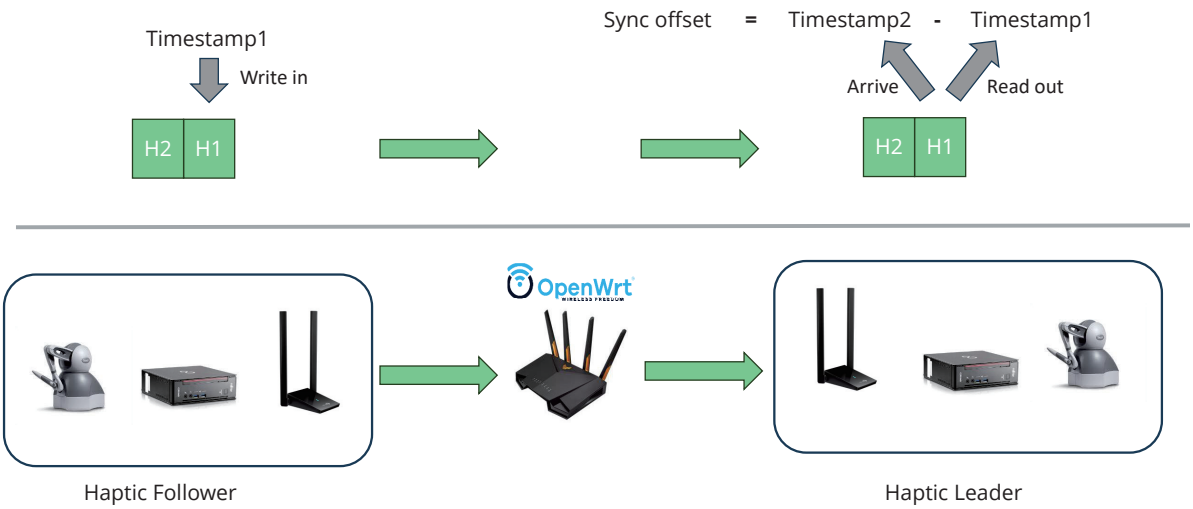


Figure 4.1: Haptic End-to-End Latency Measurement by Adding Timestamps in Haptic Packets.

In haptic communication, the *Perceptual Latency* can be broken down into several stages. First, there is the **input capture delay**, which is the time it takes for the device to detect a user's interaction or an environmental change, such as a collision. Following this, the system experiences a **processing delay**, during which the haptic input is converted into a data packet suitable for transmission. This stage involves encoding the haptic signal, ensuring that the data is ready to be sent over the network.

Next, the data packet encounters the **transmission delay** as it travels from one device to another across the network. This network latency varies based on the quality and speed of the network, influenced by factors such as network congestion and the physical distance between devices. Upon arrival at the receiving device, a **decoding and feedback generation delay** takes place as the haptic signal is decoded and processed, enabling the system to generate the correct haptic feedback for the user. Finally, the system experiences the **haptic rendering delay**, during which the feedback signal is transformed into physical sensations, like vibrations or pressure, that the user can feel.

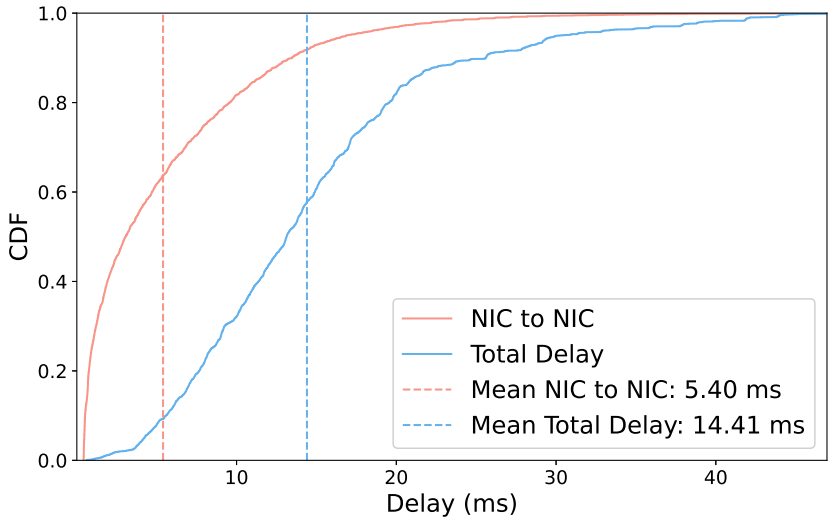


Figure 4.2: Results of Haptic End-to-End Latency Measurement.

To accurately measure these delays, a timestamp is added to each haptic data packet on

the haptic follower's side before it is sent. Once the packet reaches the haptic leader's side, a second timestamp is recorded. This difference between the timestamps captures the delay between the application layers of the two devices. Notably, this measured delay reflects more than just the network transmission time; it also includes the processing delay involved in preparing the data packet and the decoding delay on the receiving end. Therefore, this measurement provides a broader understanding of the system's latency, encompassing multiple stages that affect the responsiveness of the haptic feedback.

As shown in Figure 4.2, the distribution of delay components reveals that the average total delay between the two application layers is 14.41 ms, while the average network transmission delay is 5.40 ms. This difference indicates that a substantial portion of the delay arises from processing and decoding rather than pure transmission time. Given the communication frequency between devices and PCs, which is set at 1 kHz, the ideal delay between each PC and its connected device should theoretically remain below 1 ms. However, real-world measurements capture additional delays due to system overhead and processing requirements, providing a more accurate reflection of the user's experience.

This approach provides a comprehensive view of the delay components in the system, identifying application layer delay as a significant factor influencing overall performance. By understanding these distinct contributions, we can more effectively target latency-reduction strategies to enhance system responsiveness, ultimately improving the user's perception of smooth and real-time interaction.

4.1.2 Latency of Video

The end-to-end delay in video transmission consists of several distinct stages, beginning with the **capture delay**, which is the time required for the camera to capture a video frame. This initial step involves converting visual information into a digital signal for further processing and is influenced by the camera's hardware capabilities and frame rate. Following capture, the system encounters an **encoding delay** as the video data is compressed to reduce its size for efficient transmission. This encoding process varies in duration based on the complexity of the algorithm, compression settings, and the speed of the hardware or software performing the encoding.

After encoding, the video data undergoes **network transmission delay** as it traverses the network infrastructure, passing through routers and switches. This delay is impacted by network congestion, available bandwidth, and the physical distance between the source and destination devices. Once the data arrives at the receiving end, a **decoding delay** occurs as the encoded video is converted back into a displayable format. This delay depends on the codec used and the processing capabilities of the receiving hardware.

Following decoding, the system faces a **rendering delay**, during which the video frame is prepared for display. Rendering may involve adjustments to fit the display requirements, such as scaling or applying visual effects, and can contribute further delay, especially if complex rendering operations are needed. Finally, there is a **display delay** as the frame is shown on the screen. This last step can introduce a minor latency, influenced by the refresh rate of the monitor or screen in use.

Together, these components make up the overall delay experienced from the moment a video frame is captured until it appears on the receiver's screen. By understanding each component in detail, we can pinpoint specific sources of latency and identify opportunities to optimize system performance, ultimately enhancing the smoothness and responsiveness of the user experience.

A common method to measure end-to-end delay is to transmit a timestamp and compare the timestamps on both sides by capturing them in a single photo, thereby providing a direct measurement of the total delay. This approach offers a practical way to evaluate the cumulative

effect of all individual delays, allowing us to quantify the overall latency of the system in real-world conditions.

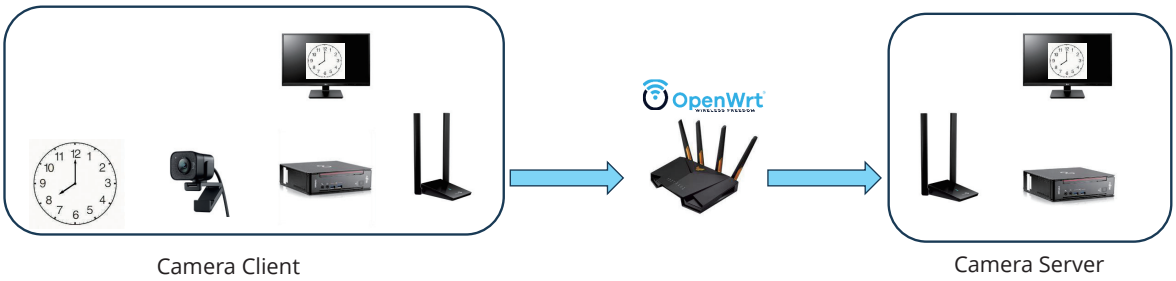


Figure 4.3: Video Screen-to-Screen Latency Measurement via Timestamp Photography.

As shown in Figure 4.3, the camera captures the current time from a millisecond-precision clock, displaying it on the client-side screen. This time is then transmitted through the system and displayed on the server's screen. By taking a single photo that captures both displays at the same moment, we can compare the timestamps on each screen. The difference in times shown represents the end-to-end video delay, providing a straightforward visual indication of total latency and making it easier to quantify the video transmission system's performance.

Each component of this delay measurement provides valuable insights into the system's sources of latency, offering a detailed view of how various stages contribute to the total delay. For instance, the initial capture and encoding delays are influenced by hardware capabilities, such as the camera's speed and resolution and the processing power available for encoding. These factors determine how quickly visual data can be converted into a digital format suitable for transmission. Network transmission delay, on the other hand, is affected by external factors like available bandwidth, network congestion, and the physical distance between devices, all of which play significant roles in determining how fast data travels from the client to the server. Once the data reaches the receiver, decoding and rendering delays are closely tied to the processing power of the receiving device, which impacts how quickly frames can be prepared for display. Lastly, display delay is influenced by the refresh rate of the display hardware, which can introduce minor but noticeable latency, affecting the final output timing.

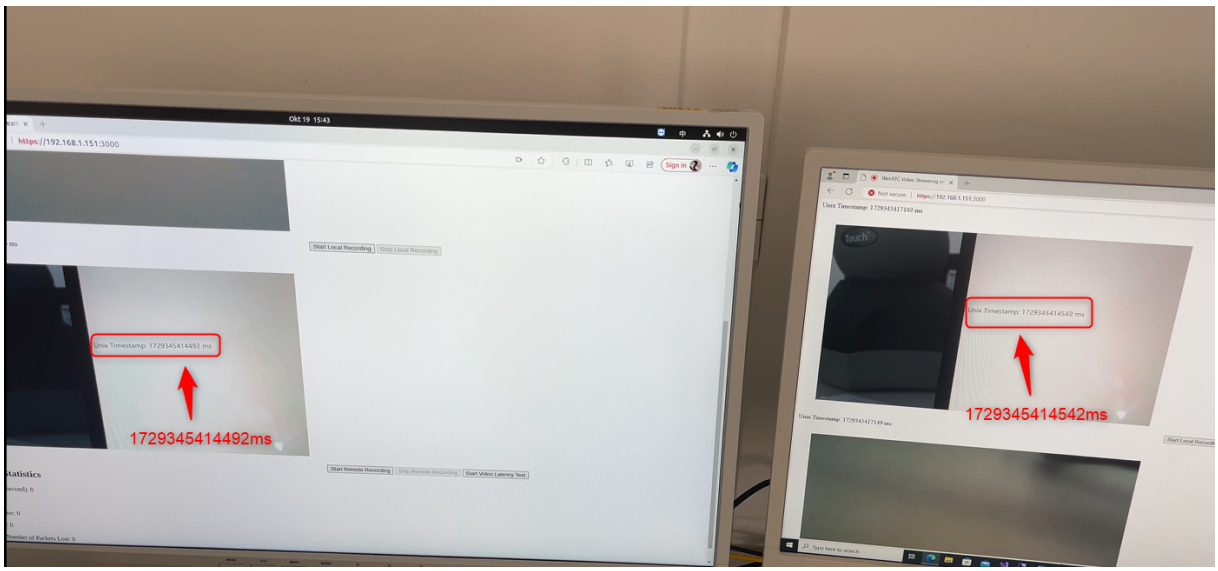


Figure 4.4: Timestamps Displayed on Both Camera Client Screen and Camera Server Screen.

As demonstrated in Figure 4.4, we measure this end-to-end delay by using a camera to

capture a timestamp with millisecond precision, then extracting a specific frame for analysis. For example, at a particular moment, the timestamp displayed on the camera client reads 1729345414542 ms, while the time on the camera server reads 1729345414492 ms. This difference of 50 ms represents the end-to-end delay, where the time on the camera client is expectedly later than the server due to transmission delay. This straightforward timestamp comparison provides an effective visual representation of the delay between client and server.

To build a comprehensive analysis, we sampled a 4-minute video at 60 frames per second (fps), extracting 15 frames per second, yielding a total of 4950 images, similar to the example shown in Figure 4.4. Using EasyOCR [53], we recognized the timestamps in each image and filtered the data with regular expression matching to ensure accuracy. This process successfully yielded 312 clear and valid timestamp pairs, which were then used to conduct a statistical analysis of the end-to-end delay.

