eXpressSFU: Toward Super-Scalable Video Conferencing with SmartNICs

Tuan Tran[†], *S. M. H. Hosseini*[†], *Seyeon Kim*[‡], *Kyunghan Lee*[§], *Nam Bui*[¶], *Dirk Grunwald*[†], *Sangtae Ha*[†]
[†]*University of Colorado Boulder* [‡]*Korea University* [§]*Seoul National University* [¶]*University of Colorado Denver*
{tuan.tran, hosseini, dirk.grunwald, sangtae.ha}@colorado.edu,
seyeon625@korea.ac.kr, kyunghanlee@snu.ac.kr, nam.bui@ucdenver.edu

Abstract

Video conferencing has emerged as a critical Internet application. Unlike video-on-demand services, high-quality video conferencing necessitates minimal latency, as media streams are generated, transmitted, processed, forwarded and received in real time. Our empirical analysis reveals that processing latency at the media server, particularly Selective Forwarding Units (SFUs), is the dominant barrier to scalability. Notably, cryptography, memory, and I/O operations account for approximately 79% of the media packet processing latency.

In this paper, we introduce *eXpressSFU*, a re-architected video conferencing system designed to significantly enhance scalability for large-scale and high-quality video conferences. By decoupling the fast-control and data planes from the slow-control plane, *eXpressSFU* accelerates the media packet processing pipeline through the use of emerging network technologies, such as Smart Network Interface Cards (SmartNICs). Experimental results show that our system reduces packet processing latency by a factor of 8. This improvement allows it to support 3× more concurrent users while cutting computational power consumption by up to 60%.

1 Introduction

Video conferencing has experienced skyrocketing growth, transforming it from an occasional business tool into an essential communication channel. This change has been driven by the increasing demand for remote work [29], online classes [24], and remote collaboration [56]. According to the latest report from Zoom, the most popular video conferencing platform, there are 300 million daily participants in meetings [53]. This represents a remarkable growth with the average yearly increase rate of 87.8 million since their record of 10 million in December 2019 [14]. The video conferencing market is now projected to reach $9.6 billion by 2028, growing at a compound annual rate of 8.35% [8].

Today's video conferencing demands have shifted significantly toward accommodating higher video quality streams, such as 1080p. The rise of immersive applications like Augmented Reality (AR) and Virtual Reality (VR), which necessitate exceptionally high-quality video streams with minimal latency, further amplifies this requirement [16, 52]. To meet these growing demands, modern video conferencing systems have transitioned from the compute-intensive Multipoint Control Unit (MCU) architecture to the more scalable Selective Forwarding Unit model [1, 26]. By distributing encoding and decoding tasks to the endpoints, SFUs enhance scalability. However, as the number of participants increases, processing latency becomes a critical bottleneck, reducing video quality through reduced bitrates and increased frame drops [2, 32].

Previous research has attempted to address the SFU limitations through software optimization of video codecs and application stacks [15, 22, 44, 46]. On the other hand, the industry's standard approach to handling growing demands is horizontal scaling—deploying more server instances to distribute conference load [4, 13, 34]. However, these approaches only mitigate symptoms rather than addressing the fundamental architectural constraints.

The main reason these approaches were ineffective is the inherent latency bottlenecks within current architecture, which executes the entire SFU stack on x86 CPU. These bottlenecks emerge at the intersection of two opposing trends: the exponential growth in video data rates and concurrent users [5, 18] versus the plateauing advancement in CPU and memory technologies. This widening gap demands innovative architectural solutions that go beyond traditional software optimizations. A promising approach is to re-architect the SFU system by offloading latency sensitive packet processing tasks to emerging hardware accelerators, such as SmartNICs.

To this end, we propose *eXpressSFU*, a novel SFU architecture informed by empirical insights and in-depth analysis of Jitsi Meet [11], one of the most widely adopted open-source SFU platforms. By processing media packets directly on the SmartNIC, *eXpressSFU* can transmit outbound traffic as early as possible, bypassing the host entirely. Most importantly, it significantly accelerates data plane operations to meet the stringent low-latency requirements of high-quality video con-

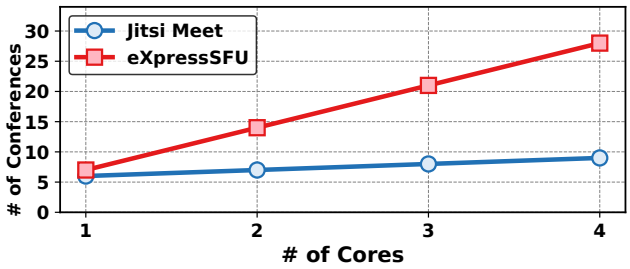


Figure 1: Number of concurrent conferences without quality degradation. Software-based solution struggles to scale on host CPU cores, while our *eXpressSFU* shows significantly better scalability in handling FullHD streams on SmartNIC ARM cores with a single host CPU core as control plane.

ferencing. As shown in Figure 1, conventional software-based solutions struggle to handle multiple concurrent 15-user conferences at 1080p@24fps, even when additional CPU cores are allocated, leading to limited scalability. In contrast, *eXpressSFU* achieves much better scaling, supporting an increasing number of conferences as additional, albeit significantly less powerful, SmartNIC ARM cores are utilized. *eXpressSFU* leverages the decade-long of software engineering efforts by retaining the majority of the existing SFU system while selectively decoupling both the data plane and the fast-control plane from the original SFU stack. This architectural separation enables latency-sensitive operations to be offloaded and accelerated on the SmartNIC. Furthermore, *eXpressSFU* maintains broad compatibility with diverse endpoint devices by supporting dynamic quality switching for both Simulcast and Scalable Video Coding (SVC) natively on the SmartNIC.

We evaluate *eXpressSFU* on a cluster comprising a workstation equipped with a Nvidia BlueField-2 SmartNIC [39] and an Intel i9-14900K CPU serving as the SFU server, alongside 100 bare-metal client machines running Chrome Driver for highly realistic evaluation. By measuring the maximum number of concurrent users or conferences served by *eXpressSFU* against a state-of-the-art SFU system (e.g, Jitsi Meet), we demonstrate the scalability advantages of our design.

The main contributions of this paper are four-fold:

- We investigate the underlying causes of scalability bottlenecks by stress-testing Jitsi Meet and Janus across diverse workloads. Particularly, we provide a detailed latency breakdown, pinpoint the root causes of latency increases, and explain their impacts on media quality.
- We propose a novel SFU architecture that offloads latency-sensitive processing to a SmartNIC while preserving the core system logic to minimize disruption.
- We enable native support for both Simulcast and SVC on the SmartNIC, ensuring compatibility with a wide range of endpoint devices.
- We implement *eXpressSFU* using DPDK on the SmartNIC in a transparent and minimally invasive manner, requiring only minimal modifications to Jitsi Meet. We evaluate *eXpressSFU* against the latest Jitsi Meet release, demonstrating substantial performance and scalability gains over this widely-used SFU platform.

2 Background

2.1 Selective Forwarding Unit

A selective forwarding unit (SFU) is a media packet bridge that consists of tightly coupled control and data planes. The control plane handles rate adaptation by collecting receiver feedback to adjust sending rates and optimize Quality of Experience (QoE), while the data plane performs packet-level operations on ingress and egress streams without transcoding. Unlike the earlier MCU, which decoded, mixed, and re-encoded frames into a composite stream, the SFU forwards packets directly for lower latency and higher scalability.

❶ Receive and decrypt streams (data plane). The SFU receives media streams of varying qualities—either multiple streams in simulcast or a single layered stream in SVC. Each stream, identified by its Synchronization Source (SSRC), represents a bitrate-quality option to match diverse network and device conditions. The SFU decrypts incoming streams so the control plane can access metadata for bitrate control (§2.2).

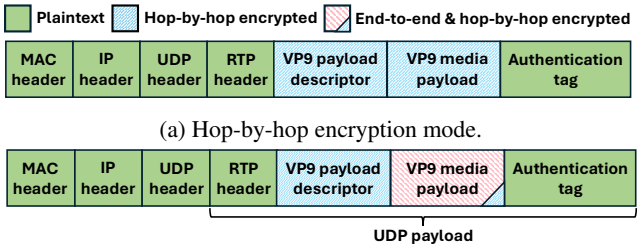
❷ Estimate and adapt bitrates (control plane). The SFU analyzes per-packet metadata and receiver feedback to select the appropriate quality level per endpoint, maximizing bandwidth utilization while maintaining low E2E latency and frame jitter.

❸ Clone, re-encrypt, and forward (data plane). Based on control plane decisions, the SFU clones selected packets, applies per-receiver encryption keys, and forwards them ensuring secure, low-latency delivery at the chosen quality level.

2.2 Bitrate Control

Simulcast. The sender encodes the same video into multiple independent streams at distinct bitrates and quality levels, transmitting them simultaneously to the SFU. The SFU then selects and forwards the most suitable stream to each receiver based on factors such as network conditions, device capabilities, and display needs. For example, a user on a high-speed connection and a large display might receive the 1080p@24fps stream, whereas another user with limited bandwidth may be served a 360p@12fps version. Although Simulcast offers per-user bitrate flexibility to individual users, it imposes significant uplink bandwidth and computational demands on the sender [35]. As a result, it is increasingly being replaced by a more efficient alternative: SVC.

