NORTHWESTERN UNIVERSITY

Cross-Layer Network Optimization Targeting Endusers

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Sen Lin

EVANSTON, ILLINOIS

December 2025

Abstract

Modern Internet protocols, originally designed for modest and predictable traffic, struggle to accommodate the diverse and stringent demands of emerging applications. Despite significant advances in handling private traffic, deployable network optimizations targeting endusers remain constrained by the rigid Internet protocols and infrastructure.

This thesis investigates cross-layer network optimization opportunities for endusers by treating the Internet protocol stack as a coordinated system rather than isolated layers. Specifically, I explore how information can be propagated across application, transport, and network layers to collaboratively improve end-to-end performance without violating protocol compatibility or requiring intrusive deployment.

Guided by this cross-layer perspective, I identify three key entry points for enduser-oriented network optimization. First, I present CloudCookie, a provider-side mechanism that enables in-band information synchronization across public-facing data center traffic, facilitating flow packet scheduling and predictive load balancing. Second, I introduce a user-side prioritization-based optimization for realtime interactive streaming that leverages cross-directional dependency. Finally, I propose virtual multipath, a provider-user collaborative technique that repurposes VPN proxy nodes to emulate multipath transmission on single-interface devices by tricking the kernel's MPTCP stack. Together, these projects demonstrate a holistic approach to enduser-oriented, cross-layer network optimization that respects Internet constraints while unlocking new performance gains through judicious use of available yet underutilized information and resources.

Thesis Committee

Aleksandar Kuzmanovic Northwestern University	Committee Chair
Peter Dinda Northwestern University	Committee Member
Yan Chen Northwestern University	Committee Member
Tom Barbette UCLouvain	External Committee Member

Acknowledgment

When I first entered the PhD program, I often flipped straight to the Acknowledgment of other students' dissertations and wondered what I might write one day. Now that I am finally here, this has become the most difficult section to complete. The years behind me hold far more gratitude, struggle, and joy than a few pages can capture. Northwestern has been the place I stayed the longest, and these years have shaped me more deeply than any previous period of my life.

I am deeply grateful to my advisor, Aleksandar Kuzmanovic, for his guidance, encouragement, and trust throughout my PhD journey. His mentorship played a central role in shaping me as an independent researcher. I learned not only to solve problems but also to identify them, to see the broader landscape behind individual decisions, and to take ownership of my work. Despite the challenges, the journey was genuinely enjoyable. Aleksandar gave me remarkable freedom to explore directions I cared about, even when they were risky or uncertain, and he always offered thoughtful support. His enthusiasm for new ideas resonated with me, and the independence he encouraged taught me as much about responsibility as curiosity. I doubt I will ever again have this level of freedom to explore ideas and pursue my own directions in my future career.

I am sincerely thankful to my committee members: Peter Dinda, Yan Chen, and Tom Barbette. Peter's passion for computing systems has always inspired me and sparked my lasting interest in low-level networking systems. Yan guided me through my first publication during my PhD and encouraged me throughout the years. Tom inspired the first independent project I ever worked on and always provided generous feedback. I am grateful for the time and care they devoted to strengthening this dissertation and expanding my perspective.

My gratitude also goes to my collaborators and colleagues. I especially thank Yunming Xiao, my only senior labmate; Kaiyu Hou, who led the first paper we published in my doctoral program; Jianfeng Wang, who collaborated with me on my first first-author paper; and Jing Gu, Andre Chen, Kevin Zikai Chen, Vinod Yegneswaran, and Yibo Zhao, who supported me in countless discussions and moments that shaped this work.

I would like to thank Dolby Laboratories, where I completed a meaningful and memorable internship. I am grateful to my mentor Jason Cloud, whose curiosity, kindness, and sense of humor made my experience especially enjoyable. I also thank Jeffrey Riedmiller, Kadierdan Kaheman, and Jin Heo for their support and for creating such a positive environment.

To the friends I met at Northwestern, including Hanjie Xie, Suting Chen, Yanzhi Li, Caleb Wang, Ying Zhang, Peizhi Liu, Weili Wu, and Kailai Tang, thank you for your companionship, your encouragement, and your willingness to listen to my worries and complaints over the years. I am equally grateful to my friends outside Northwestern, including Shanshan Xiao, Rongxuan Xie, Siwei Mai, and Qifan Yang, for their patience, kindness, and constant support.

Most importantly, I thank my family. My mother Xiao-Lin Chen, my father Da-Jun Shan, and my sister Shan Shan have been the most steady and unconditional source of support throughout my life. Their love and belief in me gave me the courage to continue, even during the most difficult moments.

Finally, I want to thank this planet, Earth, for every moment of serendipity, every unexpected encounter, and every challenge that shaped this journey.

Love and peace.

Dedication

To my mother, Xiao-Lin Chen.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern network protocols, originally designed for modest best-effort data delivery in a more predictable Internet, are increasingly challenged by the diverse and stringent demands of emerging applications, ranging from dynamic web services to latency-sensitive realtime interactive streaming services. The Internet now carries diverse and intensive traffic in both directions, often with distinct requirements for reliability, bandwidth, and latency. However, as network protocols remain fragmented in stacked layers, the design space for deployable optimization systems targeting endusers is heavily constrained by the inertia of legacy Internet protocols. While fully controlled network environments, such as private data center networks, enjoy the flexibility to deploy novel mechanisms and custom protocols, optimizations on the public Internet must adhere to protocol compatibility and minimize the extent of changes required from involved parties.

Despite being logically decoupled, different layers of Internet protocols form an integrated and interdependent system, sharing implicit state and influencing one another's behavior. Unfortunately, existing optimization efforts often neglect this interdependence. Valuable information available at the application layer, such as flow-level metadata, content semantics, or multipath availability, is rarely passed down to inform

lower-layer decisions. Similarly, network and transport layer observations, including sending progress and buffer states, are typically not surfaced back to the application layer, resulting in blind spots and missed optimization opportunities.

The twin forces of technological advancement and their unintended side effects further complicate the design and deployment of enduser-oriented network optimizations. On the one hand, programmable dataplane technologies (*e.g.,* P4 [116, 77], data plane development kit (DPDK) [94]), emerging transports (*e.g.,* QUIC [79]), and widely available overlay networks (*e.g.,* virtual private network (VPN)) have opened new avenues for fine-grained traffic handling. On the other hand, some advancements have introduced side effects that render network behavior increasingly opaque and unpredictable. For instance, traffic encryption obscures flow visibility, and middlebox behaviors controlled by different entities tend to be inconsistent, making traffic handling unpredictable.