The resulting CDF (Cumulative Distribution Function) distribution, illustrated in Figure 4.5, shows that the average end-to-end delay is around 72.23 milliseconds. Outliers were first filtered using the boxplot rule, and then a 95% confidence interval was applied to further refine the data, resulting in a standard deviation of 25.32 milliseconds. The relatively small standard deviation suggests that delays were fairly consistent across the samples, reflecting a stable transmission performance over time. This consistency in delay measurements provides confidence in the reliability of the video transmission system. Identifying these delay values, alongside the standard deviation, allows us to pinpoint potential optimization areas to further reduce latency and improve user experience.

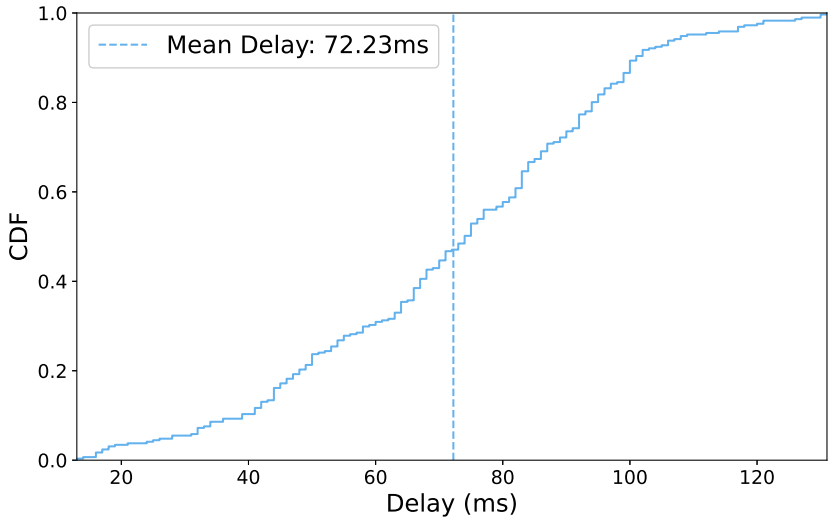


Figure 4.5: CDF plot of Video Screen-to-Screen Latency.

Some components of the delay can be measured with high precision using dump data from the WebRTC API, which provides detailed statistics such as encoding time and processing time. Transmission delay, in turn, can be directly measured by capturing and analyzing packets at both the sender's and receiver's Network Interface Cards (NICs). The statistical results of this analysis, as shown in Figure 4.6, reveal that the average network delay is 5.63 ms, the average encoding delay is 10.14 ms, and the average processing delay is 41.79 ms. Here, processing delay represents the time taken for an audio sample or video frame to be fully decoded and ready for playback after the first RTP packet is received. This includes the time required to receive, decode, and prepare the frame for rendering on the monitor.

Additionally, the frame rates of both client and server monitors, as well as the camera and recorded video, are set to 60 Hz. Operating at this frame rate can introduce additional latency

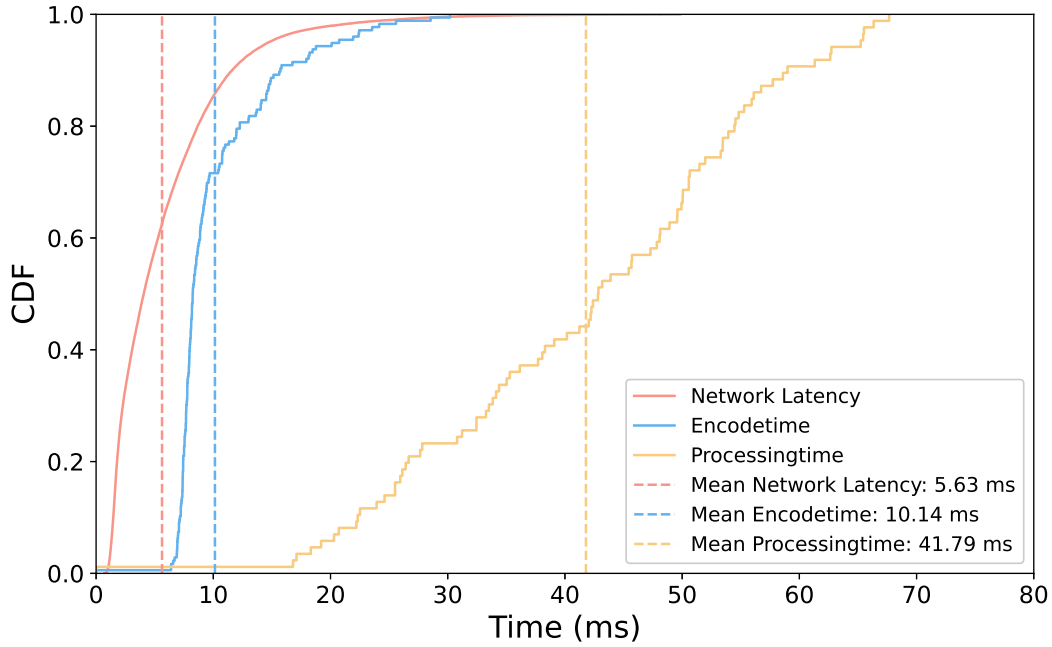


Figure 4.6: CDF Plot of Measurable Components of Video Latency.

due to synchronization requirements with the frame cycle, as each component may add a slight delay depending on its alignment with the frame cycle. Although these frame rate-induced delays are not directly cumulative, they do add a small amount of latency to the system that is spread over the frame cycle.

The precisely measured delays (5.63 ms for network transmission, 10.14 ms for encoding, and 41.79 ms for processing) account for most of the observed end-to-end delay of 71.55 ms, leaving a remaining delay component that cannot be directly measured but is reasonable in magnitude. This unknown component likely consists of small cumulative delays from frame rate synchronization and other minor system overheads, which together align closely with the expected total end-to-end latency.

This layered approach to analyzing delay not only validates the measured end-to-end delay but also highlights the contributions of each major component, offering a clearer understanding of where further improvements might reduce latency and enhance system performance.

4.2 Test bench for algorithm Implementation

In this section, we compare the performance of CoDel, CAKE, and PFIFO, both with and without the addition of the XDP algorithm. Our evaluation uses five key metrics to assess the effectiveness of the XDP algorithm in improving data transmission quality and user experience across different queuing mechanisms.

4.2.1 Metrics and Parameters

For the data transmitted by the testbench, the effectiveness of the XDP algorithm is evaluated using the following five metrics:

1. **Packet Loss Rate and Frame Loss Rate**: Packet loss rate represents the percentage of packets lost during transmission, directly impacting data integrity and the quality of

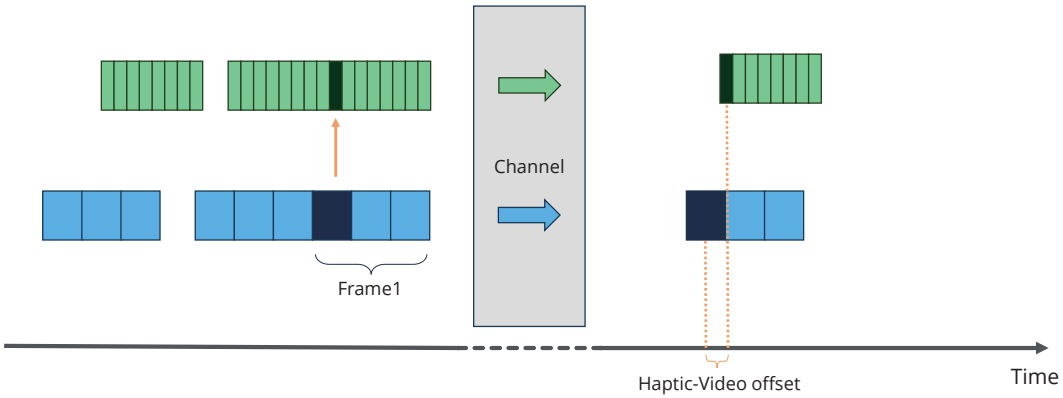


Figure 4.7: Measurement Principle of Video-Haptic Synchronization Offset.

service. Frame loss rate, in contrast, measures the percentage of video frames that fail to reach the receiver, which can significantly affect user experience, especially in real-time applications.

In our setup, the haptic data stream is given high priority and adequate bandwidth, resulting in no packet loss for haptic transmissions. However, the video stream experiences discrepancies between packet loss and frame loss due to the VP8 encoding method; any packet loss within a single frame renders the entire frame undecodable. Ideally, the packet loss rate and frame loss rate should be closely aligned, as this would indicate minimal impact on video quality from packet loss.

2. **Frames Per Second (FPS)**: Frames per second (FPS) indicates the number of frames displayed or processed each second. Higher FPS contributes to smoother visual and sensory experiences, while lower FPS can lead to lagging or choppy visuals that detract from user experience. By analyzing the number of successfully transmitted frames per second across different bandwidth conditions, we can assess how varying bandwidths impact video quality and overall user experience.

3. **Latency of Haptic Stream**: The haptic stream requires extremely low latency, as any delay can disrupt the user's sense of real-time feedback. To ensure this, the haptic stream is assigned both high priority and sufficient bandwidth, keeping its latency consistently below 5 ms. This metric highlights the effectiveness of the prioritization strategy and ensures that haptic feedback remains responsive under varying network conditions.

4. **Latency of Video Stream**: To evaluate the transmission delay of the video stream, we measured the latency of all successfully transmitted video packets. This metric provides insights into how network congestion affects the lower-priority video stream, especially when the high-priority haptic stream is occupying substantial bandwidth. By understanding video latency under congestion, we can assess the system's robustness in maintaining acceptable delay levels for video traffic despite potential bottlenecks.

5. **Haptic-Video Offset**: The haptic-video offset measures synchronization performance between the haptic and video streams. As shown in Figure 4.7, for each video frame at the sender side, we identify the nearest haptic packet within a specified threshold range, then track these paired packets. Upon reaching the receiver, the difference in arrival times is measured to determine the haptic-video offset. This metric serves as an indicator of how well the system synchronizes haptic and video data streams, which is crucial for creating a cohesive user experience.

These five metrics collectively provide a comprehensive view of the system's performance by capturing both the quality and responsiveness of data transmission. By analyzing packet and frame loss rates, we can understand how well data integrity and service quality are maintained, which is crucial for stable transmission. Similarly, FPS measurements allow us to gauge the smoothness of visual and sensory experiences, highlighting the system's ability to handle different bandwidth conditions without compromising user experience.

Latency measurements for both the haptic and video streams provide further insight into the system's responsiveness under network congestion, particularly in prioritizing low-latency data streams like haptic feedback while managing video traffic effectively. Finally, the haptic-video offset serves as a key indicator of synchronization performance, ensuring that haptic and visual information are well-aligned to provide a cohesive, real-time experience.

Together, these metrics allow us to identify specific areas where transmission quality can be optimized and to assess the overall impact of the XDP algorithm in maintaining high-quality data transmission across varying network conditions. This analysis highlights the strengths and limitations of each queuing mechanism and the effectiveness of XDP in achieving reliable, low-latency performance, offering valuable insights for further system improvement.

4.2.2 Effect Without TC Queue Management and Classification

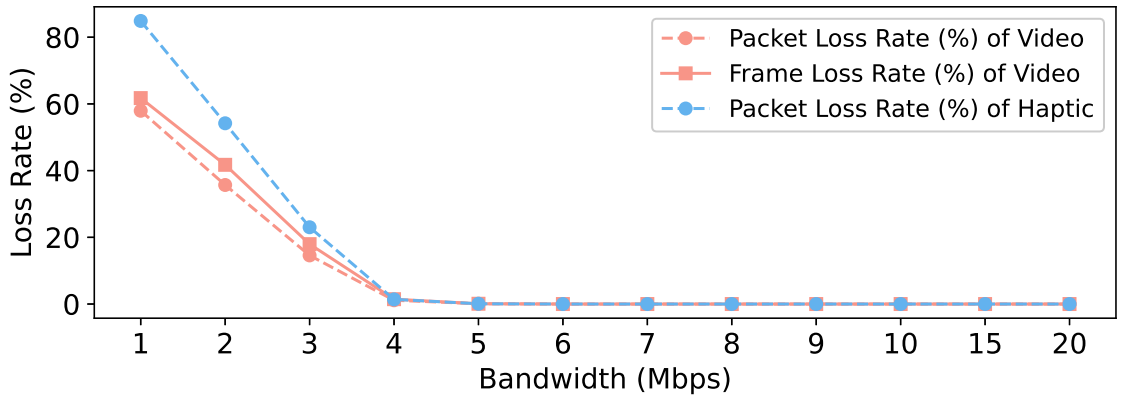
In this section, we assess the system's performance without Traffic Control (TC) queue management and classification. In the absence of TC, packets are processed directly using a First-In, First-Out (FIFO) manager, which can lead to congestion buildup and uncontrolled queuing delays under high network loads. Additionally, without classification, all traffic is treated equally, resulting in no prioritization for latency-sensitive streams, such as video or haptic data. This baseline measurement allows us to observe the impact of not implementing queue management and classification on overall network performance, latency, and frame stability, providing a point of comparison for evaluating the benefits of TC.