SVC. The sender encodes video into a single stream composed of multiple layers. These layers consist of a mandatory base layer—which provides a minimal-resolution, low-framerate version—and additional enhancement layers built on top of it. By encoding all quality levels into a single stream,



(a) Hop-by-hop encryption mode.

(b) End-to-end encryption mode.

Figure 2: Encrypted fields of VP9 media packet. Hop-by-hop encryption is mandatory in both encryption modes. End-to-end encryption is optionally applied to the media payload before hop-by-hop encrypted.

SVC reduces redundancy, encoding overhead, and uplink bandwidth. The SFU selectively forwards required layers to each receiver, thereby ensuring optimal quality according to individual bandwidth and device constraints. Google's latest video encoding format, VP9 [25], provides native support for SVC, offering both temporal and spatial scalability. Temporal adjusts the framerate, whereas spatial adjusts the resolution.

2.3 Cryptography

Hop-by-hop encryption (mandatory). WebRTC-based SFU systems ensure secure communication between each user and the SFU by enforcing hop-by-hop encryption with Datagram Transport Layer Security (DTLS) as key negotiation protocol, Secure Real-time Transport Protocol (SRTP) for media, and Secure Real-time Transport Control Protocol (SRTCP) for control messages. During conference setup, each participant performs a DTLS handshake to establish a shared secret with the SFU. This process creates encrypted channels for media exchange while still allowing the SFU to inspect packet metadata necessary for forwarding decisions. Specifically, senders encrypt packets before sending to the SFU. Upon reception, the SFU decrypts packets to extract information such as the layering fields in the VP9 payload descriptor (Figure 2a), re-encrypts them with per-recipient keys, and forwards. In essence, each packet is unique despite carrying the same media payload. While hop-by-hop encryption secures media streams during transit, it still allows the SFU to access media content in plaintext, raising potential privacy concerns.

End-to-end encryption (optional). Due to privacy concerns associated with hop-by-hop encryption , end-to-end encryption (E2EE) has been introduced as an advanced feature. As illustrated in Figure 2b, the sender first encrypts the media payload using keys known only among the users. These E2EE keys are negotiated among users at the start of the conference, with the SFU merely acting as a message relay. After end-to-end encryption is applied, hop-by-hop encryption proceeds as usual, leading to a double-encrypted media payload. This setup prevents the SFU from accessing the media content while still allowing it to access essential layering information.

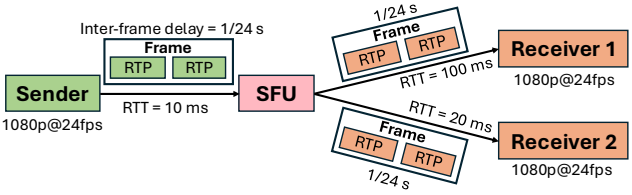


Figure 3: Receivers adjust video quality based on inter-frame variance, not total latency; that inter-frame variance is determined by the per-packet latency and the number of receivers.

3 SFU Processing Insights

3.1 Latency Budget for SFU Processing

Video conferencing is sensitive to latency and jitter, or variance in the latency; increased latency creates lag between participants, but jitter is used as an indication that links are congested and the video controller should drop back to a lower resolution [6]. If the inter-packet arrival interval drops below the frame rate, the rate controller signals the sender (or SFU) to drop video rates. For a video stream at 24 frames per second, that means frames need to arrive with $1/24s \approx 41ms$. When the SFU replicates a stream, each frame needs to arrive in $\approx 41ms$; since most frames in a 1080p@24 fps stream use 2 packets, the per-packet processing time should be 20.5ms or less. This is illustrated in Figure 3, where two clients may have varying RTTs (20ms vs. 100ms) but each needs to receive the two packets making up a frame within 41ms.

Since each incoming packet is duplicated for each client, the inter-packet latency for any given client depends on the total number of clients being served. If the SFU takes 1ms to clone and send a single packet, the time to clone and send packets to 30 clients would be 30ms, larger than the 20.5ms allowed; this would cause the clients to decrease video quality.

We measured the latency of each component in the media packet processing pipeline. Figure 4 shows the per-packet processing latency for Jitsi [11] and Janus [31], two widely used SFUs. These latencies are challenging to improve due to the tightly coupled control and data planes design of current SFU architecture. In particular, latency-sensitive data plane operations can only be executed after the control plane has made a per-packet forwarding decision for each and every receiver, introducing unavoidable processing dependencies.

Although-packet processing is only $\approx 0.250ms$ with 125 users, the resulting inter-packet latency at each client is $0.25 \times 125 = 31.25$ms, which is above the ≈ 20.5ms that triggers frame loss and switching to a reduced video quality. Figure 5 shows the reported frame drop rate from the receiver as we increase the number of concurrent users. As expected, when the number of concurrent users reaches 125, or per-user inter-packet delay exceeds 20.5ms, the frame drop rate increases significantly. On our SFU hardware, Jitsi Meet could only handle a single stream with 100 clients at 1080p quality, 180 clients at 720p quality and 200 clients at 360p quality.

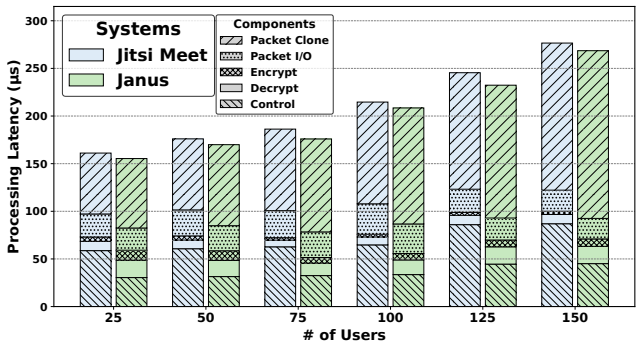


Figure 4: Per-packet video processing latency breakdown. As more users join the conference, packet cloning becomes the latency bottleneck of the current SFU systems.

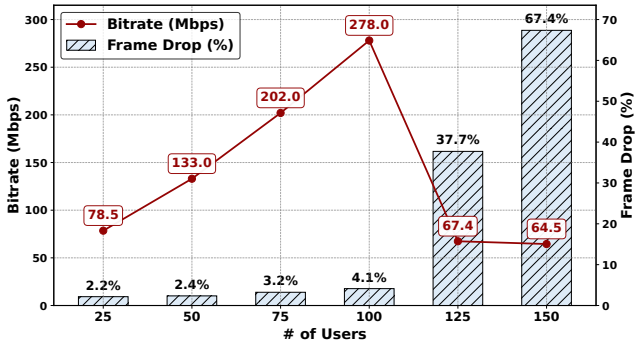


Figure 5: The impact of rising concurrent users on frame drops and video bitrate when using a single core SFU. Once the number of concurrent users reaches 125, high packet processing latency in the SFU leads to substantial frame drops, triggering receiver feedback that forces the SFU to lower the sending rate, thus, degrading video quality.

3.2 Opportunities of SmartNIC Acceleration

To explore the potential of SmartNIC acceleration, we compare the per-operation latency of identical dataplane tasks executed by software-based implementation on the host CPU and our DPDK-based counterpart on the SmartNIC's ARM. Table 1 shows that all operations run markedly faster on the SmartNIC than on the x86 CPU when measured on a single core in each case: encryption and decryption accelerate by roughly 3× and 8×, respectively. Packet cloning, the dominant source of latency, improves by about 4×. Notably, packet I/O latency is neglectable on SmartNIC.

Two factors drive these gains. First, the SmartNIC exploits DPDK's batch-processing acceleration. Because each incoming packet must be cloned and re-encrypted for multiple receivers, the workload naturally forms sizeable batches, allowing cryptographic and packet cloning costs to be amortized. Second, packet transceiver are handled entirely by the NIC's DMA engines, freeing the on-board ARM cores. Control packets, which account for only 16% of total traffic, will be processed on the host since it remains well within the processing capabilities of the host CPU. As demonstrated in Appendix §A, avoiding PCIe traversal between the host and NIC is crucial for maintaining frame timeliness, reinforcing the advantage of executing data plane operations entirely on the SmartNIC.

Table 1: Latency Comparison of Dataplane Operations.

Operations	Host CPU (ns)	SmartNIC ARM (ns)
Packet Cloning	1029	241
Packet Decryption	9732	1253
Packet Encryption	3380	1247
Packet I/O	966	24

4 eXpressSFU Design

In this section, we outline four key design considerations, followed by how we realize *eXpressSFU*, which offloads data plane operations to SmartNICs—thereby dramatically enhancing scalability. The design overview of *eXpressSFU* is presented in Figure 6.

4.1 Design Considerations

Leverage existing SFU development. Careful architectural design is essential to leverage the benefits of over a decade of SFU software development while dramatically accelerating data plane operations on SmartNIC hardware. Most importantly, because SFU software continues to evolve, a transparent design allows *eXpressSFU* to seamlessly adopt future improvements.

Control and data plane separation. An SFU is a complex system composed of tightly integrated components. To achieve both transparency and high performance in media packet processing, it is critical to determine which components should remain on the host CPU and which should be offloaded to the SmartNIC. While SmartNICs provide low-latency, high-throughput packet processing, the host x86 CPU is far more powerful and better suited for complex logic and algorithms. Therefore, *eXpressSFU* must strike a balance between performance and complexity by assigning tasks to the most suitable processing unit.