Moreover, we outline a set of design principles for deployable optimization systems targeting endusers. First, optimizations should repurpose mechanisms within Internet protocols to ensure compatibility and non-intrusiveness with respect to existing infrastructure. Second, optimizations are supposed to be independently executable by either providers or endusers without requiring mutual cooperation. Finally, when provider-user collaboration is needed, user-side deployment should be confined to userspace to facilitate easy and practical adoption.

1.1 Approach

In order to build network optimization systems targeting endusers with a cross-layer mindset, this thesis focuses on the problem space of ignorance and mistreatment in handling public-facing Internet traffic, and explores deployable design opportunities within the

constraints of the existing Internet ecosystem and in response to the demands of emerging applications.

Given the shared interests and implicit coordination potential across network layers, we examine which parties—service providers or endusers—are positioned to independently or cooperatively implement practical network optimizations. Specifically, this thesis explores three opportunities for enduser-oriented cross-layer optimization: a provider-side activation of data center advances for public-facing traffic; a user-side exploitation of cross-directional dependency in realtime interactive streaming; and a provider-user collaboration that expands multipath access by tricking the kernel and utilizing widely available VPN nodes.

1.2 Optimizing Traffic in Public-Facing Data Centers

Data centers contribute the majority of today's Internet traffic [64], and the past decade has seen significant progress in optimizing the performance of private data center networks (DCNs) [7, 89, 67, 8, 80]. However, these advances have rarely translated into improvements for public-facing data centers that host applications ranging from text-based web services to media delivery platforms and emerging large language model (LLM) inference.

Optimizing public-facing DCNs is fundamentally challenging for two reasons. First, the diverse nature of application types and content obscures traffic predictability from the perspective of network middleboxes. For instance, flow sizes often exhibit heavy-tailed distributions [7]. Second, the design space is heavily constrained by rigid Internet protocol, the uncontrolled client-side and third-party entities along the end-to-end path.

This thesis begins by identifying optimization opportunities in such constrained environments. Growing footprints of major DCNs have been witnessed in recent years, extending the traffic control even into endusers' eyeball networks [16, 64, 124, 32]. We are thus motivated to leverage this expanded control over public-facing traffic to deploy advanced optimization systems. In addition to this Internet-scale infrastructure migration, we spot that valuable but underutilized information resides in the application layer and host systems, such as flow size and server loads. The key challenge then becomes how to synchronize this information between the host and the network middleboxes that implement advanced network optimization systems, such as flow-packet schedulers and predictive load balancers.

To this end, we propose *CloudCookie* [92], a versatile information carrier embedded in the transport or network layer of each data packet. CloudCookie enables in-band synchronization of optimization-relevant information with zero client-side modification, making it practical for the service provider, *i.e.,* DCN authorities, to independently deploy the aforementioned optimization techniques, without client-side and third-party cooperation.

1.3 Leveraging Cross-Directional Dependency in Realtime Interactive Streaming

We then shift our focus from provider-side systems to user-side opportunities, targeting an emerging class of user-oriented applications—*realtime interactive streaming*. This paradigm enables low-latency, bidirectional exchanges of diverse media types, supporting immersive applications such as holographic communication and digital twins that have moved from science fiction to near-reality. However, these applications demand

high-performance network capabilities that far exceed what today's consumer-grade Internet typically provides, posing a significant barrier to their widespread accessibility.

To optimize realtime interactive streaming, we first examine the unique characteristics of its traffic patterns and identify the *cross-directional dependency*, where traffic in one direction depends on the timely delivery or content of traffic in the opposite direction. Building on this observation, we propose a two-flow user-side design [93]. First, we adopt QUIC with the datagram extension [119] to support diverse types of streaming traffic with varying transmission demands. Second, we extend the QUIC priority system to include QUIC datagrams, prioritizing traffic that is depended upon by flows in the opposite direction. This datagram-prioritization strategy can not only protect critical traffic to mitigate buffer overflows caused by streaming microbursts, but also transform chaotic congestion into a predictable process.

1.4 Ubiquitous Multipath Without Multihoming

Despite the benefits of provider-side and user-side optimizations, their effectiveness is ultimately limited by physical resource constraints, particularly the capacity of the end-to-end path. To further extend the design space, we explore provider-user collaborative mechanisms that unlock additional physical network capacity.

As the Internet ecosystem continues to evolve, a wide range of overlay network services have been globally deployed to improve performance, privacy, and security. Typical examples include content delivery networks (CDNs), anonymity-preserving systems such as Tor [122] and iCloud Private Relay [15], and commercial VPNs [152]. These overlay services offer primitives that enable new forms of provider-user collaborative optimization.

In this thesis, we propose *virtual multipath*, a technique that significantly increases the accessibility of multipath capabilities to endusers. Multipath transmission is known to improve connection performance, resilience, and mobility. However, traditional multipath protocols are tightly coupled with multihoming, where multiple network interfaces are required to establish independent paths. This design severely restricts its availability. Even smartphones equipped with multiple network interfaces can satisfy the multihoming prerequisite only when both Wi-Fi and cellular connectivity are simultaneously available. Furthermore, modern smartphone operating systems impose conservative restrictions on MPTCP usage due to concerns about power consumption, data charges, and potential privacy risks. Specifically, iOS supports MPTCP only for HTTP traffic and requires special entitlements, while Android disables MPTCP entirely. Consequently, these limitations discourage server-side adoption: approximately 5% of web servers support MPTCP [58, 59], and merely 0.002% of TCP bytes in a North American traffic dataset belong to MPTCP flows [17].

Virtual Multipath overcomes this limitation by leveraging widely deployed VPN proxy nodes to establish multiple virtual paths over a single physical network interface. The overhead-free nature of these virtual paths eliminates common client-side concerns associated with enabling MPTCP. In contrast to traditional multipath protocols that rely on black-box routing across separate physical interfaces, virtual multipath transport provides application-layer control over path selection. Following the principles of end-user-oriented optimization, Virtual Multipath is implemented by tricking the kernel—disguising Virtual Multipath traffic as MPTCP subflows. The application-guided transport behavior is realized through a userspace front end powered by eBPF [62], together with dummy and TUN devices [114, 115], for ease of deployment.

1.5 Thesis Organization

The remainder of this thesis is organized as follows. In Chapter 2, I present the thesis statement, which serves as the guiding principle of this work. Next, I introduce three major projects that address cross-layer network optimization from the provider side, the user side, and the provider–user collaborative perspective. In Chapter 3, I describe a provider-side optimization for public-facing data center network traffic across Internet protocols. Chapter 4 focuses on user-side optimization, which leverages cross-directional dependency in realtime interactive streaming. Chapter 5 presents a provider–user collaborative approach that enables virtual multipath transport, further unlocking the potential of cross-layer optimization. Finally, Chapter 6 concludes this dissertation by summarizing the key contributions and outlining potential directions for future work.