Figure 4.8 illustrates the behavior of haptic and video streams in three subfigures under conditions without Traffic Control (TC). Each subfigure represents a distinct aspect of network performance, highlighting the limitations of a system without traffic management in terms of packet loss, latency, and frame stability.

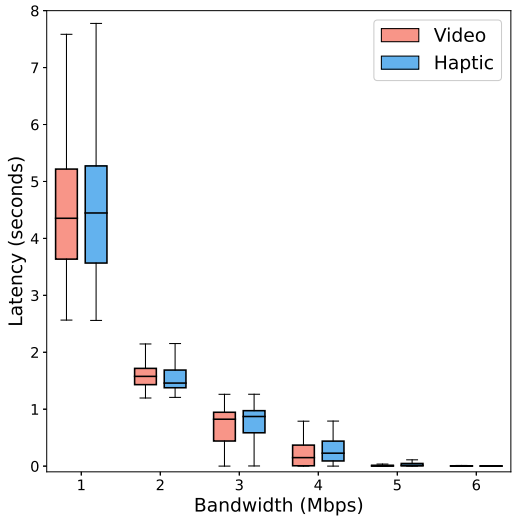
Subfigure 4.8a illustrates the relationship between bandwidth and the loss rate of haptic and video streams. As bandwidth decreases below 4 Mbps, both packet and frame loss rates rise significantly due to congestion, impacting overall data integrity and quality. Interestingly, the haptic stream exhibits a higher packet loss rate than the video stream, likely due to FIFO's tail-drop mechanism, where packets at the end of the queue are discarded when the queue becomes full. Consequently, video frame loss rates closely match packet loss rates, indicating that entire frames are lost when any packet within the frame is dropped. This effect is particularly pronounced with VP8 encoding, where losing a single packet within a frame prevents the entire frame from being decoded, emphasizing the importance of packet retention for video quality.

Subfigure 4.8b presents the latency of video and haptic streams across various bandwidth levels. As bandwidth drops below 5 Mbps, both streams experience a noticeable increase in latency, exceeding acceptable Quality of Service (QoS) thresholds. The haptic stream, which is more sensitive to latency, suffers from increased delays as it competes equally with video traffic in the FIFO system. This scenario highlights how, without prioritization and queuing mechanisms, latency-sensitive data like haptic feedback can experience delays similar to video traffic, potentially reducing the responsiveness of real-time applications.

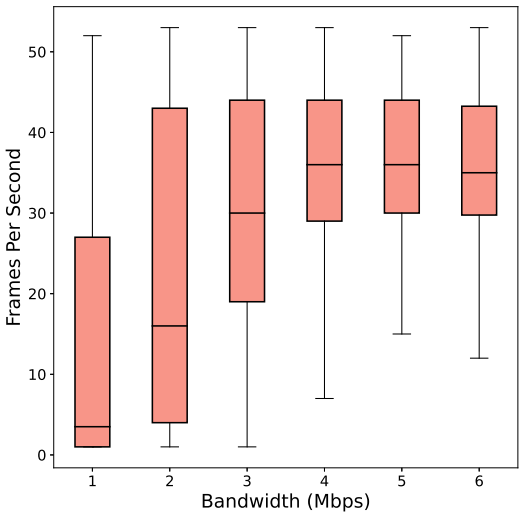
Subfigure 4.8c illustrates the frames per second (FPS) of the video stream as a function of available bandwidth. The box plots reveal that FPS becomes highly variable and begins to degrade as bandwidth drops below 4 Mbps. This reduction in FPS stems from the higher



(a) Packet loss rate and frame loss rate without TC.



(b) Haptic and Video Latency without TC.



(c) Video FPS without TC.

Figure 4.8: Effect Without TC Queue Management and Classification.

packet loss rates and increased latency at lower bandwidths, which interrupt smooth video playback. Without TC mechanisms, the system lacks the ability to maintain a stable FPS under constrained bandwidth, resulting in an inconsistent and degraded viewing experience. This subfigure underscores the limitations of the FIFO queue under heavy load, where the lack of bandwidth management directly impacts frame rate stability.

In summary, Figure 4.8 provides a comprehensive overview of network behavior in the absence of traffic control. As shown, the FIFO-based system faces substantial limitations under reduced bandwidth, leading to increased packet loss, higher latency, and decreased FPS stability. Without Traffic Control and classification, all packets are treated uniformly, resulting in significant performance degradation for latency-sensitive applications. This baseline analysis underscores the importance of effective queue management and classification in achieving acceptable network performance, particularly for applications that rely on consistent low latency and high reliability.

4.2.3 Strategy of the Proportional Controller

As described previously in Table 3.5, there are different strategies for adjusting the parameters of the P controller. Here, we compare these strategies in terms of their advantages and disadvantages and select the best approach for further analysis.

In this section, we will first compare the advantages and disadvantages of different AIMD and DYNA methods and then select the best one from each (AIMD and DYNA) for a final comparison.

AIMD

As shown in Table 3.5, five different AIMD strategies are compared. Figure 4.9 illustrates how various performance metrics respond under each AIMD strategy when CoDel is used for queue management, providing insight into the trade-offs between packet loss, latency, framerate, and synchronization.

Subfigure 4.9a illustrates the packet loss rate for each AIMD strategy. Packet loss begins to occur when video bandwidth drops below 2.75 Mbps. The parameter following DYNA represents how XDP handles packet dropping when congestion occurs in the network stack, where a larger parameter leads to stricter packet dropping (i.e., more packets are discarded). Among the AIMD strategies, AIMD1 has the lowest packet loss rate, while AIMD5 has the highest, indicating that different strategies prioritize either packet retention or rapid congestion control.

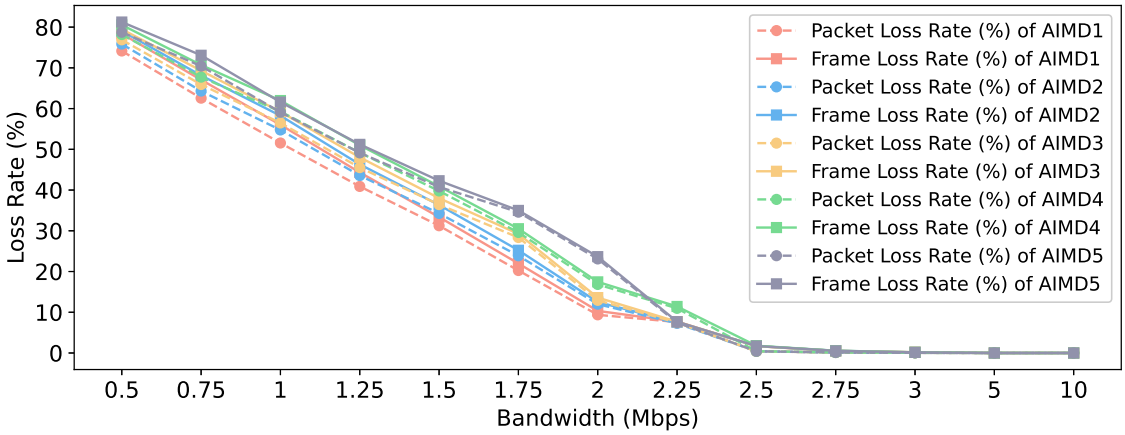
Subfigure 4.9b presents the CDF (cumulative distribution function) of latency for the haptic stream across different strategies. The results show that the haptic stream consistently meets Quality of Service (QoS) requirements with very low latency across all AIMD strategies, indicating that each approach can support real-time haptic feedback.

Subfigure 4.9c compares latency levels for three representative AIMD strategies: AIMD1, AIMD3, and AIMD5. As bandwidth increases, latency decreases across all three strategies. AIMD1 exhibits the highest latency, while AIMD5 achieves the lowest, indicating that stricter AIMD strategies prioritize lower latency by reducing congestion more aggressively.

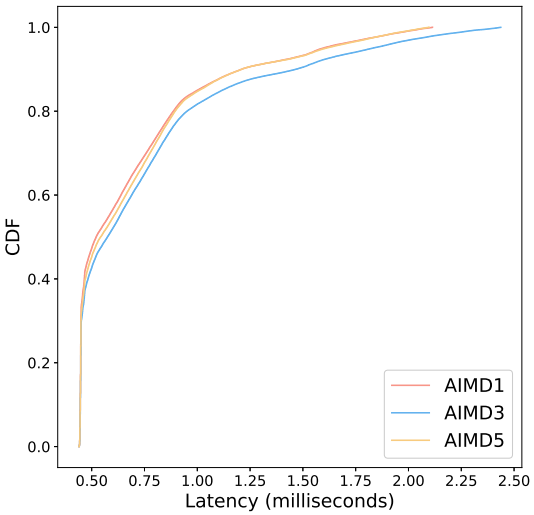
Subfigure 4.9d compares the framerate achieved by these three AIMD strategies. As bandwidth increases, the framerate also rises, with AIMD1 attaining the highest framerate and AIMD5 the lowest. This trend highlights that less aggressive AIMD strategies support higher video quality at the expense of increased latency.

Subfigure 4.9e examines the effect of these strategies on the haptic-video offset. As bandwidth increases, the haptic-video offset decreases noticeably. AIMD1 has the highest offset, followed by AIMD3, while AIMD5 achieves the lowest offset. This finding shows that stricter AIMD strategies not only reduce latency but also improve synchronization between haptic and video streams.

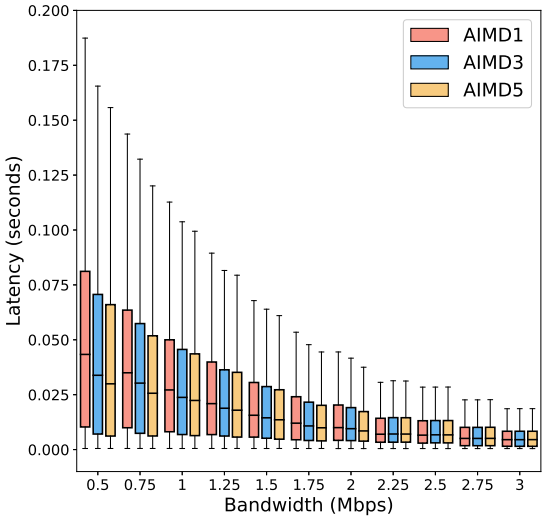
In summary, Figure 4.9 highlights how each AIMD strategy impacts various performance metrics. Stricter AIMD controls, such as AIMD5, lead to higher packet loss rates, which reduce video framerate but also help decrease channel traffic, resulting in lower latency and a smaller haptic-video offset. Therefore, selecting an AIMD strategy requires balancing video framerate with haptic-video offset and latency. Ultimately, AIMD3 was chosen as our preferred strategy, as it offers low latency, a small haptic-video offset, and maintains a satisfactory framerate (only 1-2 frames lower than AIMD1 at low bandwidth levels), making it an effective compromise between video quality and synchronization.



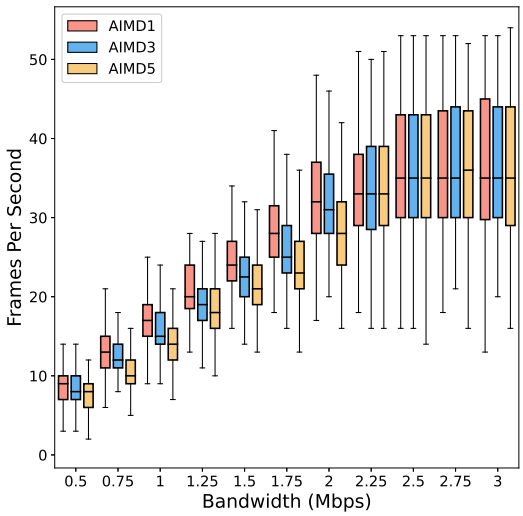
(a) Packet Loss Rate and Frame Loss Rate.



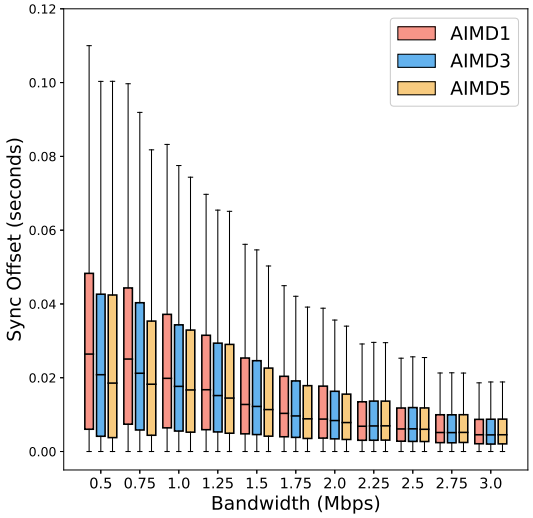
(b) CDF of Haptic Stream Latency.