Support for both legacy and modern devices. SVC has become the *de facto* standard for video conferencing due to its efficient bandwidth usage and flexible quality layer switching. However, many legacy devices still lack SVC support. To ensure broad compatibility, *eXpressSFU* must support both Simulcast and SVC, enabling adaptive bitrate functionality regardless of endpoint capabilities.

SFU system optimization for high performance Maximizing packet processing performance on SmartNICs requires careful core-level workload distribution. Two key objectives must be met: evenly balancing the workload across cores and preserving per-stream packet ordering since out-of-order packets can trigger WebRTC congestion control. Additionally,

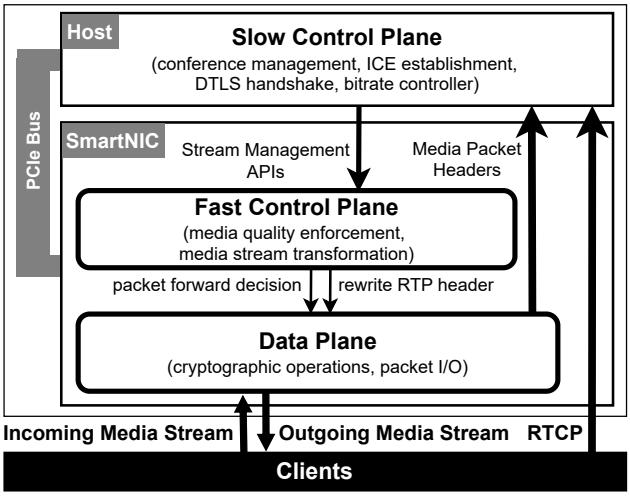


Figure 6: eXpressSFU Design Overview.

given the varying latencies of different operations, it is crucial to allocate more cores to high-latency tasks and avoid locking or synchronization mechanisms. This prevents stalls in the packet polling loop, which would otherwise lead to significant packet drops at the NIC buffer.

4.2 Slow and Fast-Control Planes Design

As discussed in §3.2, our goal is offloading data plane operations to SmartNICs to meet strict latency requirements while leveraging existing SFU software as much as possible. A key challenge lies in the bitrate controller's need to inspect each incoming media packet for rate estimation. However, our key observation is that the controller only requires metadata in the packet headers and packet size to perform estimation—it does not need access to the media payload. Leveraging this insight, we split packet handling into two parallel paths. In the slow path, the packet header is separated and forwarded to the bitrate controller running on the host for rate estimation, while the full media packet remains on the SmartNIC for high-performance processing on the fast path. Figure 6 illustrates the resulting *eXpressSFU* architecture.

Slow-control plane. Complex management tasks that tolerate higher latency are handled by the host CPU, taking advantage of its flexibility. The slow-control plane is responsible for tasks such as DTLS key negotiation, data channel establishment, conference management (e.g., participant join/leave), and most importantly, bitrate control. To integrate with our system, we make only minimal modifications to the existing bitrate controller and conference manager—primarily by exposing media stream metadata and forwarding quality decisions via our stream-management APIs—while leaving all other SFU modules completely unchanged.

Fast-control plane. Due to *eXpressSFU*'s transparent design, the slow-control plane selects the appropriate video quality for each receiver but does not directly participate in media packet transmission. Instead, enforcement of quality decisions is delegated to the fast-control plane, which runs on the SmartNIC and serves as an intermediary between the slow-control and data planes. It receives quality decisions from the bitrate controller in the slow-control plane and instructs the data plane to selectively duplicate packets accordingly, ensuring that quality policies are applied on a per-packet basis.

4.3 Offloading Data Plane to SmartNIC

Data plane. In an SFU, incoming media packets must be decrypted to extract forwarding metadata, such as temporal and spatial layer IDs. Based on receiver feedback, packets are then selectively cloned and encrypted in-place before being forwarded. As shown in §3, data plane operations contribute significantly to the processing overhead of software-based SFUs. However, these operations can be dramatically accelerated on SmartNICs (Table 1). To meet the latency demands of large-scale, high-resolution video conferencing, *eXpressSFU* delegates data plane operations to the SmartNIC. Since the data plane directly handles packet I/O, it must also support both Simulcast and Scalable Video Coding (SVC) to enable adaptive bitrate features.

SVC. Each media packet corresponds to a specific spatial and temporal layer. Given the target layer specified by the bitrate controller, the fast-control plane enforces quality by determining whether a packet should be delivered to a particular receiver. If a packet belongs to a layer higher than the assigned target, the fast-control plane can instruct the data plane to drop the packet for that receiver. Conversely, if the packet falls within the target layer, it is forwarded as expected. This mechanism ensures that each receiver receives only the video layers appropriate to their current bandwidth.

Simulcast. In simulcast, each incoming stream represents a different quality level (e.g., 360p, 720p, 1080p). The bitrate controller, running on the slow-control plane, selects the most suitable stream for each receiver based on their available bandwidth. This selection is passed to the fast-control plane, which enforces it. After the data plane clones the packets, the fast-control plane rewrites RTP headers of outgoing packets to match the selected stream mapping, ensuring proper quality delivery to each receiver.

4.4 Loosely Coupled Control-Data Planes

In conventional software-based SFUs, the control and data planes are tightly coupled, requiring every media packet to pass through control logic, which makes it difficult to meet the latency requirements of high-quality conferencing. *eXpressSFU* breaks this dependency by decoupling the latency-tolerant control plane from the latency-sensitive data plane. This enables the data plane to be hardware-accelerated and scaled independently on the SmartNIC, while the complex control logic remains on the host. To enable timely bitrate adaptation, *eXpressSFU* introduces a lightweight, asyn-

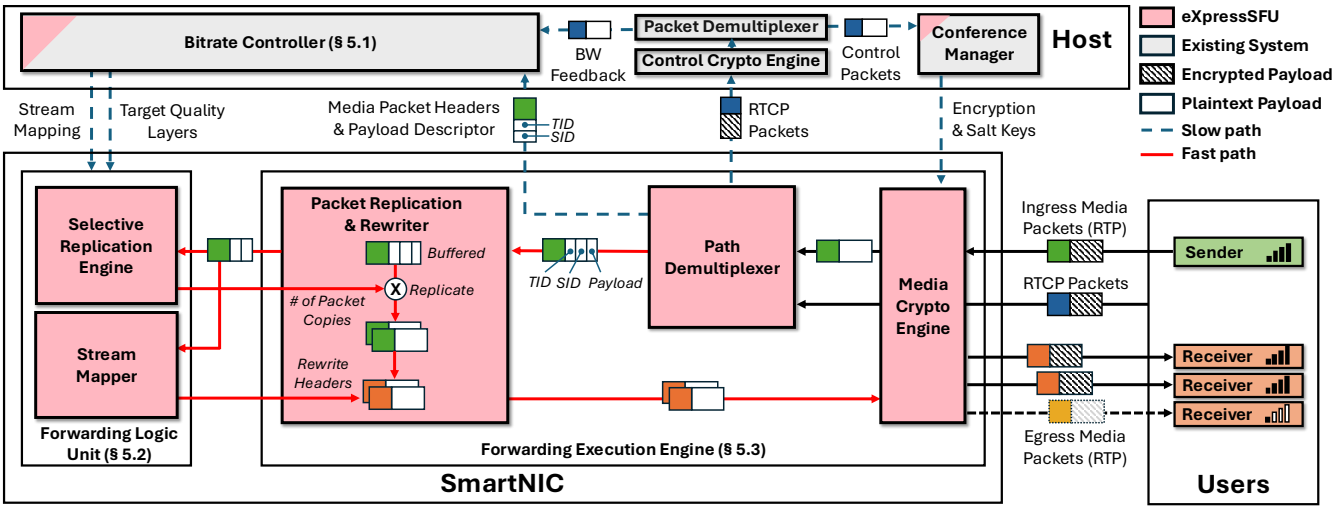


Figure 7: *eXpressSFU* System Overview.

chronous, closed-loop coordination mechanism. As shown in Figure 6, full media packets remain on the SmartNIC, while only packet headers and control traffic are sent to the slow-control plane. The bitrate controller and DTLS module process this information and update the fast-control plane, via stream-management APIs, with cryptographic keys, stream mappings, and target-layer directives. Since the slow and fast paths run in parallel, coordination is triggered only upon media-related events, such as changes in user bandwidth.

5 *eXpressSFU* System

This section introduces *eXpressSFU*, a system designed to scale large, high-quality video conferences by offloading latency-sensitive packet processing to SmartNICs while maintaining transparency. Figure 7 illustrates *eXpressSFU* system overview. Ingress media packets first reach the Forwarding Execution Engine (§5.3) on the SmartNIC, where the Crypto Engine removes hop-by-hop encryption. The packets then enter the Path Demultiplexer, which splits traffic into the slow path (control) and the fast path (data):

Slow path. The demultiplexer forwards two types of traffic to the host: (i) RTCP packets containing control information, such as per-receiver bandwidth feedback and DTLS handshakes, and (ii) each packet's RTP header along with its VP9 payload descriptor, which reveals the packet's temporal (TID) and spatial (SID) layers. The host-side Bitrate Controller (§5.1) estimates the sender's ingress rate and uses receiver bandwidth feedback to make quality decisions. These decisions—expressed as "stream-mapping" and "target-layers" directives—are sent back to the SmartNIC, where the Forwarding Logic Unit (§5.2) applies them to control the sending quality for each receiver at the packet level in egress streams.