Chapter 2

Thesis Statement

Enduser-oriented network optimization is enabled by integrating logically decoupled protocol layers into a coordinated system that preserves Internet protocol semantics while adapting to emerging application demands and dynamic network conditions.

Chapter 3

Optimizing Traffic in Public-Facing Data Centers

The performance of data-center networks (DCN) continues to thrive due to ever-improving techniques and systems deployed at different DCN layers [7, 89, 67, 8, 80]. Contrary to the public Internet where a consensus among multiple disjoint entities needs to be reached before any deployment is made, data centers require no such consensus. Indeed, the data center *centralization* is one of the key reasons driving significant progress in this space. However, these advances, which have proven successful within *private* data centers, where both endpoints and the intermediate network devices are controlled by the same data center authority, are rarely deployed in *public-facing* data centers that serve general web users and handle most of today's Internet traffic. [64].

A public-facing data center authority no longer has central control because the network between the client and server can be segmented by multiple entities (Figure 3.1). For example, a client's request must first traverse the local ISP. Next, from the local ISP to the destination data center, the traffic could (1) be shipped on other portions of the public Internet, *e.g.,* ISPs, IXPs, or third-party DCNs (P_1); (2) directly enter the controlled DCNs (P_5); or (3) bounce back and forth between the public Internet and controlled DCNs (P_2, P_3, and P_4), when inter-DC traffic needs to be transited via third parties. The uncontrolled

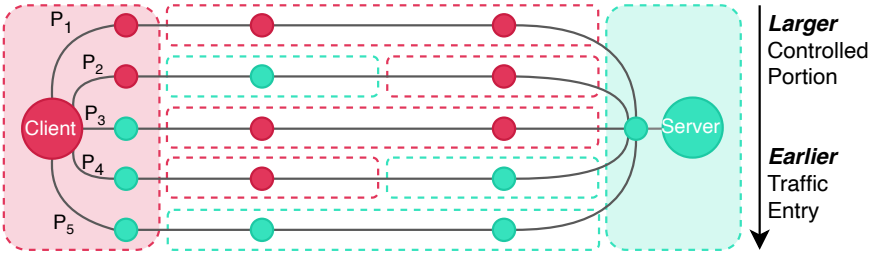


Figure 3.1: Different paths of routing public-facing traffic. Network devices (circles) and autonomous systems (ASes, dashed rectangles) are color-coded: green and red indicate whether the device/AS is controlled by the same authority as the destination DCN. Source and destination ASes are highlighted.

off-net partitions prevent data center advances from being deployed in public-facing data centers.

As a consequence, synchronizing public-facing traffic with optimization signals for achieving advanced traffic handling becomes challenging. First, the compatibility requirements of Internet protocols prevent the deployment of custom headers, *e.g.,* packet encapsulation. Second, due to the inevitable traversal through uncontrolled devices, common signal fields in Internet protocols, *e.g.,* Traffic Class, are low-res and volatile. Third, without the cooperation from the client-side, it is difficult to ensure the signal presence in the traffic from end users. Last, connections initiated by end users, unlike internal services, introduce increased uncertainty, *e.g.,* varied sizes, bursty arrivals, and dynamic routing. Out-of-bound (OOB) channels hence fail to enable timely and spatially aligned signal communication. As in the example of Figure 3.1, it is uncertain which path will be chosen. Given these restrictions, we ask: *if it is possible, and how, to utilize private-DCN advances in public-facing data center settings?*

This question is becoming even more relevant as DCN authorities have significantly moved forward their boundaries and enlarged the network control over the past decades. In particular, tech giants construct new data centers closer to client endpoints (P_2 and P_5)

and deploy off-net infrastructures in clients' eyeball networks (P_3, P_4, and P_5) as shown in Figure 3.1 [16, 64, 124, 32]. The ideal scenario is P_5, despite the client endpoint is still out of control. There are two incentives behind this *flattering* Internet trend. On one hand, directing the traffic to enter the DCN earlier can significantly reduce operational costs, including peering expenses. On the other hand, an increased control over the traffic by the same DCN authority enables more optimization on both data and control planes.

With a growing overlap with inter-DC traffic, public-facing traffic from data centers of these giants now contributes more than half of the global Internet traffic [64].

In this project, we aim to optimize the overall performance of public-facing traffic, and propose CloudCookie as a lever for harnessing the expanded control within the data center. CloudCookie serves as an optimization signal carrier embedded in Internet protocols, offering several properties that are key for advanced public-facing DCN traffic handling: (1) **versatility**, as its spacious coding space allows assorted expressive signals for different optimization techniques; (2) **deployability**, as its bidirectional presence requires no client-side modification and it's hot-swappable to existing Internet ecosystem; (3) **timeliness**, with signals piggybacked on regular packets and synchronized in time and space with their corresponding reaction points; (4) **robustness**, since loss occurs only with dropped packets, with transport layer retransmissions ensuring reliable delivery; and (5) **security**, ensuring signal integrity and resistance to tampering through cryptographic tools.

CloudCookie is *decoupled* from specific optimization techniques. To exemplify its ability to port private data center advances into public-facing traffic, we let CloudCookie carry a combination of signals that enable Shortest Remaining Processing Time (SRPT) flow packet scheduling, for bandwidth allocation, and *predictive* load-aware load balanc-

ing, for traffic distribution. To produce the required signals, i.e., RPT and loads, we design a CloudCookie layer-7 load balancer ($CCLB_7$) that leverages neglected information in applications and systems. To consume the signals, we design a stateless CloudCookie layer-4 load balancer ($CCLB_4$), that makes foresight decisions; and adapt an state-of-the-art in-switch flow scheduler [163]. In §3.4, we further analyze the synergistic optimization of both techniques.

We summarize the contributions of this project as follows:

- We propose CloudCookie, a versatile, deployable, timely, robust, and secure carrier for optimization signals. CloudCookie enables applying state-of-the-art private data center optimizations for public-facing traffic.
- CloudCookie's design is hot-swappable: it utilizes Internet protocols and respects the existing Internet ecosystem; it also requires zero client-side modification.
- To serve CloudCookie, we propose an easy-to-deploy infrastructure suite consisting of an optimization signal producer–$CCLB_7$, and two signal consumers–$CCLB_4$ and *scheduler-on* switch.
- For the first time, we implement SRPT flow scheduling on real-world web traffic, which is characterized by data flows triggered by bursty end-user requests.
- In emerging QUIC/IPv6 networks, CloudCookie utilizes a 64-bit coding space, while, in classic TCP networks, the space is limited to 16 bits. In both contexts, we explore various optimization techniques and their synergistic effects, reducing the 99^{th} percentile flow completion time (FCT) of the majority flows by up to $20\times$.