(c) Video Stream Latency.



(d) FPS of Video.



(e) Video-Haptic Synchronization Offset.

Figure 4.9: Comparison of Different AIMD Strategies Across Various Bandwidths.

DYNA

The DYNA strategy is conceptually similar to AIMD, with adjustments made to the k value of the p-controller. As shown in Table 3.5, five different DYNA strategies are compared, and Figure 4.10 illustrates the impact of these strategies on various performance metrics.

Subfigure 4.10a shows the packet loss rate across different DYNA strategies. Noticeable packet loss begins when bandwidth falls below 2.5 Mbps, and as the DYNA parameter increases, the loss rate rises. In this context, bandwidths above 3 Mbps are generally sufficient to prevent significant loss, so the subsequent subfigures focus on performance below this threshold. The DYNA parameter controls the degree of change in the k value of the p-controller, with larger values resulting in stricter adjustments to bandwidth. Consequently, when packet loss occurs, a larger DYNA parameter reduces k more drastically, leading XDP to impose stricter bandwidth limitations to mitigate congestion.

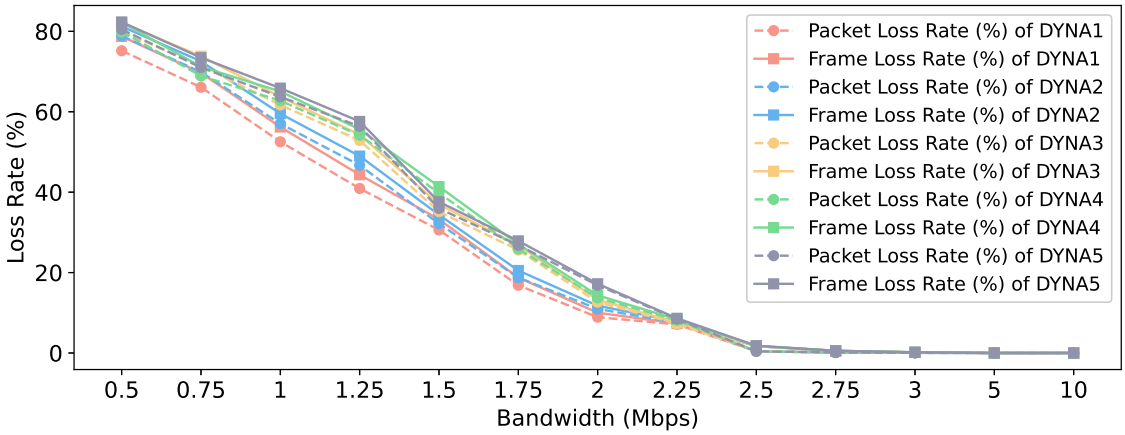
Subfigure 4.10b shows the CDF (cumulative distribution function) distribution of haptic latency for different DYNA strategies. This metric consistently meets Quality of Service (QoS) requirements across all strategies, indicating that the DYNA parameter variations do not adversely impact haptic stream latency.

Subfigure 4.10c illustrates video packet latency under different DYNA strategies. As bandwidth increases, latency visibly decreases across all strategies, with larger DYNA parameters yielding more significant latency reductions. Notably, at low bandwidths, DYNA3 demonstrates a marked improvement in latency over DYNA1, reducing latency by approximately 10 ms. This finding suggests that larger DYNA parameters can effectively improve latency performance under constrained bandwidth conditions.

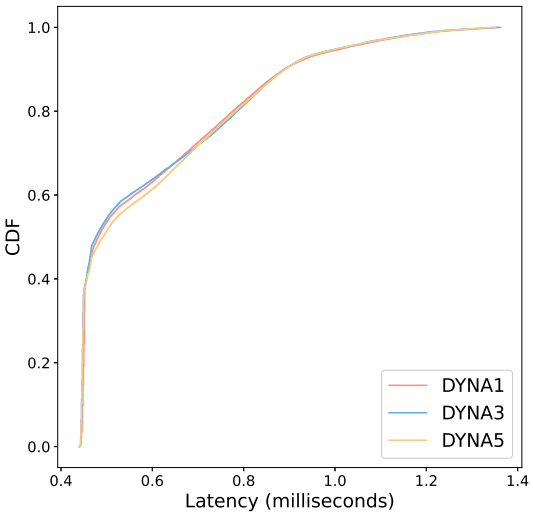
Subfigure 4.10d compares video framerate across different DYNA strategies. As bandwidth increases, framerate also rises significantly for each strategy. However, with larger DYNA parameters, framerate experiences a slight decrease of about 1-2 frames, reflecting a trade-off between framerate and latency reduction.

Subfigure 4.10e illustrates the impact of different DYNA parameters on the haptic-video offset. As bandwidth increases, the haptic-video offset decreases across all strategies. Higher DYNA parameters result in a more substantial offset reduction, indicating improved synchronization between haptic and video streams.

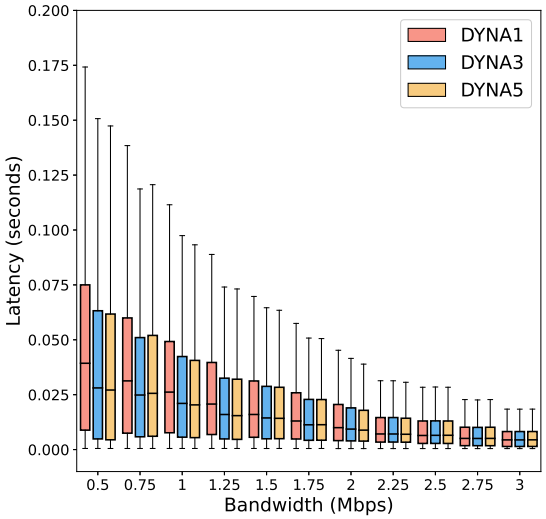
In summary, Figure 4.10 highlights how varying DYNA parameters affect key performance metrics. Similar to the AIMD strategy, choosing an optimal DYNA strategy requires balancing video framerate and haptic-video offset, as well as managing packet loss and latency. While higher packet loss typically reduces video quality, it also leads to lower latency and improved synchronization. Ultimately, DYNA3 was selected as the preferred strategy, as it achieves low latency, maintains an adequate framerate, and effectively balances performance for both video and haptic data.



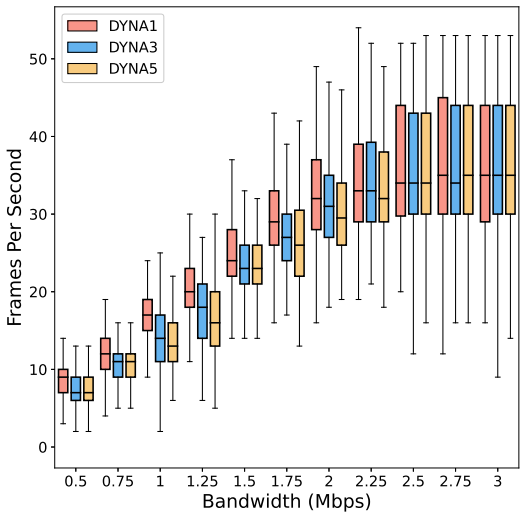
(a) Packet Loss Rate and Frame Loss Rate.



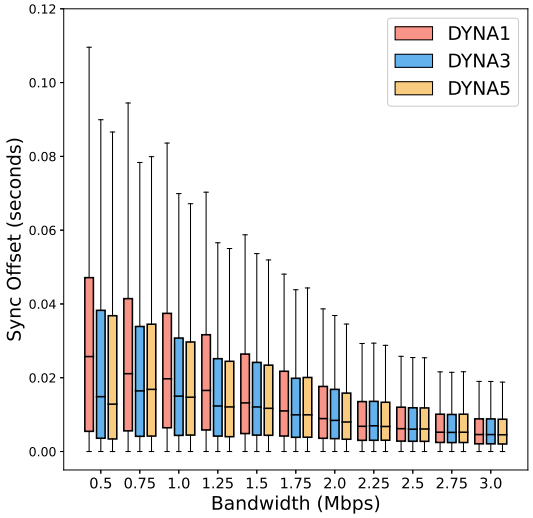
(b) CDF of Haptic Stream Latency.



(c) Video Stream Latency.



(d) FPS of Video.

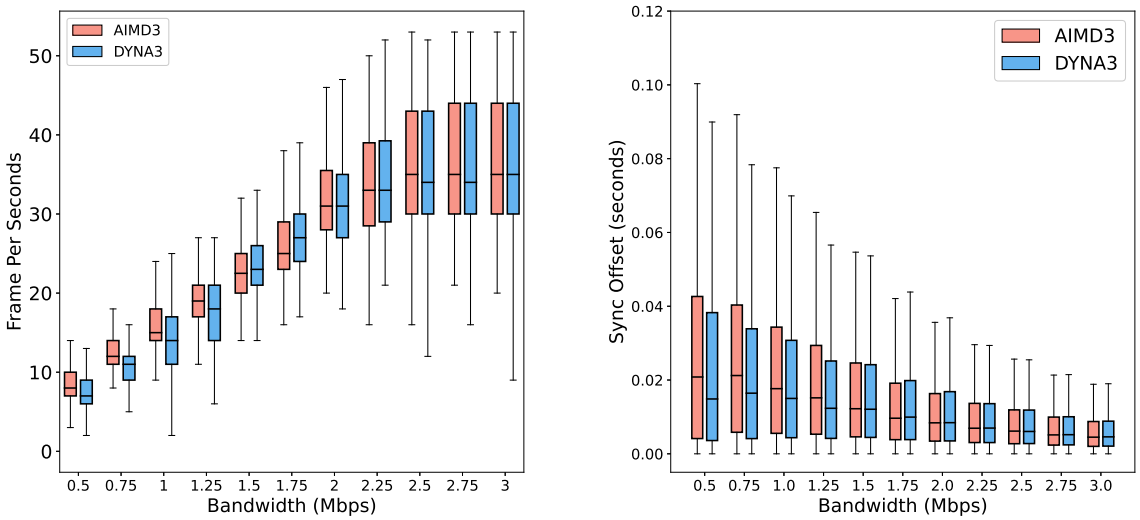


(e) Video-Haptic Synchronization Offset.

Figure 4.10: Comparison of Different DYNA Strategies Across Various Bandwidths.

AIMD VS DYNA

Figure 4.11 compares two of the better-performing DYNA and AIMD strategies, focusing primarily on video FPS and the video-haptic offset. As shown, when bandwidth is the same, AIMD3 consistently achieves a higher FPS, with 1-2 frames more than DYNA3. Meanwhile, DYNA3 performs better in terms of reducing the haptic-video offset, suggesting improved synchronization. Ultimately, AIMD3 was selected as the final strategy, mainly for the following reasons:



(a) FPS of Video.

(b) Video-haptic Synchronization Offset.

Figure 4.11: Comparison of AIMD3 and DYNA3 Strategies Across Various Bandwidths.

First, AIMD3 provides a more stable framerate under varying bandwidth conditions. As shown in Subfigure 4.11a, at low bandwidth levels, the video framerate provided by AIMD3 remains more stable, with a smaller range between maximum and minimum values. In contrast, DYNA3's FPS drops below 5 FPS at a bandwidth of 1 Mbps, which could negatively impact the user experience by creating choppy or delayed visuals. This stability in framerate under AIMD3 enhances the overall viewing experience, especially in scenarios with constrained bandwidth.

Second, both strategies achieve a median latency of around 20 ms at low bandwidths, as shown in Subfigure 4.11b, meeting the Quality of Service (QoS) requirements. At this level of latency, the higher FPS provided by AIMD3 offers a greater marginal benefit (as previously shown in Figure 2.1), as it translates to smoother video playback without compromising latency performance. This combination of stable framerate and QoS-compliant latency makes AIMD3 an advantageous choice.

In summary, while DYNA3 offers improved haptic-video synchronization, AIMD3's superior framerate stability and QoS-aligned latency make it the preferred strategy. This choice balances both video quality and responsiveness, providing an optimized experience across a range of bandwidth conditions.

4.2.4 Effect of XDP on PFIFO

In this section, we explore the benefits of our XDP algorithm on a PFIFO-managed queue, evaluating performance using the metrics described above.

Figure 4.12 illustrates the effect of implementing the XDP algorithm on the PFIFO (Packet First-In-First-Out) queue system. Each subfigure presents a metric used to assess network

performance under various bandwidth constraints, allowing for a comparison of configurations with and without the XDP algorithm. The five subfigures evaluate packet and frame loss rate, haptic stream latency, video stream latency, frames per second (FPS), and haptic-video offset, respectively, to provide a comprehensive view of performance impacts.