Fast path. Complete media packets remain on SmartNIC. The Path Demultiplexer forwards each packet to the Packet Replication & Rewriter (PRR), which creates per-receiver replicas. Whether a receiver should receive a packet depends on the quality decision from the Bitrate Controller. The Selective Replication Engine (SRE) examines the packet's header and its TID/SID fields. Using the target-layer directives from the Bitrate Controller, the SRE determines exactly how many replicas should be generated to satisfy each receiver's quality requirement. The Stream Mapper then rewrites SSRCs, sequence numbers, timestamps, and other RTP header fields according to stream-mapping directives. Finally, the Crypto Engine reapplies hop-by-hop encryption, and the SmartNIC transmits the media packets directly to receivers, completing the data-plane pipeline without involving the host.

5.1 Bitrate Controller

To fully process media packets on the SmartNIC, the bitrate controller running on the host must provide essential stream-related information to the SmartNIC. However, these interactions should remain simple and easy to integrate across a wide range of platforms. To achieve this, we developed four minimal yet sufficient APIs to transmit stream and DTLS information to the SmartNIC, as summarized in Table 2.

To meet WebRTC's hop-by-hop encryption requirements, the `send_keys()` function transmits the encryption and salt keys for each media stream to the SmartNIC after DTLS negotiation between the SFU and each user. Because the SmartNIC has no concept of users or conferences, `send_mapping()` establishes the mapping between incoming (sender) and outgoing (receiver) streams, including their sequence number offset. This mapping is especially crucial for Simulcast, as it dictates which quality stream should be forwarded based on each receiver's bandwidth.

For SVC, the SmartNIC must know the target quality layers—specifically, SID for resolution and TID for frame rate—assigned to each outgoing stream. This is handled by

Table 2: *eXpressSFU* Host-to-SmartNIC media stream management APIs.

Method	Description
`send_keys(long ssrc, byte[] encKey, byte[] saltKey);`	`Sends encryption & salt keys for a stream to SmartNIC.`
`send_mapping(long rx_ssrc, long tx_ssrc, long delta_seqNo);`	`Sends a mapping of ingress and egress streams to SmartNIC.`
`send_layers(long tx_ssrc, int TID, int SID);`	`Sends the target SVC layers for an egress stream to SmartNIC.`
`expire_stream(long ssrc);`	`Sends the expiry message for a stream to SmartNIC.`

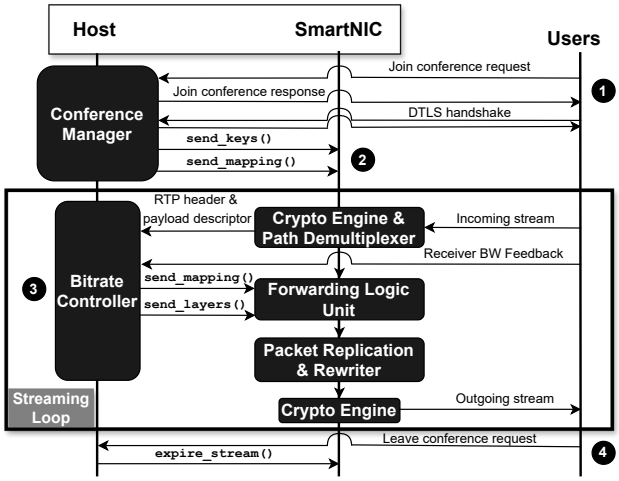


Figure 8: Conference managament APIs in *eXpressSFU*.

the `send_layers()` function. When a participant leaves, the `expire_stream()` function informs the SmartNIC of the associated stream ID (SSRC), allowing it to release resources and avoid memory overflow. Figure 8 illustrates how these functions operate within the conference system.

Referring to ❶, when a participant joins a conference, the conference manager first performs SDP negotiation [27] to exchange media and transport parameters. It then establishes media sessions using ICE protocol [50]. If a direct connection cannot be formed, the system falls back to STUN or TURN [37, 51]. A DTLS handshake [49] then follows, establishing shared keys for hop-by-hop encryption [3].

After the conference setup steps, as shown in ❷, the conference manager invokes the `send_keys()` and `send_mapping()` functions to transmit cryptographic keys and initial stream mappings to the SmartNIC.

In step ❸, the bitrate controller selects the appropriate quality level for each outgoing stream within the streaming loop. Incoming media packets are preprocessed and tail-trimmed on the SmartNIC, retaining only the RTP header and VP9 payload descriptor for the bitrate controller. The host CPU inspects these headers solely for bitrate estimation and then discards the packets, as all egress streams are generated independently and entirely on the SmartNIC. This results in negligible CPU load while improving privacy by keeping media content confined to the Data Link Layer, unlike the state-of-the-art solution where media content was exposed at the Application Layer.

SVC. The bitrate controller calls `send_layers()` to configure the SmartNIC's SRE, ensuring that only packets matching the target layers are replicated and sent. Target layers are adjusted dynamically based on incoming stream bitrates and receiver bandwidth feedback. `send_layers()` is invoked only when a quality decision changes. The bitrate controller also uses `send_mapping()` to update the SmartNIC's Stream Mapper, which rewrites RTP headers for outgoing packets.

Simulcast. Each quality level is delivered as a separate RTP stream, eliminating the need for intra-stream layers; thus, `send_layers()` is never used. Instead, the bitrate controller invokes `send_mapping()` during quality switches, enabling the Stream Mapper to rewrite RTP headers on the fly.

Regardless of whether Simulcast or SVC is used, the SmartNIC continues forwarding media packets until new target quality settings via either "target-layers" or "stream-mapping." As shown in ❹, when a participant leaves, the bitrate controller invokes `expire_stream()` to free associated resources tied to the stream (e.g., DPDK crypto operations, key buffers). This also prevents the reuse of outdated cryptographic keys in case SSRCs are reassigned to new streams.

5.2 Forwarding Logic Unit

In WebRTC, receivers are unaware of Simulcast and SVC, so every quality switch made by *eXpressSFU* must be completely transparent. This means the receiver should experience a continuous stream with sequentially increasing sequence numbers and timestamps, regardless of whether a quality switch occurs. The Forwarding Logic Unit (FLU) handles packet-level quality switching and consists of two submodules: the SRE and the Stream Mapper. The Stream Mapper supports Simulcast by transforming RTP headers and remapping incoming media packets to outgoing streams according to bitrate controller instructions. Working in conjunction with Stream Mapper, the SRE enables SVC by enforcing the outgoing stream's bitrate according to target layer decision from bitrate controller.

Stream Mapper. Simulcast packet flow is visualized in Figure 9. It is relatively straightforward to support because incoming streams are encoded at distinct quality levels, each identified by a unique SSRC. During a quality change, the bitrate controller sends an updated stream mapping using `send_mapping()`. While the outgoing stream identifier `tx_ssrc` remains unchanged, the incoming stream identifier `rx_ssrc` is updated. To facilitate the switch, it waits for a keyframe from the new stream, then replaces the old mapping with the new one and continues forwarding packets without interruption. This ensures that the decoder at the receiver can correctly decode frames from the newly mapped stream.

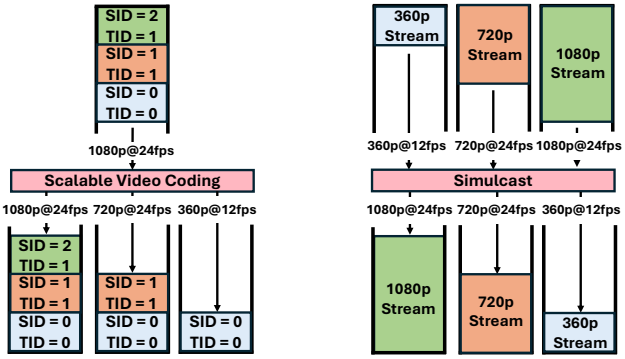


Figure 9: *eXpressSFU*'s packet forwarding. For SVC, it forwards only the layers each receiver can sustain. For Simulcast, it picks the best-fit stream for every receiver.

The RTP header rewriting process involves three steps. First, the latest egress sequence number is incremented by 1; if no history exists, `delta_seqNo` from `send_mapping()` is applied to map the ingress packet to the first egress packet. Second, the timestamp is updated—either copied from the most recent transmitted packet or incremented by Δ_{tstamp} if the packet starts a new frame. Here, Δ_{tstamp}, the inter-frame interval from the incoming stream, is applied across all associated outgoing streams to maintain consistent pacing. Finally, the sender's SSRC is replaced with the receiver's, correctly associating the packet with its outgoing stream

Selective Replication Engine. SVC packet flow is also visualized in Figure 9. Supporting SVC is more complex because multiple quality layers are encoded within a single stream, identified by a single SSRC. The first challenge is determining the optimal packet for scaling up or down between spatial and temporal layers. The second challenge is deciding whether a packet should be forwarded to each receiver to achieve the desired resolution/framerate specified by the bitrate controller.

Temporal Layer Switching. The process for switching temporal layers is illustrated in Algorithm 1. While downscaling a temporal layer can occur at any time, upscaling requires waiting for the `U bit` in the VP9 payload descriptor (line 8). The `U bit` serves as temporal switch-up point, indicating whether the current layer frame depends on previous layer frames of the same temporal layer [55]. The temporal layer can be upscaled one or two layer(s) at a time.