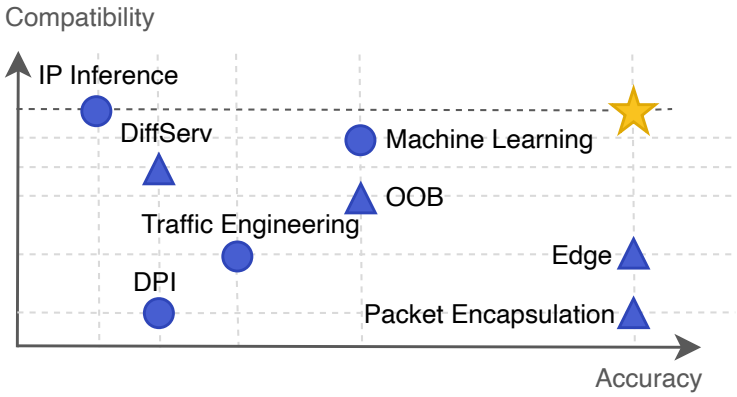


Figure 3.2: Comparison of state-of-the-art methods that exchange (▲) and gain (●) . The upper-right corner is preferred, which is the goal of CloudCookie (★).

3.1 Background and Related Work

The split central control of public-facing traffic introduces challenges in exchanging optimization signals among multiple DCN-controlled network devices (§3.1.1) and obtaining such traffic knowledge (§3.1.2). These challenges highlight certain ignorance that CloudCookie aims to tackle. Additionally, to clarify our design and approach, we present a multi-tier-LB architecture abstraction that enables DCN's adaptation to CloudCookie (§3.1.3) and state-of-the-art flow packet scheduling that CloudCookie targets to activate (§3.1.4).

3.1.1 Challenges in Signal Exchanges

Figure 3.2 compares signal exchange methods in two dimensions: their compatibility with the current Internet ecosystem and their ability to deliver expressive messages accurately and timely. The most straightforward method is to exchange optimization signals by utilizing existing fields in IP headers, which are called differentiated service (DiffServ),

i.e., `DSCP` [1] in IPv4 and `Traffic Class` in IPv6 to encode required information. For example, Mahout [37] uses `DSCP` to encode scheduling signals in the packet. This method boasts high compatibility since it doesn't require any specific modifications, yet there are still two issues to consider: First, the compatibility is compromised because of its volatility. DiffServ fields are commonly reset or remarked when crossing controlled network boundaries due to compatibility, policy, and security concerns [38]. Second, DiffServ's low resolution, *i.e.,* 6-bit long excluding 2 ECN bits, forbids deploying fine-grained optimizations, such as scheduling packets according to flow sizes or RPT.

On the other hand, the OOB communication channel can serve as an effective method for exchanging signals between network devices in private data centers. It offers ample coding space, thereby resolving issues related to resolution. OOB channels can be established through dedicated messages, *e.g.,* pHost [63] and Homa [104], or dedicated connections, *e.g.,* the software-defined networking (SDN) family. Nevertheless, deploying OOB methods in public-facing data centers faces several challenges. Synchronization issues are at the forefront. First, the inevitable delays, between optimization signals and their serving traffic, damage the accuracy of signals. The damage increases as the reaction points become close to the client, but distant from the data center where the signal is generated. We observe recent efforts that optimizes public-facing traffic with OOB methods [160, 135, 124]. Although these proposals offer improvements in directing traffic across multiple paths, we maintain that there remains considerable potential for optimization within the data plane, particularly for bursty and ephemeral public-facing traffic. Second, the spatial unsynchronization of OOB signals with their corresponding reaction points raises additional concerns regarding server compatibility. Transmission paths can differ from one connection to another or even from packet to packet, which

[1] `DSCP` stands for Differentiated Services Code Point, which was originally defined as `ToS`, Type of Service [57, 69].

complicates the issue. Consequently, it is challenging to ascertain which network devices are the recipients of a specific signal. Therefore, it becomes infeasible to determine which network devices ought to be the subscribers for a certain signal. Third, the implementation of OOB methods entails increased overheads on resources and bandwidth. The deployment of these mechanisms typically requires centralized controllers, and the delivery of OOB signals generates extra traffic that competes with regular data flows. While such overheads may be acceptable in private data centers with over-provisioned resources and bandwidth, this assumption does not hold in public-facing data centers.

Packet encapsulation resolves synchronization issues by aligning the optimization signals in bound with the regular traffic that they serve. Generic tunneling, *e.g.,* GRE [87] and GENEVE [68], and custom encapsulation, *e.g.,* VL2 [67] and PDQ [72] are popular methods in this category. However, a grave concern is that packet encapsulation is often an all-or-nothing choice. Deploying this technique for a service requires changing all servers, clients, and optional intermediate network devices depending on the task. In the case of load balancing, an unsupported load balancer would reject all encapsulated packets, while a supported load balancer would discard any packet that is not encapsulated. Additionally, due to extra headers, the inflated packet size may exceed the MTU size allowed on the public Internet. Based on these two reasons, we rate the compatibility of this method as the lowest level. Moreover, computational overheads and delays are multiplied by the need for encapsulation and decapsulation to be performed on each intermediate device.

So far, *all* above methods fall short when they come to preserving signals in packets from unmodified normal clients. A solution is to deploy edge servers creating a non-intrusive illusion for client endpoints. Taking encapsulation as an example, the edge

server is responsible for (1) decapsulating packets going to clients, (2) maintaining an enormous table to cache decapsulated headers, and (3) looking up the table and recovering corresponding headers. It fixes the compatibility problems for clients, whereas it does not change the situation of servers and intermediate devices. Moreover, it raises two new concerns. First, querying a huge table is computationally intensive and, consequently, not scalable. Second, the effectiveness diminishes if the edge server is located far from the client endpoint. Thus, one of our primary motivations is to identify a technique that can offer high accuracy without compromising compatibility.

3.1.2 Challenges in Gaining Traffic Knowledge

Gaining traffic knowledge is essential to derive optimization signals, and hence another challenge in handling public-facing traffic. Figure 3.2 compares state-of-the-art methods. We categorize these methods based on their underlying causes, and expand the elaboration accordingly.