Subfigure 4.12a shows the relationship between bandwidth and both the packet loss rate and frame loss rate for video traffic under PFIFO queues with and without the XDP algorithm. As bandwidth increases, both packet and frame loss rates decrease significantly. The PFIFO configuration alone exhibits a notably higher packet loss rate, especially when bandwidth is below 2 Mbps. Therefore, in the following four subfigures, we focus on cases where bandwidth is less than or equal to 3 Mbps, allowing a more focused analysis of low-bandwidth performance. In contrast, the XDP & PFIFO configuration effectively reduces frame loss, demonstrating improved performance in low-bandwidth scenarios. The close alignment between frame loss and packet loss rates in the PFIFO configuration highlights the tail-drop behavior of PFIFO queues, where packet drops directly impact frame completeness in VP8-encoded video streams.

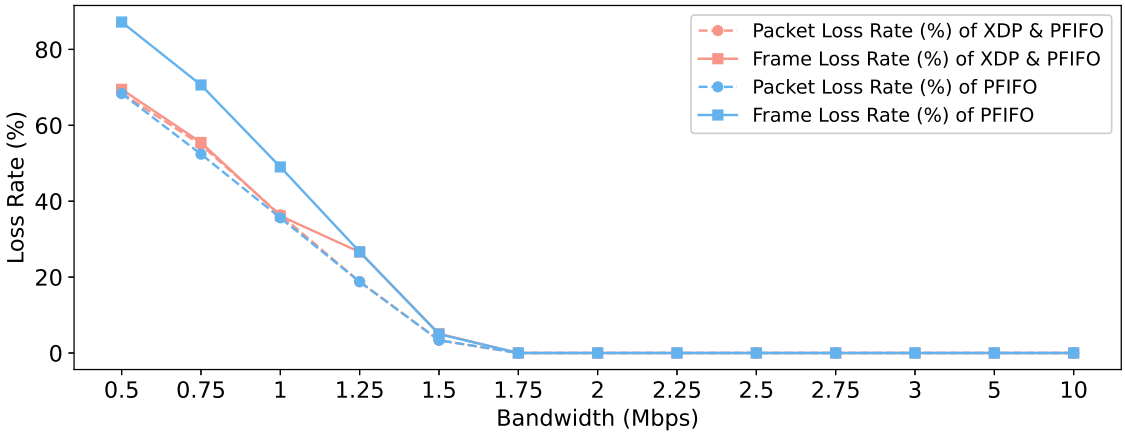
Subfigure 4.12b provides a CDF plot of the haptic stream latency distribution for both configurations. The latency CDF indicates that the XDP & PFIFO configuration does not significantly increase the transmission latency of the haptic stream, maintaining latency levels that meet the Quality of Service (QoS) requirements for haptic feedback.

Subfigure 4.12c examines video stream latency across different bandwidths. As bandwidth increases, latency noticeably decreases for both configurations, but the XDP & PFIFO setup achieves a greater latency reduction than PFIFO alone. However, even with the XDP enhancement, the latency remains above the threshold required by QoS for video. This is primarily because PFIFO does not actively manage packet drops, while the XDP algorithm helps to mitigate congestion by intelligently handling packet drops at the XDP layer. Consequently, although PFIFO with XDP shows some latency reduction, this improvement alone is insufficient to meet strict QoS requirements.

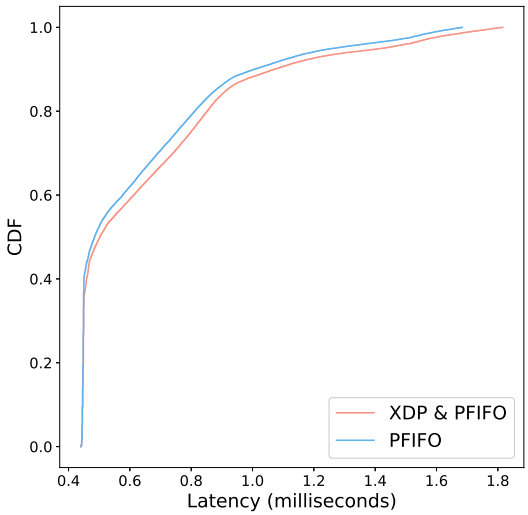
Subfigure 4.12d displays the frames per second (FPS) of the video stream as a function of bandwidth. The box plots reveal that FPS stability improves as bandwidth increases, with the XDP & PFIFO configuration achieving a higher median FPS and reduced variability at lower bandwidths compared to the PFIFO setup alone. This enhanced FPS stability is crucial for video applications, as it contributes to smoother playback and a more consistent user experience, particularly in bandwidth-constrained environments.

Subfigure 4.12e presents the offset between video and haptic streams, a metric of synchronization quality. As bandwidth increases, the haptic-video offset decreases, reflecting improved synchronization. Similar to the latency situation, XDP & PFIFO improves the haptic-video offset but still does not meet the required QoS. Consequently, while this improvement indicates that the XDP algorithm can enhance synchronization, it remains insufficient for applications with stringent QoS demands.

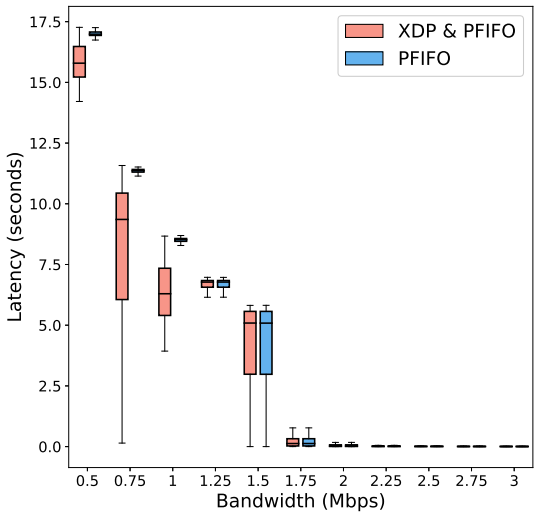
In summary, Figure 4.12 provides a comprehensive analysis of the performance improvements achieved by integrating the XDP algorithm with PFIFO queue management. The XDP-enhanced configuration effectively reduces frame loss rates, lowers latency for both haptic and video streams, stabilizes video FPS, and improves haptic-video synchronization. Although XDP & PFIFO cannot meet all QoS requirements, it demonstrates that our algorithm provides measurable improvements in video FPS and haptic-video offset, serving as a reference for the subsequent comparison with CoDel and CAKE algorithms.



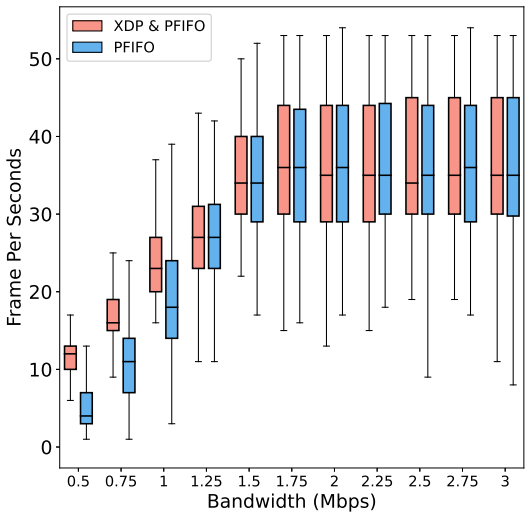
(a) Packet Loss Rate and Frame Loss Rate.



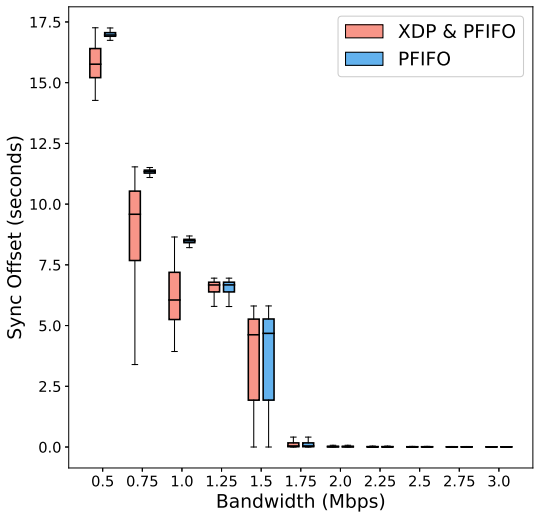
(b) CDF of Haptic Stream Latency.



(c) Video Stream Latency.



(d) FPS of Video.



(e) Video-Haptic Synchronization Offset.

Figure 4.12: Impact of XDP Algorithm on PFIFO-Managed Queues Across Different Bandwidths.

4.2.5 Effect of XDP on CoDel

In this section, we explore the benefits of our XDP algorithm on a CoDel-managed queue, evaluating its performance using the defined metrics.

Figure 4.13 illustrates the effect of implementing the XDP algorithm on the CoDel queue management system. CoDel is known for its ability to control queue delay by dynamically dropping packets based on delay, rather than solely relying on queue length. This figure uses five subplots to evaluate the impact of XDP on CoDel under varying bandwidth conditions. The metrics analyzed include packet loss rate, frame loss rate, latency, frames per second (FPS), and haptic-video synchronization offset, providing a comprehensive assessment of XDP's effect on CoDel's performance.

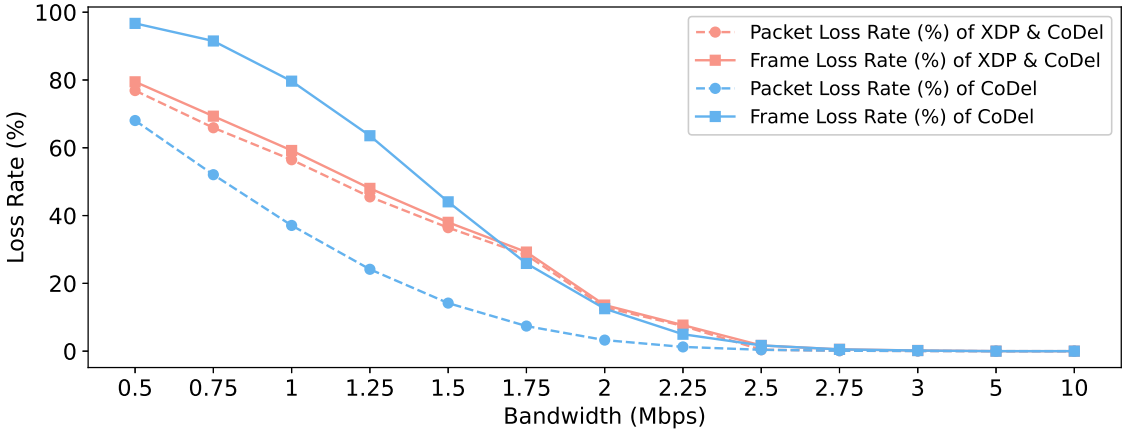
Subfigure 4.13a describes the relationship between bandwidth and both the packet loss rate and frame loss rate for video traffic, comparing the performance of CoDel alone with that of XDP & CoDel. As bandwidth increases, both packet and frame loss rates decrease significantly, indicating improved transmission stability. When bandwidth falls below 2.75 Mbps, packet loss begins to occur, suggesting that bandwidths of 3 Mbps and above can be considered sufficient for smooth operation. Therefore, the following four subfigures focus on data below this threshold to highlight performance under constrained conditions. As shown, in XDP & CoDel, the packet loss rate increases while the frame loss rate decreases, indicating that packet loss primarily occurs at the XDP layer. This behavior suggests that XDP efficiently drops packets before they reach the CoDel-managed queue, preserving frame integrity by dropping unnecessary packets at an earlier stage.

Subfigure 4.13b provides a CDF plot of the haptic stream latency under both configurations. The CDF shows that both CoDel and XDP & CoDel meet the Quality of Service (QoS) requirements for the haptic stream, with latency remaining within acceptable levels for real-time applications.

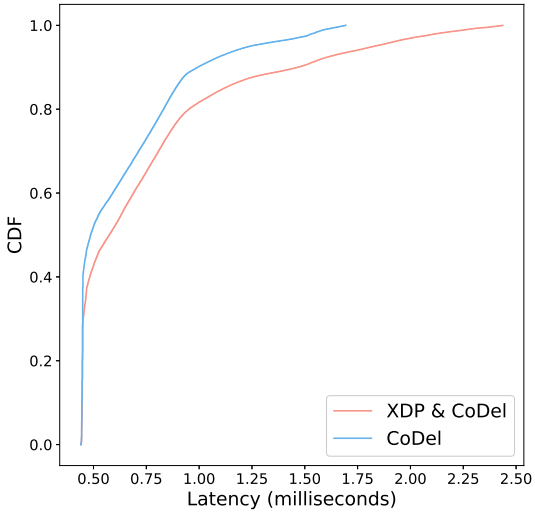
Subfigure 4.13c illustrates video stream latency across different bandwidths. Both the CoDel and XDP & CoDel configurations show a decrease in latency as bandwidth increases. However, XDP & CoDel achieves consistently lower video latency under limited bandwidth, especially at lower levels. This latency reduction is particularly significant: without XDP, the median latency of the video stream under CoDel control is around 200 ms, whereas it decreases to approximately 50 ms with the XDP algorithm implemented. Additionally, when bandwidth is sufficient, XDP does not negatively impact video stream latency, demonstrating that the XDP-enhanced setup provides latency benefits in congested conditions without compromising performance at higher bandwidths.