Spatial Layer Switching. Switching spatial layers is illustrated in Algorithm 2. Downscaling a spatial layer is possible at the start of a new frame, as indicated by the `B bit` (line 6). Upscaling, however, requires waiting for the `P bit` to be set to 0 (line 10), signaling that the frame does not rely on inter-picture prediction [55]. This enables an upscale from `(SID-1)` to `(SID)`, and only one spatial layer can be upscaled at a time (line 10).

Packet forwarding decision. The SRE determines how many copies of a packet are needed to serve the intended receivers. Based on TID, receivers requiring lower frame rates do not receive packets from higher temporal layers. Similarly, based on SID, receivers targeting lower resolutions are not assigned packets from higher spatial layers, and may also skip lower layers that do not contribute as references. Once this filtering is complete, the SRE instructs the PRR to generate the exact number of packet copies, avoiding unnecessary duplication.

Algorithm 1: Temporal Layer Switching in SVC.

```
   Input  : cur_TID ; // Temporal layer in use i.e. 15fps
1  tar_TID ; // Desired temporal layer i.e. 30fps
2  p ; // Incoming video packet with fields: p.TID, p.U_bit
   Output : Decision whether to switch to new temporal layer
3  Function TemporalSwitch(cur_TID,tar_TID, p):
4      if tar_TID ≠ cur_TID then
           // Scale down the frame rate
5          if tar_TID < cur_TID then
6              cur_TID ← tar_TID;
           // Scale up the frame rate
7          else
               // Wait for Switching up point
8              if p.U_bit and cur_TID < p.TID ≤ tar_TID then
9                  cur_TID ← p.TID;
10             else
                   // Cannot switch TID at this packet
11     else
           // No TID change
```

Final Frame Signaling. The `M bit` in the RTP header indicates the last packet of a frame. Since lower spatial layers contain fewer packets, *eXpressSFU* must rewrite the `M bit` in the RTP header to appropriately signal the end of a frame.

5.3 Forwarding Execution Engine

The Forwarding Execution Engine (FEE) is responsible for direct media packet manipulation and is where dataplane acceleration occurs. It consists of three submodules, namely the Media Crypto Engine, Path Demultiplexer and PRR.

Media Crypto Engine. Upon packet arrival, media packets (RTP) are decrypted and authenticated using encryption and salt keys provided by the conference manager. Control packets (RTCP), however, bypass the SmartNIC and are decrypted on the host. Before media packets are sent out, the Media Crypto Engine re-applies hop-by-hop encryption and appends an authentication tag at the end of the media payload to ensure packet integrity and prevent tampering. To accelerate cryptographic operations, the system leverages DPDK's batch-processing capabilities. Each SmartNIC core maintains its own dedicated DPDK crypto session (`rte_crypto_sym_xform`) [21] to eliminate inter-core locking and ensure parallel, contention-free processing.

Path Demultiplexer. After decryption, the RTP header is parsed, and the payload-type field is used to demultiplex media from non-media traffic. Non-media packets are forwarded immediately to the host for handling. Media packet headers and the VP9 payload descriptors are also sent to the host, enabling ingress stream bitrate estimation. Meanwhile, complete media packets are passed to the PRR for fast-path processing.

Algorithm 2: Spatial Layer Switching in SVC.

```
Input  : cur_SID ; // Spatial layer in use i.e. 720p
tar_SID ; // Desired spatial layer i.e. 1080p
p ; // Incoming video packet with fields: p.SID, p.B_bit, p.P_bit
Output : Decision whether to switch to new spatial layer
Function SpatialSwitch(cur_SID, tar_SID, p):
    if tar_SID ≠ cur_SID then
        // Scale down the resolution
        if tar_SID < cur_SID then
            // Wait for new frame
            if p.B_bit then
                cur_SID ← tar_SID;
            else
                // Cannot switch SID at this packet
        // Scale up the resolution
        else
            // Up-switching from directly lower SID
            // And no inter-picture prediction in use
            if cur_SID + 1 == p.SID and !p.P_bit then
                cur_SID ← p.SID;
            else
                // Cannot switch SID at this packet
    else
        // No SID change
```

Packet Replication & Rewriter. The RTP header and VP9 payload descriptor are first processed by SRE, which determines how many copies of a packet are needed to serve all intended receivers. The PRR module then creates exactly the required number of duplicates, avoiding unnecessary egress traffic. Each replicated packet header is modified by the Stream Mapper, which rewrites the RTP header to map the incoming into the appropriate outgoing packet. In addition, the source and destination IPs/ports are swapped before handing egress packets to the Media Crypto Engine for re-encryption.

Latency aware master-slave processing. The master core receives incoming packets and assigns each stream to a worker core in a round-robin fashion. Each worker core is responsible for performing data plane operations for the assigned stream and transmitting egress traffic directly from its own queue, avoiding a centralized bottleneck at the master core under high load. When there are many receivers, the decryption workload is shifted to the master core, which also duplicates the incoming media stream. Each worker core then receives the same set of decrypted packets and is responsible for forwarding them to a disjoint subset of receivers. This approach helps reduce the accumulated processing latency to meet the tight latency requirements of large, high-quality conferences.

6 IMPLEMENTATION

6.1 Host Stack

The *eXpressSFU* host stack is implemented by modifying the Jitsi Videobridge [12] to incorporate the stream management functions outlined in Table 2. Notably, enabling SmartNIC offload requires only minimal modifications to the original SFU codebase (254 out of 69,409 lines of code). The other components of the Jitsi Meet system remain unchanged, including a XMPP server [9] and a React web application [10]. Jitsi Meet was chosen as the baseline system because it supports both Simulcast and SVC with VP9, Google's latest video codec. Additionally, Jitsi Meet has a large user base, which enhances the adoption potential of *eXpressSFU*.

The stream management functions communicate with the SmartNIC via the PCIe bus using NVIDIA's DOCA Comm Channel [42]. This communication channel is secure, as only users with root privileges can access it. While control messages are sent over the PCIe lane for security and low-latency reasons, other types of packets—such as tail-trimmed media packets or RTCP control packets—are transmitted and received through the host-facing interface of the SmartNIC.

6.2 SmartNIC Stack

The SmartNIC stack is implemented on the BlueField-2 [39] platform as a DPDK [47] application running on eight ARM cores. Open vSwitch [48] and Scalable Functions [43] are configured so that incoming packets are first directed to the DPDK program for processing. To enable hop-by-hop encryption, the Cryptodev Scheduler Poll Mode Driver (PMD) library [30] from DPDK is employed. It is worth noting that no usable hardware crypto acceleration is available to DPDK for AES-GCM-128/256, which is the suite used by Jitsi Meet for packet encryption, decryption, and authentication.

7 Evaluation

In this section, we evaluate the performance of *eXpressSFU* against a software-based SFU solution (e.g., Jitsi Meet). Our evaluation is guided by the following questions. **Q1:** What is the maximum number of additional concurrent conferences that *eXpressSFU* can support compared to software baseline in typical conference sizes? Refer to §7.2. **Q2:** By what factor can media packet processing be accelerated on SmartNIC relative to host CPU? Refer to §7.3. **Q3:** How effectively can *eXpressSFU* adapt the sending rate in response to variations in the receiver's available bandwidth? Refer to §7.4. **Q4:** What is the expected power saving for service providers who adopt *eXpressSFU*? Refer to §7.5.

7.1 Experiment Setup

Our experiments are performed in a cluster consisting of 100 bare metal machines as load-testing clients and a workstation as the SFU. All machines are connected back-to-back via a Top-of-Rack L2 switch, running at 100 Gbps. The client machine specification is listed in Table 3a, while the *eXpressSFU* machine specification is listed in Table 3b.

Both *eXpressSFU* and Jitsi Meet are deployed on identical infrastructure to ensure a fair comparison. When running Jitsi Meet, the SmartNIC is configured so that packets bypass the

Table 3: Evaluation Setup.

(a) Client Machine Specification.

Component	Specification
CPU	Intel E5-2680 v2 @ 2.8GHz (20 cores)
Memory	64GB DDR3
NIC	10G Mellanox ConnectX-3

(b) *eXpressSFU* Machine Specification.

Component	Specification
CPU	Intel i9-14900K (24 cores total)
Memory	64GB DDR5
NIC	100G Nvidia BlueField-2

(c) Evaluation Scenarios.

Conference size	# of Users	Video quality
Small	15	720p/1080p@24fps
Medium	50	720p/1080p@24fps
Large	100	720p/1080p@24fps

ARM core, being handled by the host CPU. An Ansible-based orchestration framework also automates the deployment, execution, and log collection processes across all client machines. The evaluation scenarios are presented in Table 3c, and each conference has a single sender while the others are receivers. This conference configuration helps to detect quality degradation reliably by collecting resolution and inter-frame delay reports from each receiver via `getStats()` method [19].