Low Visibility in Middleboxes

Is it possible to utilize the increase in DCN-controlled middleboxes for better traffic insight? Unfortunately, the visibility of Internet traffic is shrinking due to evolving technologies, standards, and practices. While these changes have positive intentions, they restrict traffic inference in middleboxes, and hence limit various optimization methods. First, web services and content can no longer be inferred by IP addresses. On the one hand, web services are less coupled with IP addresses. A service may be associated with multiple IP addresses, while an IP address can refer to multiple services. On the other hand, the response content of web service is not idempotent and is determined upon each request individually. Even when sending two identical requests, the second one might be times larger than the first in size due to the change in content, *e.g.,* the Twitter feeds. Second,

deep packet inspection (DPI) has stronger insights into complex traffic than IP inference. However, it is at the compatible margins of today's Internet traffic as over 90% of traffic is encrypted [66, 105]. Additionally, the implementation of HSTS preload list [33] and the rolling out of HTTP/3 [24] are making traffic encryption become a mandatory standard. Therefore, it is anticipated that an effective measure should yield knowledge about public-facing flows directly, rather than relying on insights from the middlebox.

Client-Initiated Connection

Public-facing flows always initiated by end users increase the uncertainty of traffic patterns and distribution which further differentiate themselves from intra- and inter-DC flows. First, indirect observation, *e.g.,* flow aging [127, 20], is simple yet effective in scheduling flows with sparsely distributed sizes. In contrast, given that the majority of public-facing flows are short [23, 8], indirect observation could prove ineffective or detrimental. Second, to effectively infer flow sizes, efficient machine learning algorithms are proposed [48]. Still, the effectiveness is questionable in public data centers. Recent literature reported that traffic distributions and volumes varying from service to service and time to time were observed in public-facing data centers [164]. Last, to avoid congestion and optimize bandwidth usage, traffic engineering (TE) is often adopted with SDN techniques to split and direct traffic among multiple paths. Besides the signal propagation delay indicated in §3.1.1, the TE controller makes a centralized decision based on collective observations, which tends to be outdated for ephemeral public-facing flows. For the same reason, TE leaves data plane optimization opportunities, *e.g.,* flow scheduling, which can further optimize network performance. In fact, the other motivation is to leverage the ignored application consciousness to gain accurate traffic knowledge rather than blindly infer it.

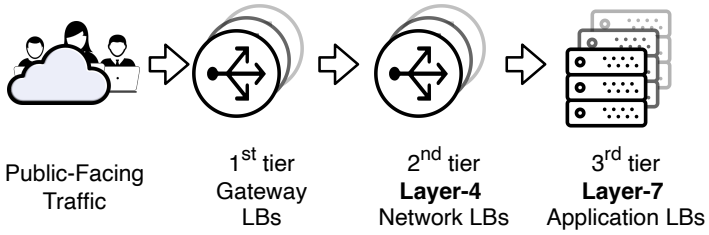


Figure 3.3: Multi-tier load balancers that splits public-facing traffic in data centers.

3.1.3 Multi-Tier Load Balancers

Independent from diverse network topology, let us view the DCN from the perspective of multi-tier load balancers (LBs). In order to serve massive requests, data center providers employ a multi-tier structure of load balancers, as shown in Figure 3.3, to split the huge traffic around the world [118, 56].

Under this structure, the 3rd tier Layer 7 (L7) LBs are the end hosts providing web services and terminating connections. These L7-LBs are identified through Direct IPs (DIPs). However, these DIPs are internal IP addresses and hence are unrecognizable to the outside networks. In contrast, virtual IPs (VIPs) are public IP addresses announced to the DNS. A web service is identified through one or a set of VIPs, and it can also be hosted on one or many L7-LBs, *i.e.,* multiple DIPs. This decoupled design of VIP and DIP benefits cloud tenants with the flexibility to configure their services.

The 1st tier load balancers are data center *gateways* performing Equal-Cost Multi-Path (ECMP) load balancing to forward incoming packets from the Internet to the next hops. While the 1st tier load balancer is responsible for assigning the packet to one of the 2nd tier Layer 4 (L4) LBs that recognizes this VIP. It is the L4-LB's responsibility to assign the packet to a specific L7-LB by translating this VIP to a DIP of that L7-LB. Either the serving, 1st tier or 2nd tier, load balancers may vary for different incoming packets of the

same connection, but these packets have to terminate on the same 3rd tier L7-LB, *i.e.,* per-connection consistency (PCC). Otherwise, a different L7-LB cannot recognize the packet and respond properly.

L4-LBs can be categorized as load-agnostic or load-aware. Load-agnostic balancers, like round-robin and hash-based systems, perform inconsistently, especially during off-peak periods, as they do not use load information [50]. In contrast, load-aware balancers (both statefule [101, 118] and stateless [22, 112]) measure load distribution to make informed assignments. However, stateful variants struggle with scalability, and neither type meets the ideal L4-LB properties (§3.3.2).

3.1.4 Flow Packet Scheduling

Data centers handling public-facing flows face a unique challenge due to the heavy-tailed distribution of flow sizes [7], where a minority of long flows account for most of the traffic, while the majority are shorter flows. Implementing the optimal SRPT flow scheduling is, however, NP-hard. Achieving near-optimal average FCT in this context has seen advances through flow scheduling on both switches [8, 163, 5] and end-hosts [63, 104]. The effective deployment of SRPT flow scheduling in public-facing data centers is challenging due to the difficulty in obtaining and exchanging necessary flow knowledge, *e.g.,* RPT, as explained in §3.1.2 and §3.1.1.

3.2 CloudCookie Design

We view CloudCookie as the signal *carrier* that establishes a synchronized in-bound communication channel for optimization signals to piggyback on public-facing traffic.

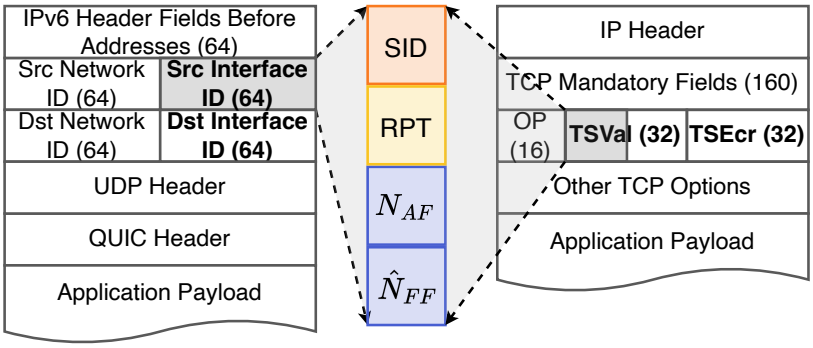


Figure 3.4: Accommodations of CloudCookie in Internet protocol headers.

Given that, our first decision is to position CloudCookie within the transport and network layers, as opposed to the application layer[162], stems from the favorable trade-off between visibility and expressiveness. Moreover, in light of CloudCookie's application in public-facing traffic where client-side modification is not feasible, the key idea is to ensure bidirectional presence by utilizing the inherent *automatic echo* within widely-deployed Internet protocols. Ultimately, we discover two CloudCookie implementations: (1) in QUIC/IPv6 networks, and (2) in TCP networks. Figure 3.4 illustrates the accommodation of CloudCookie in both schemes.