Subfigure 4.13d presents the frames per second (FPS) of the video stream across different bandwidths, with FPS measured as an indicator of video quality. The XDP & CoDel configuration shows higher median FPS values, especially at low bandwidths, and demonstrates greater stability with a smaller range between maximum and minimum FPS. This stability and FPS improvement contribute to a smoother video playback experience. When bandwidth is as low as 0.5 Mbps, the median FPS of XDP & CoDel still reaches 8 FPS, while CoDel alone achieves around 2 FPS. This improvement helps maintain acceptable video quality in poor network conditions, supporting task performance even under extreme bandwidth limitations.

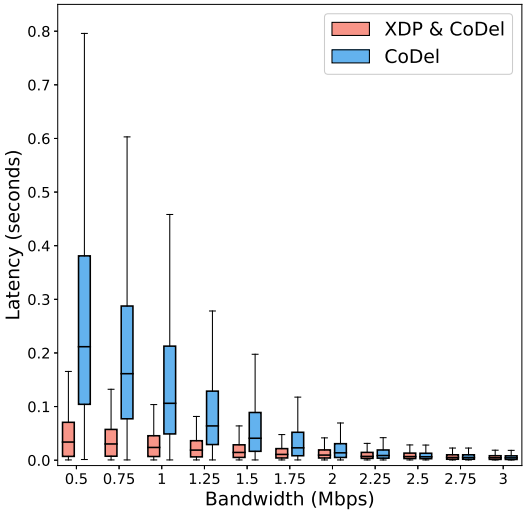
Subfigure 4.13e shows the synchronization offset between the video and haptic streams, which is a crucial metric for real-time applications requiring tight synchronization. The XDP & CoDel configuration demonstrates a lower median offset at most bandwidth levels compared to CoDel alone. As bandwidth increases, the offset decreases for both setups; however, XDP & CoDel maintains better synchronization overall. For example, at 0.5 Mbps, the median synchronization offset for XDP & CoDel is around 20 ms, while for CoDel alone it is approximately 50 ms, effectively reducing the synchronization offset by half. This improvement suggests that adding XDP helps lower the time synchronization offset between video and haptic streams.



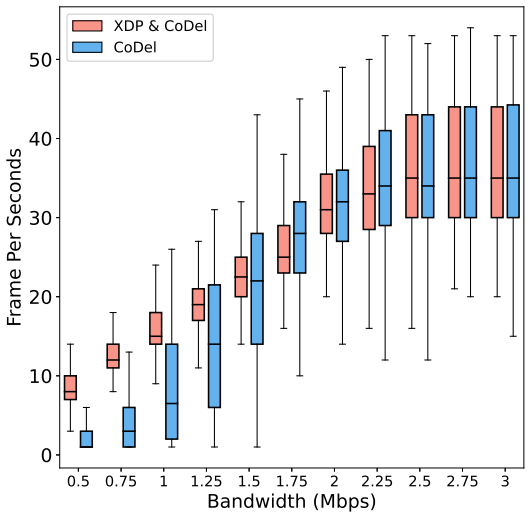
(a) Packet Loss Rate and Frame Loss Rate.



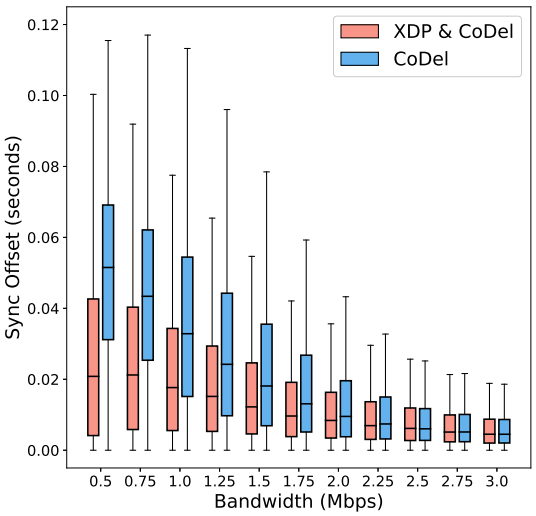
(b) CDF of Haptic Stream Latency.



(c) Video Stream Latency.



(d) FPS of Video.



(e) Video-Haptic Synchronization Offset.

Figure 4.13: Impact of XDP Algorithm on CoDel-Managed Queues Across Different Bandwidths.

In summary, Figure 4.13 highlights the advantages of combining the XDP algorithm with CoDel. Although the packet loss rate increases, the frame loss rate decreases, demonstrating that the increased packet loss helps alleviate congestion in the network stack, which in turn reduces video stream latency and improves video-haptic synchronization. Meanwhile, the decrease in frame loss rate enhances video FPS at low bandwidth, allowing the video stream to maintain around 8 FPS even under extreme conditions, which can be sufficient for certain specific tasks. This analysis shows that XDP integration with CoDel provides valuable improvements in video FPS, latency, and synchronization offset, making it a strong option for managing performance under constrained bandwidth.

4.2.6 Effect of XDP on CAKE

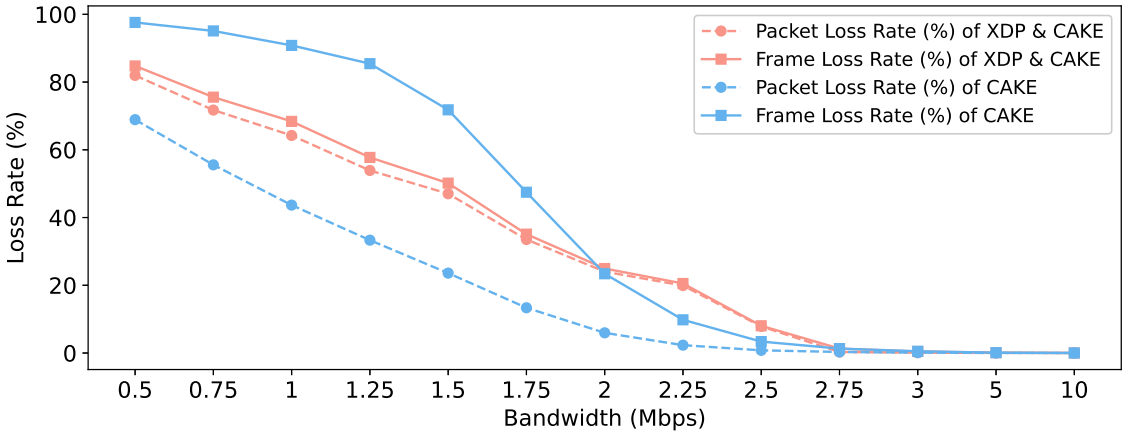
Figure 4.14 evaluates the impact of the eXpress Data Path (XDP) algorithm when combined with the CAKE (Common Applications Kept Enhanced) queue management system across various network performance metrics. CAKE is designed to improve fairness, reduce bufferbloat, and manage bandwidth efficiently by prioritizing latency-sensitive traffic. The addition of XDP, a high-performance packet processing solution, potentially enhances CAKE's queue management by accelerating packet handling. This figure includes five subplots, each examining a specific performance indicator: packet loss rate, frame loss rate, latency, frames per second (FPS), and synchronization offset between video and haptic streams.

Subfigure 4.14a displays the relationship between bandwidth and both the packet loss rate and frame loss rate for video traffic, comparing CAKE with XDP & CAKE. The XDP & CAKE algorithm performs well under both sufficient and low bandwidth conditions. When bandwidth is below 2 Mbps, it has a lower frame loss rate compared to the CAKE algorithm alone. However, at 2.25 Mbps and 2.5 Mbps, XDP & CAKE shows an increase in frame loss rate. This suggests that while XDP improves frame retention at lower bandwidths, the added processing complexity may impact performance at higher, yet constrained, bandwidth levels.

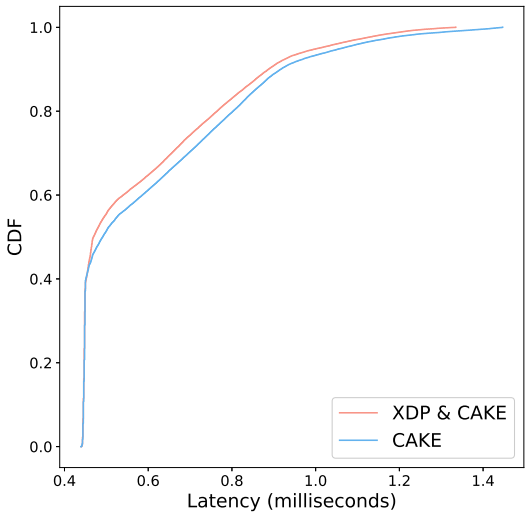
Subfigure 4.14b presents the latency distribution of the haptic stream as a CDF, contrasting CAKE and XDP & CAKE configurations. The CDF shows that both CAKE and XDP & CAKE meet the Quality of Service (QoS) requirements for the haptic stream.

Subfigure 4.14c illustrates the latency of the video stream as a function of bandwidth for both CAKE and XDP & CAKE configurations. Compared to CoDel, CAKE behaves more dynamically, as its latency does not consistently decrease with increasing bandwidth, even without XDP. However, CAKE maintains lower overall latency than CoDel-controlled queues. Interestingly, adding the XDP algorithm increases latency for the video stream, likely because CAKE's strict latency control makes any additional processing delay from XDP more noticeable. For example, at bandwidths of 0.5 Mbps, 0.75 Mbps, and 1 Mbps, the introduction of XDP increases latency by 5-10 ms. However, in the 1.75-2.5 Mbps range, adding XDP actually reduces CAKE's latency, indicating that the benefits of XDP may depend on specific bandwidth conditions.

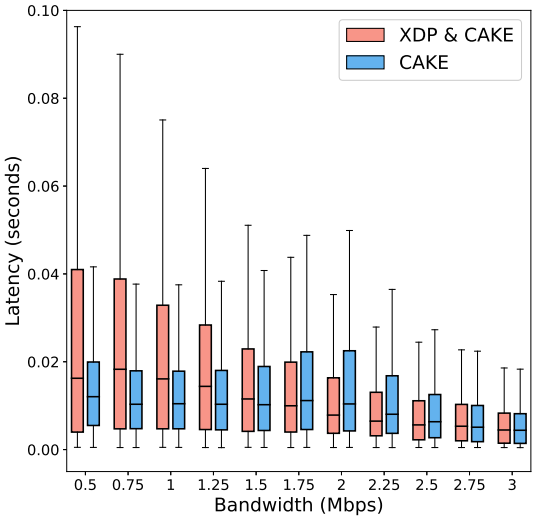
Subfigure 4.14d shows the frames per second (FPS) of the video stream across different bandwidth levels, used here as an indicator of video quality and playback stability. The box plots reveal that the XDP & CAKE configuration achieves higher median FPS values at nearly all bandwidths, especially under low bandwidth conditions. When bandwidth is in the range of 0.5-2 Mbps, FPS shows a significant increase with a narrower range between minimum and maximum values, indicating greater stability. The improvement in median FPS reaches up to 10 at 1.5 Mbps bandwidth. Even at extremely low bandwidth, around 0.5 Mbps, the median FPS remains above 5, which is sufficient for specific tasks. However, an exception occurs in the 2-2.5 Mbps range, where FPS shows a slight decrease. As noted in Subfigure 4.14a, packet loss at these bandwidth levels is higher with XDP & CAKE than with CAKE alone. Nevertheless, with FPS still around 30, the marginal drop of 5 frames is unlikely to impact most tasks significantly, as higher frame rates provide diminishing returns.



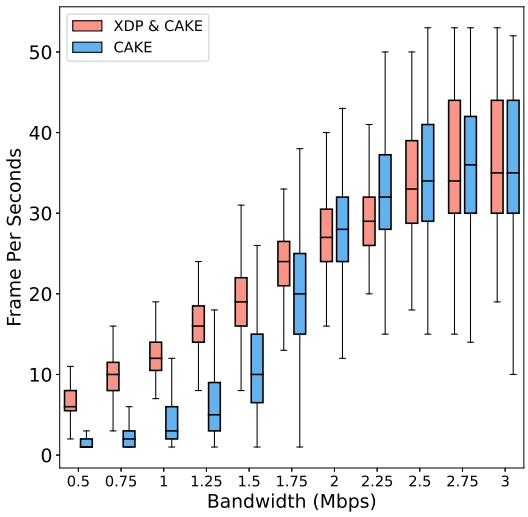
(a) Packet Loss Rate and Frame Loss Rate.