7.2 Impact of User Scale & Media Quality

Figure 10 presents the number of concurrent conferences supported in large-scale settings before video quality degradation occurs when varying the number of Intel or ARM cores devoted to packet processing. This scenario presents one of the most challenging scenarios for scaling SFU systems, particularly in the context of online webinars. The complexity is notably exacerbated at 1080p@24fps, where the latency budget is constrained due to the elevated media quality and substantial number of concurrent users (refer to §7.3). At this resolution, Jitsi Meet struggles to scale effectively even when additional CPU cores are allocated. In contrast, *eXpressSFU* demonstrates linear scalability as more ARM cores are provisioned. At 720p@24fps, *eXpressSFU* still outperforms Jitsi Meet, though the gap in scalability is narrower since the latency budget is more flexible. Similar trends can be observed in medium-sized conferences, which are typically used in online classes, as shown in Figure 11.

Small-sized conferences—by far the most common—are illustrated in Figure 12. In high resolution setting, *eXpressSFU* scales particularly well, thanks to its low-latency decryption performance, as detailed in Table 1. Decryption is the most computationally expensive operation in the data plane and becomes a significant CPU bottleneck when handling a large number of ingress streams. For small-scale low-quality conferences, *eXpressSFU* and Jitsi Meet scale similarly well.

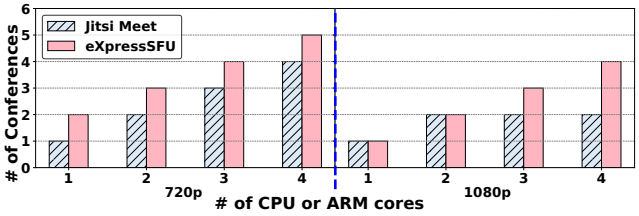


Figure 10: Large conferences (100 users).

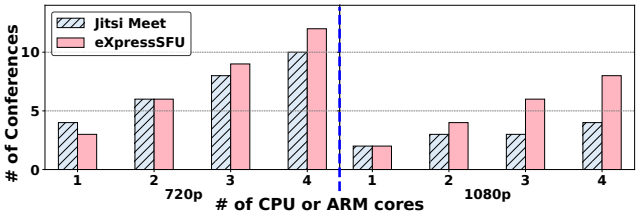


Figure 11: Medium conferences (50 users).

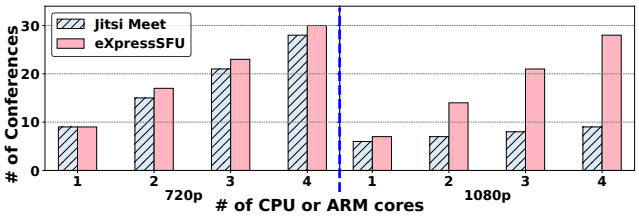


Figure 12: Small conferences (15 users).

The Bluefield-2 contains only 8 cores running at 2Ghz while the host CPU contains 24 cores running at 2.5 - 6.0 GHz. *eXpressSFU* achieves better performance because of our design focused on partitioning latency-sensitive work to the Bluefield-2, coupled with DPDK and low-latency network interfaces. Since control packets only account for 16% of total traffic, *eXpressSFU* should be able to scale to multiple Bluefield-2 SmartNICs or to the faster Bluefield-3 card, which doubles the ARM core count and includes other accelerations.

7.3 Processing Latency Improvements

Figure 13 shows the measured latency of each component in the media packet processing pipeline[1] on the SmartNIC using the same methodology in §3. Even with 150 users, the latency of *eXpressSFU* is 1/8th that of Jitsi, increasing the number of users that can be served before the packet latency causes frame drops. The absence of control plane overhead stems from the intelligent decoupling design of *eXpressSFU*, which isolates slow-control logic from the latency-sensitive fast-control and data planes. Packet I/O latency is minimal due to SmartNIC offloading; the ARM core merely issues asynchronous DMA operations, while the NIC hardware independently manages packet transceiver.

[1]The per-packet time to clone a packet increases with the number of users; we believe this is because all packets are cloned by the single core processing that packet before further processing, rather than making better use of the small caches. Nonetheless, the measured performance is improved.

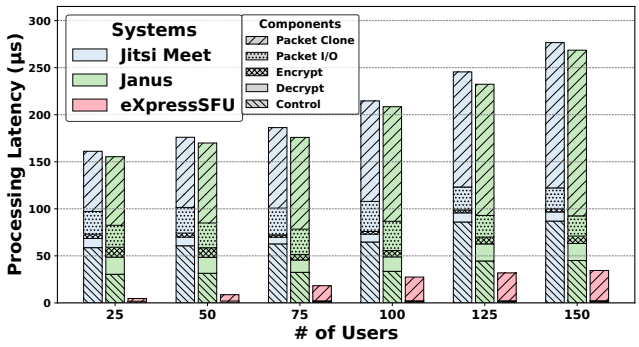


Figure 13: Jitsi Meet, Janus and *eXpressSFU* latency comparison. *eXpressSFU* reduces latency by 8× at 150 users.

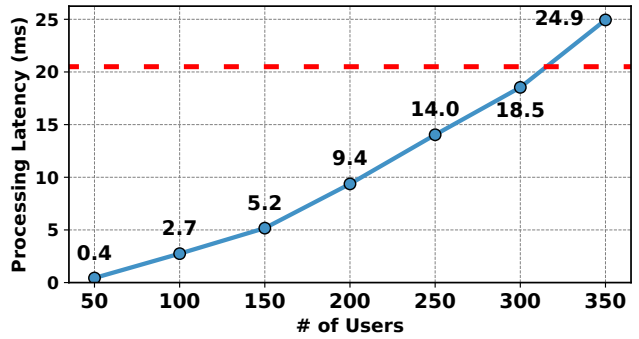


Figure 14: Total processing latency for all users per media packet of *eXpressSFU* when *using a single ARM core*. When the processing latency exceeds 20.5 ms, frame drop occurs on the receiver side.

Thanks to its extremely low-latency processing, *eXpressSFU* can support a substantially larger number of users even with a single ARM core, as shown in Figure 14. Quality degradation becomes noticeable beyond 300 concurrent users, where the total packet processing latency exceeds 20.5 ms—consistent with the latency bound discussed in §3. The corresponding impact on throughput is illustrated in Figure 15, which compares the goodput achieved under increasing load. While *eXpressSFU* can maintain optimal bitrate, Jitsi Meet stops forwarding video entirely due to significant frame drop report from users.

7.4 Bitrate Adaptation Effectiveness

eXpressSFU doesn't change the bitrate adaptation in the SVC controller; receiver's feedback is sent to the bitrate controller on the host which then adjusts the stream selection in the SmartNIC as described in §5.1. To demonstrate that *eXpressSFU* has negligible impact on bitrate control, we limited the receiver's downlink bandwidth to various rates (e.g., 2 Mbps, 1 Mbps). As shown in Figure 16, *eXpressSFU*'s sending rate closely follows the receiver's available bandwidth: delivering 1080p when bandwidth is unrestricted, downgrading to 720p and then 360p as bandwidth is reduced, and

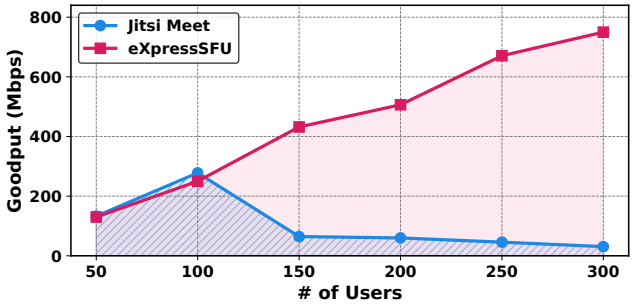


Figure 15: Goodput comparison. Using a single core, *eXpressSFU* can support up to 300 users at 1080p@24fps without quality degradation, achieving 26× better goodput.

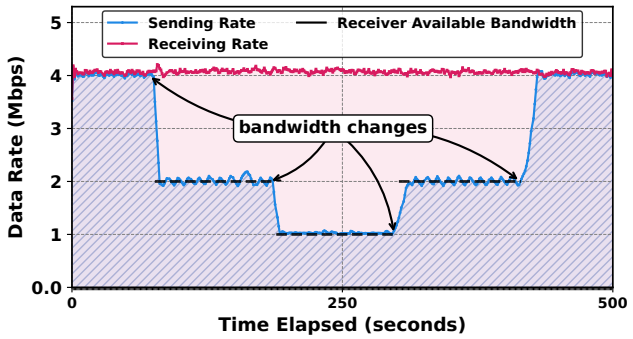


Figure 16: SVC bitrate adaptation measured on *eXpressSFU*. *eXpressSFU* can adapt the sending rate effectively according to the receiver's changing bandwidth.

then restoring higher quality when bandwidth recovers. This dynamic adaptation preserves user QoE. Before each quality switch, the host-side control plane issues a key-frame request (Full Intra Request) to the sender—an event that could be easily intercepted at the SmartNIC for even tighter Host–SmartNIC coordination.

7.5 Power Consumption

To evaluate the operational efficiency, we measured and compared the total package power consumption between *eXpressSFU* and Jitsi Meet using a wall-socket power meter. In our setup, Jitsi runs entirely on the host CPU with BlueField-2 as a NIC [2] [41]. On the other hand, *eXpressSFU* leverages both host CPU and ARM cores with BlueField-2 operating in DPU mode [40]. At idling state, the server draws 113 W in both cases; Figure 17 reports the additional power draw attributable to computation. Jitsi's power consumption increases sharply with the number of users, reaching over 55 W at 250 users. In contrast, *eXpressSFU* exhibits a much flatter growth curve, remaining under 22 W. This efficiency arises

[2] In NIC mode, the BlueField-2 functions as ConnectX-6 adapter, with its ARM cores bypassed. Although Jitsi itself does not require the ARM cores, a dedicated NIC is still necessary for packet transmission and reception, since the on-board interface alone cannot provide sufficient bandwidth.