CloudCookie, in its QUIC/IPv6 implementation, leverages the *composite* features of both protocols. Initially, we utilize a key aspect of QUIC, an emerging transport layer protocol initially developed by Google and standardized by the Internet Engineering Task Force (IETF) [86, 79], and foundational to HTTP/3 [24]. This feature is QUIC's unique `Connection ID`, which decouples the connection identifier from the traditional 5-tuple. Originally intended to preserve connections when a client's IP address changes. We propose repurposing this feature on the server-side and encoding CloudCookie within the L7-LB's IPv6 addresses. We harness a feature of IPv6: Internet devices commonly receive a /64 subnet [76], bifurcating the IPv6 address into `Network ID` and `Interface ID` [43,

110, 76]. While the `Network ID` is used for routing, the `Interface ID` identifies an interface. Thus, with QUIC as the transport layer protocol, we encode CloudCookie into the `Interface ID`, utilizing the 64 least-significant bits (LSB) of the IPv6 address space. The automatic echo feature is ensured via the IP address swap. CloudCookie is embedded in the `Src Interface ID` for server-originated packets and in the `Dst Interface ID` for client-originated packets. CloudCookie in QUIC/IPv6 networks provides a substantial 64 bit coding space to carry various optimization signals.

Inspired by Cheetah, an L4-LB that shifts the memory of flow assignments to TCP Timestamps of flow packets [22], CloudCookie, in its TCP implementation, extends the use of TCP Timestamps for versatile optimization signals. TCP Timestamps is a TCP option including two equal-size parts: `TSVal` and `TSEcr`, where either one is 32 bit long. TCP Timestamps is originally designed for round-trip measurement of retransmitted packets and the Protect Against Wrapped Sequences (PAWS) mechanism [27, 26]. TCP Timestamps also possess the automatic echo feature. Either connection peer receiving a packet with TCP Timestamps will copy the `TSVal` value to the `TSEcr` field of its upcoming departure packets, and set the `TSVal` to be an independent value of its own interest. Specifically, once CloudCookie is injected into `TSVal` of packets for a downstream public-facing flow, the subsequent upstream packets within the same connection will observe CloudCookie in their `TSEcr` fields. Thus, the bidirectional visibility of CloudCookie is guaranteed. Following Cheetah's practice to preserve the original functionality of TCP Timestamps [22], CloudCookie conservatively occupies the 16 most significant bits (MSB) of `TSVal` or `TSEcr` so that the manipulation of CloudCookie can be reversed before delivering the packet to upper layers. In comparison to CloudCookie in QUIC/IPv6 networks, CloudCookie in TCP networks has a more constrained space forcing a lower resolution of piggybacked signals.

While CLOUDCOOKIE is a versatile signal carrier decoupled from specific DCN optimization techniques, in this project, we choose load-aware load balancing and SRPT flow scheduling. Therefore, CLOUDCOOKIE encodes SID for flow assignment, RPT for flow scheduling, and two load metrics–the active flow number N_{AF} and the future flow estimation $\hat{N_{FF}}$–for updating $CCLB_7$ (server) loads on $CCLB_4$, as the scheme shown in Figure 3.4. Furthermore, we advise the DCN authority shield CLOUDCOOKIE with cryptographic tools, *e.g.,* AES. It is noteworthy that only CLOUDCOOKIE, rather than the whole packet, should be encrypted. Encrypting or decrypting a maximum 64-bit string of AES-128 can be in line rate on programmable switches [30]. Encrypting can not only protect malicious modification and forgery of CLOUDCOOKIE, but also help early filter out invalid traffic.

3.3 CloudCookie In Action

The lifecycle of CLOUDCOOKIE starts with $CCLB_7$, which acts as the signal *producer* integrating application layer knowledge into CLOUDCOOKIE. DCN middleboxes, including $CCLB_4$ and the scheduler-on switch, serves as the signal *consumers,* performing foresight load balancing and SRPT flow packet scheduling based on this knowledge.

3.3.1 CCLB7: Signal Producer

The key philosophy of $CCLB_7$ is to extend any state-of-the-art L7-LB to integrate the application layer awareness into CLOUDCOOKIE and thereby optimize the public-facing flows. Instead of building from scratch, we patch a regular L7-LB with two additional roles to make it an optimization signal *producer*:

- **Signal *collector***: collecte flow knowledge and measure the L7-LB's own load.

- **Signal *feeder***: encode optimization signals into CLOUDCOOKIE and inject it in public-facing flows' packets.

As outlined in §3.2, $CCLB_7$ plays a crucial role in generating RPT and two types of loads (N_{AF} and $\hat{N_{FF}}$) for SRPT flow scheduling and load-aware load balancing. To illustrate its functionality, we expand upon the design of $CCLB_7$ within TCP networks, which operates similarly to its counterpart in QUIC/IPv6 networks. The formula for RPT is given by $RPT = \frac{flow\_size - bytes\_sent}{throughput}$. The initial step involves acquiring `bytes_sent`, conveniently available in the kernel via `struct tcp_sock`. `tcp_sock`, the internal structure managing TCP connection states, is extended with a new field `acc_flow_size`, allowing the application layer to report the flow size. Given the common practice of connection reuse, where a single connection may handle multiple data flows, this field is designed for accumulative updates. The next component involves the application layer, specifically an L7-LB, processing a new request to determine the flow size. The approach varies depending on the type of request. Preferably, the flow size is extracted directly from the request. For static object requests, the flow size can be obtained from associated metadata, a method particularly efficient for high-volume transmissions, such as video streaming, where a video is sent in chunks. For dynamic content requests, where the response size is not predetermined, the flow size can be determined after the response composition and before initiating the `send()` system call to transfer the response to the socket buffer. To accommodate flow size reporting in both scenarios, the default `send()` is replaced with `send_cc()`, which wraps the original `send()` call and updates `acc_flow_size`. The final component, real-time connection throughput, presents significant challenges. In our study, we adopt a methodology similar to that of previous research [8, 5, 163], which involves assuming a constant bandwidth. This approach has proven to be effective in our evaluations.

The other optimization signal in $CCLB_7$ is its own loads. In addition to the current load number (N_{AF}), we propose estimating future flows ($\hat{N_{FF}}$) to maximize the use of flow knowledge. $CCLB_7$ utilizes two global counters for this purpose. A new flow increments counter(N_{AF}) upon invoking `send_cc()`. If the flow size surpasses a predefined "ahead-looking" threshold, indicating its likelihood of persisting in the near future, counter($\hat{N_{FF}}$) is also incremented. Conversely, when the remaining bytes ($flow\_size - bytes\_sent$) fall below this threshold, the counter($\hat{N_{FF}}$) is decremented. Similarly, counter(N_{AF}) is decremented once the remaining bytes reach zero. It is important to note that these increment/decrement operations are at the flow level. In cases of multiple requests reusing the same connection (in HTTP/1.1+) or server-pushes (in HTTP/2+), both counters react accordingly.