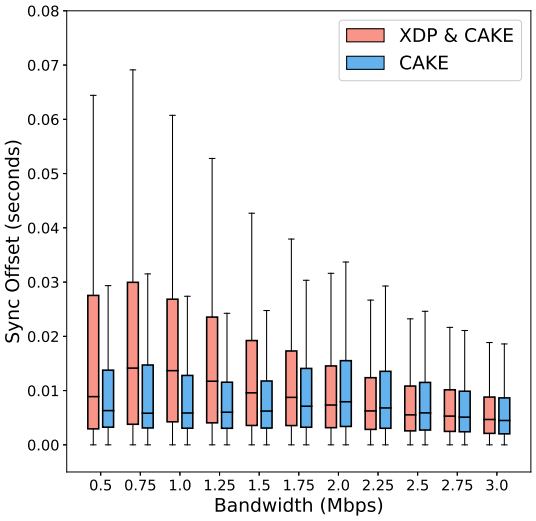
(b) CDF of Haptic Stream Latency.



(c) Video Stream Latency.



(d) FPS of Video.



(e) Video Haptic Synchronization Offset.

Figure 4.14: Impact of XDP Algorithm on CAKE-Managed Queues Across Different Bandwidths.

Subfigure 4.14e describes the synchronization offset between video and haptic streams, a key metric for real-time applications. Similar to the video stream, the haptic-video offset also increases with the addition of XDP, especially under low bandwidth conditions where the median offset rises by 5-10 ms. However, for both XDP & CAKE and CAKE alone, the median sync offset remains below 20 ms, which meets QoS requirements, ensuring that the offset remains within an acceptable range for maintaining synchronization.

In summary, Figure 4.14 highlights the impact of introducing the XDP algorithm on queues managed by CAKE. At low bandwidths, the improvement in video framerate is significant, providing a smoother viewing experience. Although FPS decreases at certain bandwidths, the resulting framerate remains sufficient for most tasks. A trade-off observed is the added latency in the video stream, which increases the haptic-video synchronization offset. However, due to CAKE's strict latency control, the increased synchronization offset still meets QoS requirements. This analysis demonstrates that combining XDP with CAKE provides notable benefits in video quality and stability while maintaining QoS-compliant synchronization for most real-time applications.

4.2.7 Between AQMs

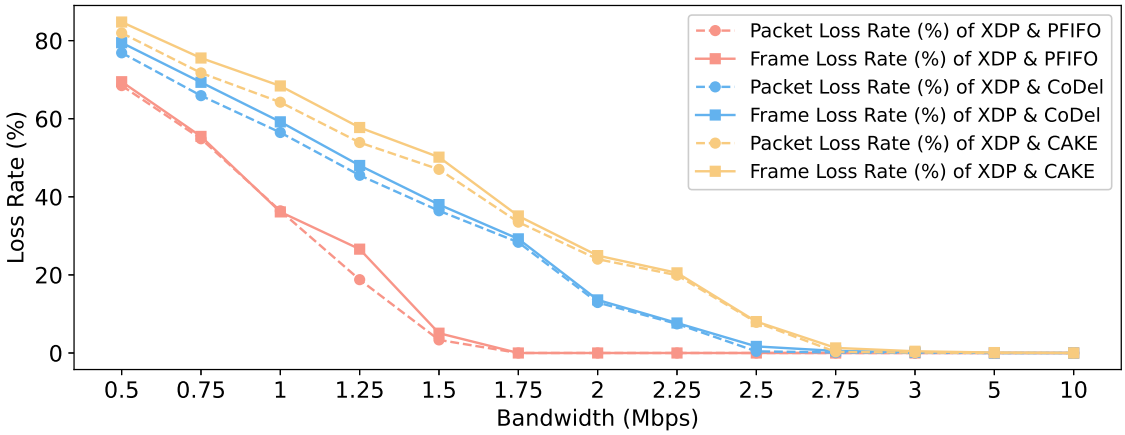
In this section, we compare metrics across different Active Queue Management (AQM) algorithms after introducing the XDP algorithm. As shown in Figure 4.15, a comparison is made between three queue management configurations: XDP & PFIFO, XDP & CoDel, and XDP & CAKE.

Subfigure 4.15a shows the changes in packet loss rate and frame loss rate across the different strategies. As bandwidth decreases, loss rates for all three strategies increase, reflecting the impact of bandwidth constraints on transmission stability. At the same bandwidth, XDP & PFIFO demonstrates the lowest packet loss rate, followed by XDP & CoDel, with XDP & CAKE having the highest loss rate. Through optimized parameter selection in the XDP algorithm, the packet loss rate and frame loss rate are very close, indicating that proactive packet dropping by XDP is highly efficient. This minimizes situations where a single lost packet in a frame prevents the frame from being decoded, which is especially beneficial for maintaining video quality.

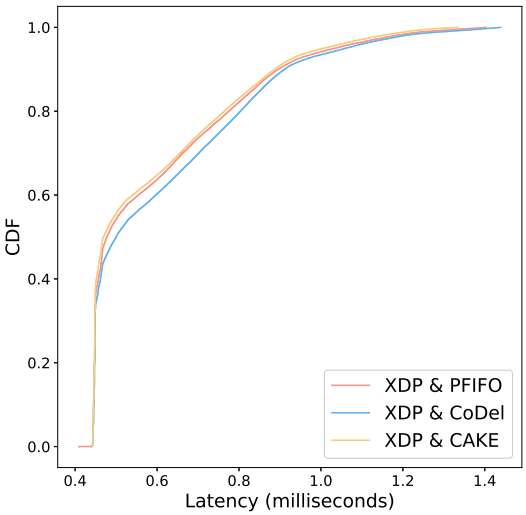
Subfigure 4.15b demonstrates that all strategies maintain very low haptic latency, meeting the Quality of Service (QoS) requirements for real-time applications. This result highlights the effectiveness of each AQM approach in providing low latency for haptic data, which is crucial for applications requiring immediate feedback.

Subfigure 4.15c presents a latency comparison between the different strategies, with latency shown on a logarithmic scale for clarity. As bandwidth increases, latency decreases across all three strategies, reflecting the direct relationship between available bandwidth and reduced transmission delays. However, at low bandwidth levels, PFIFO, which does not perform proactive packet dropping, experiences latency far above acceptable QoS levels and is therefore excluded from further discussion. Among the remaining two strategies, XDP & CAKE demonstrates a significant latency reduction compared to XDP & CoDel, reducing latency by approximately 25 ms at a bandwidth of 0.5 Mbps. This suggests that XDP & CAKE's approach to latency control is highly effective under constrained conditions.

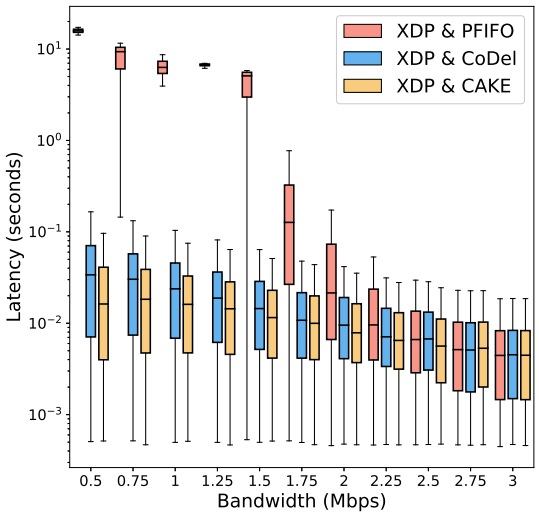
Subfigure 4.15d compares the frame rates achieved across the different strategies. PFIFO achieves a higher frame rate; however, its latency far exceeds acceptable QoS levels, making FPS irrelevant for discussion here. Among the remaining strategies, XDP & CoDel achieves a higher frame rate than XDP & CAKE, with an improvement of up to about 5 FPS in the 0.5-2.5 Mbps bandwidth range. This difference in frame rate is particularly meaningful at lower bandwidths, where increases in frame rate yield greater benefits in terms of Mean Opinion Score (MOS) at lower initial frame rates (as previously shown in Figure 2.1), enhancing the overall viewing experience.



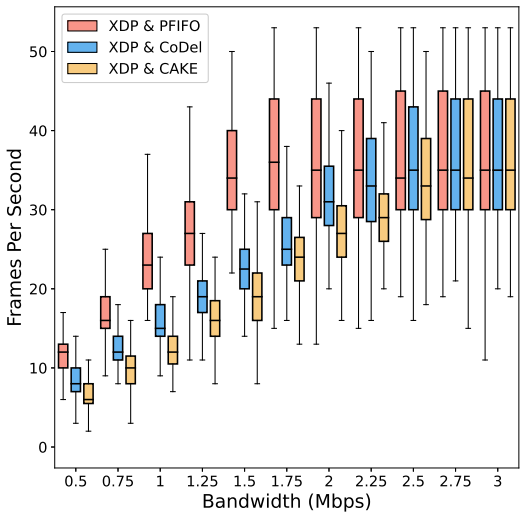
(a) Packet Loss Rate and Frame Loss Rate.



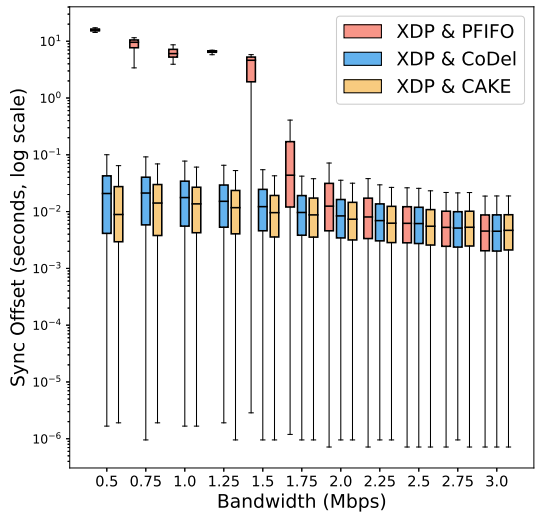
(b) CDF of Haptic Stream Latency.



(c) Video Stream Latency.



(d) FPS of Video.



(e) Video-Haptic Synchronization Offset.

Figure 4.15: Comparison of XDP algorithm implement on different AQMs.

Subfigure 4.15e compares haptic-video synchronization offset across the different strategies, excluding PFIFO from the discussion due to its high latency. Similar to Subfigure 4.15c, XDP & CAKE also outperforms XDP & CoDel in terms of synchronization offset, with the greatest improvement of around 10 ms occurring at a bandwidth of 0.5 Mbps. This result underscores XDP & CAKE's ability to maintain tighter synchronization between video and haptic data, which is essential for applications that require precise timing alignment between streams.

In summary, Figure 4.15 illustrates the impact of introducing XDP into different queue management mechanisms. XDP & CAKE performs better in terms of reducing latency and synchronization offset, making it well-suited for applications prioritizing these metrics. Meanwhile, XDP & CoDel achieves higher FPS, which is valuable for applications where frame rate is a critical quality metric. Depending on specific requirements, either strategy can be chosen. However, XDP & CoDel is generally more practical, as it already meets QoS requirements for video latency, making the FPS improvement more impactful in this context.

5 Future work

This thesis has demonstrated the effectiveness of implementing the XDP algorithm on different AQM strategies with managed queues, particularly when combined with CAKE and CoDel, in maintaining video quality and haptic responsiveness under constrained network conditions. However, several areas remain where the system could be further optimized and refined.

Future work could focus on expanding the data collection testbench to simulate a wider range of network conditions and device types. This would enable a more comprehensive understanding of algorithm performance across various environments. Additionally, integrating metrics that capture user experience, such as Mean Opinion Score (MOS) for video quality, would provide a more nuanced view of how different configurations affect perceived performance.

At present, the XDP algorithm cannot be directly implemented on the data collection testbench, as WebRTC's adaptive stream adjustment introduces instability when XDP is applied. With hardware support, such as a high-performance graphics card for encoding and decoding, a stable video stream could be achieved with FFmpeg, allowing for more accurate testing of the algorithm's real-world effects. This approach could also help validate the stability and performance of the XDP algorithm in practical applications.

It's important to note that our study uses MOS results from previous research to map the impact of FPS on user experience. However, in reality, MOS is influenced by many factors, such as the specific tasks being performed and the equipment used. Future research could involve designing experiments to assess how various device parameters affect MOS under different conditions, providing a more accurate evaluation of user experience. This direction could yield valuable insights for improving system performance in diverse real-world scenarios.

The XDP algorithm also holds potential for refinement. Combining XDP with CAKE and CoDel showed clear benefits in terms of frame rate and synchronization, but further investigation could focus on achieving an optimal balance between packet loss and latency. This could involve fine-tuning XDP's integration with these queue management algorithms to dynamically adjust based on network load, potentially combining the strengths of CAKE's latency control with CoDel's packet handling.

In summary, these proposed directions for future work aim to enhance the adaptability, user satisfaction, and overall performance of the system. By refining the algorithm, optimizing XDP with CAKE and CoDel, incorporating user-centered feedback, and developing hardware-supported testing setups, future research could produce a more robust solution for maintaining high-quality, low-latency video and haptic feedback in bandwidth-constrained environments. Additionally, assessing how device parameters influence MOS in practical scenarios would provide a more comprehensive understanding of user experience under various conditions.