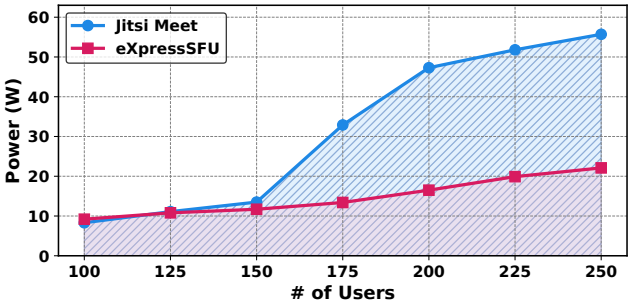


Figure 17: Jitsi and *eXpressSFU* computational power comparison. *eXpressSFU* cuts power usage by up to 60%.

from offloading latency-critical dataplane functions to the SmartNIC, which leverages packet batch acceleration running on energy-efficient ARM cores. As a result, *eXpressSFU* reduces operational costs and thermal output while sustaining superior serving capacity.

8 Discussion

Limitation of the current SmartNIC. Cryptographic operations are a significant source of overhead in the SFU. While the Nvidia BlueField-2 SmartNIC supports hardware crypto acceleration at line rate, it is limited to IPsec, where encryption keys are negotiated end-to-end between the NICs—outside the control of user-space applications such as those built with DPDK. In addition, the cryptographic suites commonly used in video conferencing—AES-GCM-128 and AES-GCM-256—are not supported by the NVIDIA MLX5 Crypto Driver on the BlueField-2 platform [20].

Opportunities with the new SmartNIC platform. NVIDIA BlueField-3 addresses this constraint by introducing a second crypto hardware module accessible to user-space applications. While this module offers high throughput for large file encryption and decryption, its API design is ill-suited for SFU (see Appendix §B). Nonetheless, BlueField-3 significantly improves overall compute capacity by doubling the number of ARM cores relative to BlueField-2. Furthermore, since control plane running on host CPU remains lightweight, deploying multiple SmartNICs within a single server could further enhance scalability.

Latency-Aware Hybrid Scheduling. An alternative design choice is to dedicate a few host CPU cores to the control plane while allocating the majority to a DPDK-based data plane as in the proposed *eXpressSFU* architecture. While this configuration works effectively for small (15–50 users) and low-quality (360p, 720p) conferences, it faces scalability limits in large (100+ users) and high-quality (1080p) settings due to the inherent PCIe round-trip latency between the host CPU and NIC hardware [7,23]. Thus, conferences can flexibly be scheduled on either ARM or CPU cores depending on their specific latency budget. The detailed design of such a load-balancing mechanism is beyond the scope of this paper and will be explored in future work.

9 Related Work

With the advent of video conferencing and the limitations of MCU-based real-time transmission, the SFU architecture was developed [26]. Unlike MCU, which mixes media streams, an SFU forwards streams from one participant to others without processing the content, reducing server load and shifting computation to clients. The evolution of SFU has been closely aligned with the success and widespread adoption of WebRTC. Numerous frameworks have been developed to support SFU architecture, including Jitsi, one of the pioneering frameworks for Videobridge supporting multi-user communication, MicroTalk SFU [17], Kurento, which supports both MCU and SFU [33], as well as Licode [36] and Janus [31]. In recent years, SFU has been adapted for emerging applications like metaverse-based virtual reality conferencing [28]. Despite these advantages, SFUs face scalability challenges, particularly under peak load and during packet replication for multiple users [45]. Recent work has addressed this by using P4 network switches to offload SFU operations, enabling low-cost and efficient packet duplication and multicast [54].

10 Conclusion

In this work, we present *eXpressSFU*, a re-architecting of the conventional SFU system that significantly enhances scalability by intelligently decoupling the media packet processing pipeline from the slow control path. This design enables fast-path media processing directly on the SmartNIC while preserving the existing bitrate controller on the host, thereby facilitating seamless integration and adoption. *eXpressSFU* also supports both Simulcast and Scalable Video Coding (SVC) in a transparent manner, ensuring compatibility across a wide range of devices with varying capabilities.

Experimental results show that *eXpressSFU* can accommodate up to 3× more concurrent users in the most challenging scenario: large-scale video conferences at 1080p@24fps—while reducing computational power consumption by up to 60% compared to the baseline.

Acknowledgments

We sincerely thank our shepherd, Daehyeok Kim, and the anonymous reviewers for their valuable feedback. We also thank Evan Ram for his contributions to the development of *eXpressSFU* and CJ Herman for his assistance with the testbed setup. This work was supported by the National Science Foundation under Grant No. 1908910, by the Institute of Information & Communications Technology Planning & Evaluation (IITP) under Grant Nos. RS-2024-00405128 and 2025-RS-2020-II201819 funded by the Korea Government (MSIT), and by the National Research Foundation of Korea (NRF) under Grant No. RS-2025-24535401. Sangtae Ha is the corresponding author.

References

[1] Dr. Bernard D. Aboba. Codec-Independent Selective Forwarding. https://datatracker.ietf.org/doc/draft-aboba-avtcore-sfu-rtp/00/. Accessed on January 4, 2025.

[2] Emmanuel André, Nicolas Le Breton, Augustin Lemesle, Ludovic Roux, and Alexandre Gouaillard. Comparative study of webrtc open source sfus for video conferencing. In *2018 Principles, Systems and Applications of IP Telecommunications (IPTComm)*, pages 1–8. IEEE, 2018.

[3] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman. The Secure Real-time Transport Protocol (SRTP). https://datatracker.ietf.org/doc/html/rfc3711, 2004.

[4] Edan Brooke. Setting up a self-hosted jitsi meet instance with multiple video bridges. https://shorturl.at/46LUN, 2023. Accessed: 2025-03-31.

[5] Ana Cárdenas, Mario García-Molina, Salvador Sales, and José Capmany. A new model of bandwidth growth estimation based on the gompertz curve: Application to optical access networks. *Journal of Lightwave Technology*, 22(11):2460, 2004.

[6] Gaetano Carlucci, Luca De Cicco, and Saverio Mascolo. Controlling queuing delays for real-time communication: the interplay of e2e and aqm algorithms. *SIGCOMM Comput. Commun. Rev.*, 46(3), July 2018.

[7] Xuzheng Chen, Jie Zhang, Ting Fu, Yifan Shen, Shu Ma, Kun Qian, Lingjun Zhu, Chao Shi, Yin Zhang, Ming Liu, et al. Demystifying datapath accelerator enhanced off-path smartnic. In *2024 IEEE 32nd International Conference on Network Protocols (ICNP)*, pages 1–12. IEEE, 2024.

[8] Co, Industry Research. Video conferencing market: Future demand and top key players analysis: 2031, Nov 2023. Accessed on February 2, 2024.

[9] Contributors. Jicofo GitHub Repository. https://github.com/jitsi/jicofo, 2024. Accessed on February 2, 2024.

[10] Contributors. Jitsi GitHub Repository. https://github.com/jitsi, 2024. Accessed on February 2, 2024.

[11] Contributors. Jitsi Meet GitHub Repository. https://github.com/jitsi/jitsi-meet, 2024. Accessed on February 2, 2024.

[12] Contributors. Jitsi Videobridge GitHub Repository. https://github.com/jitsi/jitsi-videobridge, 2024. Accessed on February 2, 2024.

[13] Contributors. Mediasoup GitHub Repository. https://github.com/versatica/mediasoup, 2024. Accessed on February 2, 2024.

[14] David Curry. Zoom revenue and usage statistics (2025). https://www.businessofapps.com/data/zoom-statistics/, 2025. Accessed: 2025-04-09.

[15] Hans L. Cycon, Thomas C. Schmidt, Matthias Wahlisch, Detlev Marpe, and Martin Winken. A temporally scalable video codec and its applications to a video conferencing system with dynamic network adaption for mobiles. *IEEE Transactions on Consumer Electronics*, 57(3):1408–1415, 2011.

[16] Mallesham Dasari, Edward Lu, Michael W Farb, Nuno Pereira, Ivan Liang, and Anthony Rowe. Scaling vr video conferencing. In *2023 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*, pages 648–657. IEEE, 2023.

[17] MiroTalk Developers. MiroTalk SFU: Secure, Real-Time Video Conferencing. https://sfu.mirotalk.com/, 2025. Accessed: 2025-04-10.

[18] Tabin Dharanikota, Dennis Edens, Rajesh Abbi, and Sudheer Dharanikota. Forecasting bandwidth utilization growth. *DTS whitepaper, Dec*, 2018.

[19] MDN Web Docs. RTCPeerConnection.getStats() method. https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/getStats, 2024. Accessed: 2025-10-21.

[20] DPDK. NVIDIA MLX5 Crypto Driver. https://doc.dpdk.org/guides/cryptodevs/mlx5.html. Accessed: 2025-04-25.

[21] DPDK. Struct Reference. https://doc.dpdk.org/api/structrte__crypto__sym__xform.html, 2025. Accessed: 2025-10-21.