Offloading challenging tasks of flow and load measurements from DCN-controlled middleboxes to $CCLB_7$, paradoxically, adds minimal overhead, as all extended operations have O(1) complexity. These signals are then fed into CloudCookie for use by stateless DCN-controlled middleboxes to execute relevant optimizations.

3.3.2 Middleboxes: Signal Consumers

To fully harness the potential of CloudCookie's carried signals, we have engineered DCN-controlled middleboxes tailored for load-aware load balancing and SRPT flow scheduling. This includes the creation of $CCLB_4$, which utilizes the loads reported from CloudCookie, and the porting of a state-of-the-art flow scheduler, AIFO [163], to public-facing DCNs by supporting CloudCookie. This subsection elaborates on the complete end-to-end lifecycle of CloudCookie and illustrates how DCN-controlled middleboxes utilize it to optimize public-facing traffic.

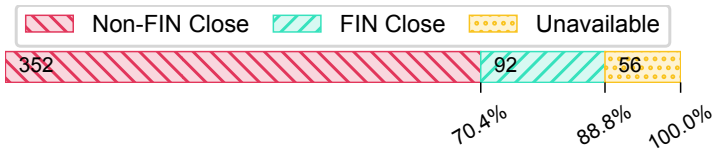


Figure 3.5: A measurement over Alexa top 500 websites shows the diversity and uncertainty of TCP connection terminations.

Cheetah's Teachings: Shaping the Ideal Load Balancing

CCLB_4 draws inspiration from Cheetah [22] by using packet headers to store flow assignments, thereby implementing a stateless L4-LB that ensures PCC. Cheetah has two variants: Cheetah_{RR} for round-robin and Cheetah_{LL} for least-loaded load balancing. However, even Cheetah_{LL}, which considers L7-LB loads in flow assignments, exhibits inherent design flaws. First, Cheetah_{LL} independently measures loads, leading to biased observations. This is due to its method of counting active flows, incrementing or decrementing based on `SYN` or `FIN` packets, which does not reliably indicate connection terminations in real-world scenarios. Our analysis of Alexa top-ranking websites [6] revealed that less than 20% use `FIN` for graceful connection closure, as Figure 3.5 shows.

Second, even with accurate active flow counting, the locally measured loads are skewed compared to the global distribution. To illustrate this, we simulate the distribution of 10,000 flows, each with sizes sampled uniformly, from $N \in [2, 4, 6, 8]$ L4-LBs to 16 L7-LBs from both local and global perspectives. The results, shown in Figure 3.6, indicate higher variance in assignments from the local perspective, worsening as the number of L4-LBs, *i.e.,* N, increases.

Last, even with accurate measurements of global load distributions, optimizing public-facing flows remains inadequate if flow awareness is ignored. As discussed in §3.1, public-facing flows exhibit high uncertainty. A representative uncertainty is the heavy-tailed dis-

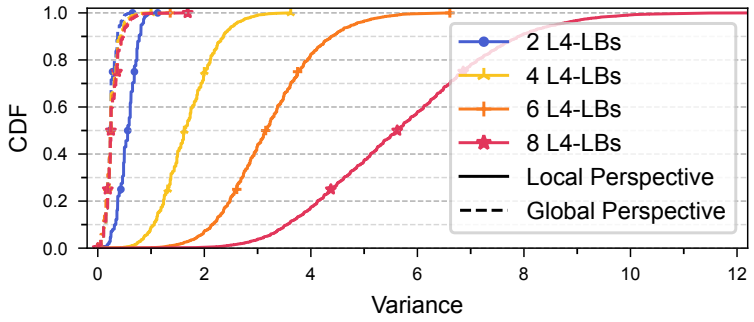


Figure 3.6: CDF of cross-L7-LB variance in flow assignments over time, from L4-LBs' local and global perspectives.

tribution in flow sizes [7, 23], necessitating future load estimation to rectify flow assignment decisions. For instance, consider assigning an incoming flow between two L7-LBs, A with 15 active flows and B with 20. Cheetah$_{LL}$ would assign the flow to A. However, if 5 and 15 flows terminate on A and B respectively within the next second, the load distribution will reverse. Cheetah$_{LL}$ does not account for flow awareness and leads to suboptimal flow assignments.

From the lessons of Cheetah's design, we derive that the ideal L4-LB should encompass three core properties: *precise measurement*, a *global perspective*, and *flow awareness*. CCLB$_4$ is designed to embody these properties by utilizing the load information reported by each CCLB$_7$ through CloudCookie. First, the loads reported by CCLB$_7$ are the ground truths. Second, the continuous update of load information in every packet sent to CCLB$_4$ endows it with an approximate global perspective. Third, the loads reported in CloudCookie, consisting of N_{AF} and $\hat{N_{FF}}$, capture both the current and future load levels. CCLB$_4$ maintains a set of maps to link VIPs, SIDs, DIPs, and loads of CCLB$_7$s.

Figure 3.7 outlines the workflow of DCN-controlled middleboxes, including CCLB$_4$ and scheduler-on switches, in response to CloudCookie. Depending on CloudCookie content and flow states, the workflow is divided into four phases: **Assigning Phase** (Fig-

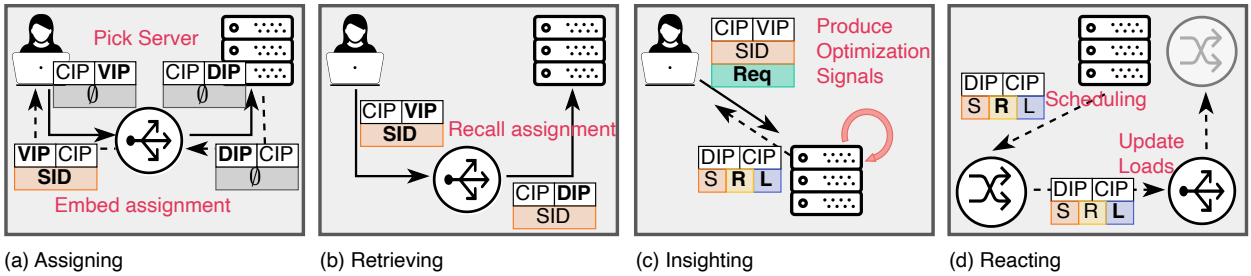


Figure 3.7: Workflow depicting the production and consumption of CLOUDCOOKIE in DCN-controlled infrastructure. Packets are shown in the scheme of [SrcIP | DstIP / CLOUDCOOKIE]. "CIP" refers to the client IP. Solid and dashed arrows indicate packets in upstream and downstream directions.