6 Conclusion

This thesis includes the setup of a testbench for data collection, a testbench for running the algorithm, and the design, implementation, and testing of the proposed algorithm.

The data collection testbench demonstrated strong performance, achieving an average screen-to-screen video delay of 72.23 ms and an average end-to-end haptic delay of 14.41 ms. These values meet QoS standards for most remote control tasks, ensuring responsive and accurate feedback for both video and haptic interactions.

For algorithm implementation, AIMD3 was chosen as the P controller strategy on an OpenWrt router due to its ability to meet QoS latency requirements under low bandwidth conditions while maintaining a high frame rate. AIMD3's balance of low latency and high FPS makes it particularly suitable for applications requiring real-time responsiveness in constrained bandwidth environments.

When integrated with Active Queue Management (AQM), AIMD3 significantly enhances framerate, especially in low bandwidth conditions (0.5-1.75 Mbps), by increasing FPS by up to 10 frames. This improvement in FPS contributes to a smoother visual experience, enhancing video quality in scenarios where bandwidth is limited. Additionally, AIMD3 demonstrates strong performance in latency control when used with a CoDel-managed queue, reducing the median haptic-video synchronization offset to around 20 ms. This reduction in offset helps maintain synchronization between video and haptic streams, which is critical for real-time applications.

Comparing the two configurations, XDP & CAKE and XDP & CoDel, each has unique strengths in managing haptic-video synchronization offset and framerate. The choice between these configurations depends on specific application needs and performance priorities. In a fixed channel, these two queue management algorithms present a trade-off between video FPS and video-haptic synchronization offset: higher packet loss reduces packets in the channel, resulting in a lower video FPS but with reduced synchronization offset. Conversely, lower packet loss leads to a slightly higher synchronization offset but also a higher video FPS, providing smoother playback at the expense of increased delay. However, since the synchronization offset of both can meet QoS requirements, priority should be given to higher framerate in this case. Therefore, XDP & CoDel is the more effective choice.

In conclusion, this thesis demonstrates that the choice of queue management strategy and P controller configuration should align with the specific QoS requirements of the application. AIMD3, particularly when implemented with CoDel, achieves a balance of low latency and high FPS, while XDP & CAKE and XDP & CoDel provide tailored options for prioritizing synchronization or frame rate, respectively. Together, these insights offer valuable guidance for optimizing real-time video and haptic feedback in constrained network conditions.

List of Figures

List of Tables

Bibliography

[1] W. R. Ferrell and T. B. Sheridan, "Supervisory control of remote manipulation," *IEEE spectrum*, vol. 4, no. 10, pp. 81–88, 1967.

[2] E. Steinbach, S. Hirche, M. Ernst, F. Brandi, R. Chaudhari, J. Kammerl, and I. Vittorias, "Haptic communications," *Proceedings of the IEEE*, vol. 100, no. 4, pp. 937–956, 2012.

[3] G. Kokkonis, K. E. Psannis, M. Roumeliotis, and Y. Ishibashi, "Efficient algorithm for transferring a real-time hevc stream with haptic data through the internet," *Journal of Real-Time Image Processing*, vol. 12, pp. 343–355, 2016.

[4] H. T. Tran, N. P. Ngoc, C. T. Pham, Y. J. Jung, and T. C. Thang, "A subjective study on qoe of 360 video for vr communication," in *2017 IEEE 19th international workshop on multimedia signal processing (MMSP)*, pp. 1–6, IEEE, 2017.

[5] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, "Controlled Delay Active Queue Management." RFC 8289, Jan. 2018.

[6] T. Høiland-Jørgensen, D. Täht, and J. Morton, "Piece of cake: a comprehensive queue management solution for home gateways," in *2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pp. 37–42, IEEE, 2018.

[7] T. Høiland-Jørgensen, J. D. Brouer, D. Borkmann, J. Fastabend, T. Herbert, D. Ahern, and D. Miller, "The express data path: Fast programmable packet processing in the operating system kernel," in *Proceedings of the 14th international conference on emerging networking experiments and technologies*, pp. 54–66, 2018.

[8] S. Din and D. Bulterman, "Synchronization techniques in distributed multimedia presentation," *MMEDIA 2012*, p. 9, 2012.

[9] J. M. Silva, M. Orozco, J. Cha, A. E. Saddik, and E. M. Petriu, "Human perception of haptic-to-video and haptic-to-audio skew in multimedia applications," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 9, no. 2, pp. 1–16, 2013.

[10] H. Kim and J.-H. Ryu, "A study on the effect of haptic to video time-delay on teleoperation and a comment for improving the performance," in *2011 11th International Conference on Control, Automation and Systems*, pp. 1329–1332, IEEE, 2011.

[11] S. Kameyama and Y. Ishibashi, "Perception of synchronization errors in haptic and visual communications," in *Multimedia Systems and Applications IX*, vol. 6391, pp. 67–74, SPIE, 2006.

[12] B. Cizmeci, X. Xu, R. Chaudhari, C. Bachhuber, N. Alt, and E. Steinbach, "A multiplexing scheme for multimodal teleoperation," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 13, no. 2, pp. 1–28, 2017.

[13] J. Cha, M. Eid, A. Barghout, A. M. Rahman, and A. El Saddik, "Hugme: synchronous haptic teleconferencing," in *Proceedings of the 17th ACM international conference on Multimedia*, pp. 1135–1136, 2009.

[14] M. Eid, J. Cha, and A. El Saddik, "An adaptive multiplexer for multi-modal data communication," in *2009 IEEE International Workshop on Haptic Audio visual Environments and Games*, pp. 111–116, IEEE, 2009.

[15] M. Eid, J. Cha, and A. El Saddik, "Admux: An adaptive multiplexer for haptic–audio–visual data communication," *IEEE Transactions on Instrumentation and Measurement*, vol. 60, no. 1, pp. 21–31, 2010.

[16] M. Eid, A. El Issawi, and A. El Saddik, "Slingshot 3d: A synchronous haptic-audio-video game," *Multimedia tools and applications*, vol. 71, pp. 1635–1649, 2014.

[17] J. Zhang, Y. Li, and Y. Wei, "Using timestamp to realize audio-video synchronization in real-time streaming media transmission," in *2008 International Conference on Audio, Language and Image Processing*, pp. 1073–1076, IEEE, 2008.

[18] Y. Xu, L. Huang, T. Zhao, Y. Fang, and L. Lin, "A timestamp-independent haptic–visual synchronization method for haptic-based interaction system," *Sensors*, vol. 22, no. 15, p. 5502, 2022.

[19] M. Ma, Y. Yang, T. V. Doan, O. Lhamo, F. H. Fitzek, and G. T. Nguyen, "Syncodel: Network-assisted synchronization of video and haptic streams for teleoperations," in *2024 IEEE 49th Conference on Local Computer Networks (LCN)*, pp. 1–6, IEEE, 2024.

[20] F. Baker, D. L. Black, K. Nichols, and S. L. Blake, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers." RFC 2474, Dec. 1998.

[21] R. T. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification." RFC 2205, Sept. 1997.

[22] P. Hinterseer, E. Steinbach, S. Hirche, and M. Buss, "A novel, psychophysically motivated transmission approach for haptic data streams in telepresence and teleaction systems," in *Proceedings.(ICASSP'05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.*, vol. 2, pp. ii–1097, IEEE, 2005.

[23] S. Clarke, G. Schillhuber, M. F. Zaeh, and H. Ulbrich, "Telepresence across delayed networks: a combined prediction and compression approach," in *2006 IEEE International Workshop on Haptic Audio Visual Environments and their Applications (HAVE 2006)*, pp. 171–175, IEEE, 2006.

[24] P. Hinterseer, S. Hirche, S. Chaudhuri, E. Steinbach, and M. Buss, "Perception-based data reduction and transmission of haptic data in telepresence and teleaction systems," *IEEE Transactions on Signal Processing*, vol. 56, no. 2, pp. 588–597, 2008.

[25] O. Dabeer and S. Chaudhuri, "Analysis of an adaptive sampler based on weber's law," *IEEE transactions on signal processing*, vol. 59, no. 4, pp. 1868–1878, 2010.

[26] N. Sakr, N. D. Georganas, and J. Zhao, "Human perception-based data reduction for haptic communication in six-dof telepresence systems," *IEEE Transactions on Instrumentation and Measurement*, vol. 60, no. 11, pp. 3534–3546, 2011.

[27] R. M. Nasiri, J. Wang, A. Rehman, S. Wang, and Z. Wang, "Perceptual quality assessment of high frame rate video," in *2015 IEEE 17th International Workshop on Multimedia Signal Processing (MMSP)*, pp. 1–6, IEEE, 2015.

[28] J. Y. Chen and J. E. Thropp, "Review of low frame rate effects on human performance," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 37, no. 6, pp. 1063–1076, 2007.

[29] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on networking*, vol. 1, no. 4, pp. 397–413, 1993.

[30] S. Floyd, R. Gummadi, S. Shenker, *et al.*, "Adaptive red: An algorithm for increasing the robustness of red's active queue management," 2001.

[31] T. Høiland-Jørgensen, P. McKenney, dave.taht@gmail.com, J. Gettys, and E. Dumazet, "The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm." RFC 8290, Jan. 2018.

[32] W.-c. Feng, K. G. Shin, D. D. Kandlur, and D. Saha, "The blue active queue management algorithms," *IEEE/ACM transactions on networking*, vol. 10, no. 4, pp. 513–528, 2002.

[33] A. S. Tanenbaum, *Computer Networks*. Pearson Education India, 2003. p. 408.

[34] S. McCanne and V. Jacobson, "The bsd packet filter: A new architecture for user-level packet capture.," in *USENIX winter*, vol. 46, pp. 259–270, Citeseer, 1993.

[35] eBPF.io, "What is ebpf?." https://ebpf.io/what-is-ebpf/#articles--blogs, 2024. Accessed on: October 2, 2024.

[36] M. A. Vieira, M. S. Castanho, R. D. Pacífico, E. R. Santos, E. P. C. Júnior, and L. F. Vieira, "Fast packet processing with ebpf and xdp: Concepts, code, challenges, and applications," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–36, 2020.

[37] IOvisor, "Iovisor project." http://www.iovisor.org, 2024. Accessed: October 2, 2024.

[38] VMWare, "p4c-xdp." https://github.com/vmware/p4c-xdp, 2024. Accessed: October 2, 2024.

[39] BCC, "Bpf compiler collection." https://github.com/iovisor/bcc, 2024. Accessed: October 2, 2024.

[40] Cilium, "ebpf library for go." https://github.com/cilium/ebpf, 2024. Accessed: October 2, 2024.

[41] eBPF Documentation, "ebpf maps - concepts." https://docs.ebpf.io/linux/concepts/maps/, 2024. Accessed: October 2, 2024.

[42] IOVisor, "Bpf features by linux kernel version." https://github.com/iovisor/bcc/blob/master/docs/kernel-versions.md, 2024. Accessed: October 2, 2024.

[43] G. Bertin, "Xdp in practice: integrating xdp into our ddos mitigation pipeline," in *Technical Conference on Linux Networking, Netdev*, vol. 2, pp. 1–5, The NetDev Society, 2017.

[44] Facebook, "Katran project." https://github.com/facebookincubator/katran, 2024. Accessed: October 2, 2024.

[45] Microsoft, "Xdp for windows." https://github.com/microsoft/xdp-for-windows/, 2024. Accessed: October 2, 2024.

[46] WebRTC Project, "WebRTC: Real-Time Communication for the Web." https://webrtc.org, 2024. Accessed: 2024-11-08.

[47] World Wide Web Consortium (W3C), "W3C Announces WebRTC as a Standard for Real-Time Communication." https://www.w3.org/press-releases/2021/webrtc-rec, 2021. Accessed: 2024-11-08.

[48] Internet Engineering Task Force (IETF) RTCWeb Working Group, "RTCWeb Working Group Documents." https://datatracker.ietf.org/wg/rtcweb/documents/, 2024. Accessed: 2024-11-08.

[49] Can I use, "WebRTC Browser Compatibility." https://caniuse.com/?search=webrtc, 2024. Accessed: 2024-11-08.

[50] A. Roach, "WebRTC Video Processing and Codec Requirements." RFC 7742, Mar. 2016.

[51] Mozilla Developer Network (MDN), "WebRTC Codecs: Supported Media Formats in WebRTC." https://developer.mozilla.org/en-US/docs/Web/Media/Formats/WebRTC_codecs, 2024. Accessed: 2024-11-08.

[52] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN systems*, vol. 17, no. 1, pp. 1–14, 1989.

[53] JaidedAI, "Easyocr." https://github.com/JaidedAI/EasyOCR, 2024. Accessed: October 2, 2024.