[22] Alexandros Eleftheriadis, M Reha Civanlar, and Ofer Shapiro. Multipoint videoconferencing with scalable video coding. *Journal of Zhejiang University-Science A*, 7:696–705, 2006.

[23] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. Azure accelerated networking:{SmartNICs} in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 51–66, 2018.

[24] National Center for Educational Statistics. How many students take distance learning courses at the postsecondary level? https://nces.ed.gov/fastfacts/display.asp?id=80&utm_source=chatgpt.com. Accessed on January 4, 2025.

[25] Google. VP9. https://developers.google.com/media/vp9. Accessed on January 4, 2025.

[26] Boris Grozev, Lyubomir Marinov, Varun Singh, and Emil Ivov. Last n: relevance-based selectivity for forwarding video in multimedia conferences. In *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '15, page 19–24, New York, NY, USA, 2015. Association for Computing Machinery.

[27] M. Handley, V. Jacobson, and C. Perkins. SDP: Session Description Protocol. https://datatracker.ietf.org/doc/html/rfc4566, 2006.

[28] Yong-Hao Hu, Kenichiro Ito, and Ayumi Igarashi. Improving real-time communication for educational metaverse by alternative webrtc sfu and delegating transmission of avatar transform. In *2023 International Conference on Consumer Electronics-Taiwan (ICCE-Taiwan)*, pages 201–202. IEEE, 2023.

[29] SNS Insider. Video Conferencing Market Set to Surge: Aiming for USD 25 Billion by 2032 Driven by Remote Work and Technological Advancements. https://tinyurl.com/26dxdexr. Accessed on January 4, 2025.

[30] Intel. Cryptodev Scheduler Poll Mode Driver Library. https://doc.dpdk.org/guides/cryptodevs/scheduler.html. Accessed on January 4, 2025.

[31] Janus. Janus: the general purpose webrtc server. https://github.com/meetecho/janus-gateway, n.d. Accessed: 2025-01-14.

[32] Stewart Jones, David Johnstone, and Roy Wilson. An empirical evaluation of the performance of binary classifiers in the prediction of credit ratings changes. *Journal of Banking Finance*, 56:72–85, 2015.

[33] Kurento. Kurento Media Server (KMS). https://doc-kurento.readthedocs.io/en/latest/index.html, n.d. Accessed: 2025-01-14.

[34] Jesús Leganés-Combarro. WebRTC Horizontal Scaling. https://piranna.github.io/2021/09/26/WebRTC-horizontal-scaling/, September 2021. Accessed: 2025-03-31.

[35] Xianshang Lin, Yunfei Ma, Junshao Zhang, Yao Cui, Jing Li, Shi Bai, Ziyue Zhang, Dennis Cai, Hongqiang Harry Liu, and Ming Zhang. Gso-simulcast: global stream orchestration in simulcast video conferencing systems. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 826–839, 2022.

[36] Lynckia. Licode: Open source webrtc communications platform. https://lynckia.com/licode/, n.d. Accessed: 2025-01-14.

[37] R. Mahy, P. Matthews, and J. Rosenberg. Traversal Using Relays around NAT (TURN): Relay Extensions to STUN. https://datatracker.ietf.org/doc/html/rfc5766, 2010.

[38] NVIDIA. DOCA AES-GCM. https://docs.nvidia.com/doca/sdk/doca+aes-gcm/index.html. Accessed: 2025-04-25.

[39] NVIDIA. NVIDIA Bluefield-2 SmartNIC. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/datasheet-nvidia-bluefield-2-dpu.pdf. Accessed on January 4, 2025.

[40] NVIDIA. NVIDIA Bluefield DPU mode. https://docs.nvidia.com/doca/archive/2-9-0-cx8/bluefield+modes+of+operation/index.html#src-3453016816_id-.BlueFieldModesofOperationv2.9.1-SmartNICmode. Accessed on Auguest 11, 2025.

[41] NVIDIA. NVIDIA Bluefield NIC mode. https://docs.nvidia.com/doca/archive/2-9-0-cx8/bluefield+modes+of+operation/index.html#src-3453016816_id-.BlueFieldModesofOperationv2.9.1-NICMode. Accessed on January 4, 2025.

[42] NVIDIA. NVIDIA DOCA Comm Channel. https://docs.nvidia.com/doca/archive/doca-v1.5.0/pdf/comm-channel-samples.pdf. Accessed on January 4, 2025.

[43] NVIDIA. Scalable Function. https://docs.nvidia.com/doca/archive/doca-v2.2.0/scalable-functions/index.html. Accessed on January 4, 2025.

[44] Stefano Petrangeli, Dries Pauwels, Jeroen van der Hooft, Tim Wauters, Filip De Turck, and Jürgen Slowack. Improving quality and scalability of webrtc video collaboration applications. In *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys '18, page 533–536, New York, NY, USA, 2018. Association for Computing Machinery.

[45] George Politis, Boris Grozev, Paweł Domas, Emil Ivov, and Thomas Noël. Experimental evaluation of dynamic switching between one-on-one and group video calling. In *2018 Principles, Systems and Applications of IP Telecommunications (IPTComm)*, pages 1–7, 2018.

[46] Miroslav Ponec, Sudipta Sengupta, Minghua Chen, Jin Li, and Philip A. Chou. Multi-rate peer-to-peer video conferencing: A distributed approach using scalable coding. In *2009 IEEE International Conference on Multimedia and Expo*, pages 1406–1413, 2009.

[47] Linux Foundation Collaborative Projects. Data Plane Development Kit. https://www.dpdk.org/. Accessed on January 4, 2025.

[48] Linux Foundation Collaborative Projects. Open vSwitch. https://www.openvswitch.org/. Accessed on January 4, 2025.

[49] E. Rescorla and N. Modadugu. Datagram Transport Layer Security Version 1.2. https://datatracker.ietf.org/doc/html/rfc6347, 2012.

[50] J. Rosenberg. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. https://datatracker.ietf.org/doc/html/rfc5245, 2010.

[51] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session Traversal Utilities for NAT (STUN). https://datatracker.ietf.org/doc/html/rfc5389, 2008.

[52] Janto Skowronek, Alexander Raake, Gunilla H Berndtsson, Olli S Rummukainen, Paolino Usai, Simon NB Gunkel, Mathias Johanson, Emanuël AP Habets, Ludovic Malfait, David Lindero, et al. Quality of experience in telemeetings and videoconferencing: a comprehensive survey. *IEEE Access*, 10:63885–63931, 2022.

[53] Statista. Zoom daily meeting participants worldwide 2020. https://shorturl.at/YdwU3, 2021. Accessed: 2025-04-25.

[54] Pavlos Tsikrikas and George Xylomenos. A selective forwarding unit implementation in p4. In *2024 IEEE Conference on Standards for Communications and Networking (CSCN)*, pages 181–186, 2024.

[55] Justin Uberti, Stefan Holmer, Magnus Flodman, Danny Hong, and Jonathan Lennox. RTP Payload Format for VP9 Video. https://datatracker.ietf.org/doc/draft-ietf-payload-vp9/10/.

[56] www.statista.com. Use of collaboration tools in the united states 2023, by remote work policy. https://shorturl.at/ylyd4. Accessed on January 4, 2025.

Appendices

A PCIe Overhead on Frame Timeliness

To examine how PCIe latency affects real-time video delivery, we conducted a micro-experiment emulating SFU packet fan-out behavior. Video traffic exhibits a periodic burst pattern, where packets arrive at the SFU every video capture interval—approximately every 41 ms for a 24 fps stream. In WebRTC, receivers maintain only a minimal jitter buffer (typically a few tens of milliseconds) to keep end-to-end latency low. Thus, each receiver must obtain a complete frame within the same 41ms interval to meet the rendering deadline. To model this, we implemented a simple UDP echo client–server: the client sends a burst of 64 packets representing 64 concurrent video streams, and the server replicates each packet 100 times (6,400 total) to simulate multi-receiver fan-out.

The experiment was conducted with the client and server running on two separate machines connected through a Top-of-Rack L2 switch to minimize network latency. The server process was executed either on a single host CPU core or on a single SmartNIC ARM core. A 20 ms end-to-end deadline was imposed, assuming a video frame is made up from two packets. When running on the host CPU, only 8.4% of the 25,600 total requests met the frame deadline. In contrast, executing the identical workload on the SmartNIC's ARM achieved 57.1% on-time delivery. These results highlight that eliminating PCIe traversal between host and NIC significantly improves frame timeliness in real-time video workloads.

B Limitation of Cryptographic Engine on Bluefield-3 Platform

While hardware offloading on the BlueField-3 platform effectively relieves the ARM cores from computationally intensive cryptographic operations, it introduces non-trivial latency due to its shared-memory API design [38]. Each encryption or decryption request requires the ARM core to copy the ingress payload (DPDK mbuf) into a pre-allocated shared memory region (DOCA buffers) accessible to the cryptographic engine, issue a command to initiate processing, and poll or wait for completion before copying the result back. This multi-step data movement and synchronization overhead significantly increases end-to-end processing latency—often dominating the actual computation time of AES-GCM operations. As a result, the BlueField-3's cryptographic offload mechanism, while beneficial for throughput-oriented workloads, becomes ill-suited for latency-critical media pipelines such as real-time video conferencing, where even microsecond-scale delays can affect frame delivery and user experience.