ure 3.7 (a)): This phase begins with a client initiating a connection to a VIP obtained from DNS. Upon receiving the first packet (TCP's `SYN` or QUIC's `Initial`) at CCLB_4, candidate CCLB_7s associated with the VIP are considered. CCLB_4 picks the candidate with the lowest load score. Score the i-th CCLB_7 as:

$$\text{Score}(i) = \alpha \times N^i_{AF} + (1 - \alpha) \times \hat{N}^i_{FF} \tag{3.1}$$

where α is a weight coefficient to balance the load balancing sensitivity between the active (N^i_{AF}) and estimated future ($\hat{N}^i_{FF}$) flow numbers. The destination address in the packet is then changed from VIP to DIP and forwarded to CCLB_7. In the downstream direction, any packet traversing CCLB_4 has its source address changed back from DIP to VIP, with the corresponding SID embedded in CLOUDCOOKIE. **Retrieving Phase** (Figure 3.7 (b)): This phase happens for every subsequent upstream packet passing CCLB_4. SIDs in CLOUDCOOKIE are used to retrieve the previous assignment, ensuring consistent delivery to the same CCLB_7. **Insighting Phase** (Figure 3.7 (c)): This phase is triggered when a complete request gets delivered. CCLB_7 collects flow knowledge, such as RPT, during request parsing or response composition. This flow knowledge, combined with load information, is packed into CLOUDCOOKIE. **Reacting Phase** (Figure 3.7 (d)): In the downstream path,

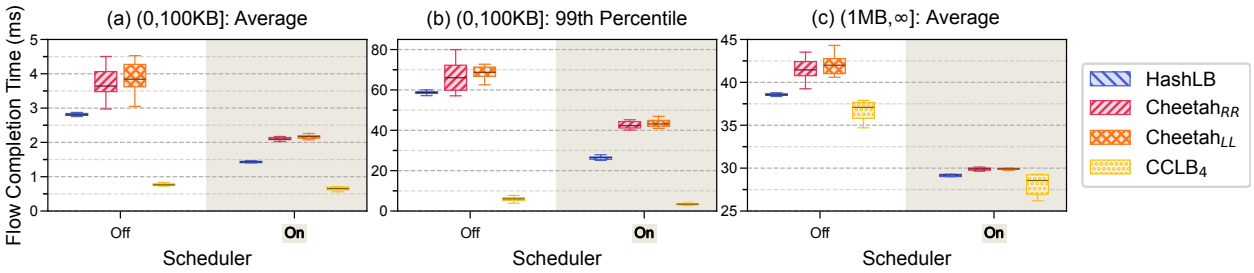


Figure 3.8: Performance of different L4-LBs with or without flow scheduler under high load (0.8) demonstrates the individual and synergistic performance improvements brought by CLOUDCOOKIE.

CCLB$_4$ updates the load information, while scheduler-on switches, even in client-side eyeball networks, utilize SRPT flow scheduling for proactive bandwidth allocation optimization.

CLOUDCOOKIE imparts *stateful* capabilities to DCN-controlled middleboxes while allowing them to remain inherently *stateless*. This aspect is critical for enabling data center advances without compromising scalability. Furthermore, CLOUDCOOKIE facilitates the synergistic optimization on public-facing traffic by fusing multiple data center advances.

3.4 Evaluation

In the development of CLOUDCOOKIE and its associated infrastructure, we have executed two Proof-of-Concepts (PoCs) to validate various facets of our design. The first, a *Functional PoC* extending the Linux kernel, demonstrates CLOUDCOOKIE's basic viability in standard operating systems and public-facing traffic. In this PoC, we also evaluate the overhead associated with integrating CLOUDCOOKIE into CCLB$_7$. The second, a *Performance-Oriented PoC*, utilizes a DPDK-based HTTP suite and L4-LBs implemented on BESS framework [94, 106, 70], and a Tofino (P4) switch [77, 116] for running the flow scheduler, to

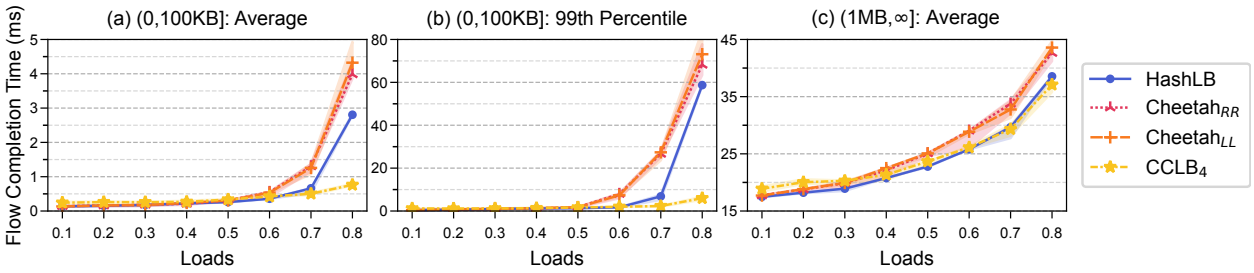


Figure 3.9: Performance of different L4-LBs under varying loads (scheduler disabled) further confirms the CCLB$_4$'s effectiveness. The shaded area indicates the interquartile range (IQR) of each L4-LB.

conduct performance assessments, particularly in high-speed data center environments. With this implementation, we simulate real-world HTTP traffic and evaluate the effectiveness of CloudCookie's approach in 100-Gbps high-speed network settings, thereby demonstrating its potential to substantially improve network performance.

In this evaluation, we formulate five key questions:

- Does load balancing and flow scheduling have a synergistic impact on network performance, given that they focus on different aspects of optimization? (§3.4.1)
- Is stateless CCLB$_4$ able to improve load balancing effectiveness by integrating load reports from CCLB$_7$ via CloudCookie, compared to its alternatives? (§3.4.2)
- Is CloudCookie able to enable SRPT flow scheduling for public-facing flows? (§3.4.3)
- What is the performance of CloudCookie TCP and CCLB$_4$ under different scoring settings? (§3.4.4)
- What is the cost of introducing CloudCookie? (§3.4.5)

Performance-Oriented PoC Testbed

We have a AIFO [163] flow scheduler deployed on the Tofino switch. Four physical machines are connected to the switch via 100-GbE links. Each of these machines has one AMD EPYC™ 7302P 16-core processor at 3.0 GHz with hyper-threading disabled. Four HTTP clients are deployed on one dedicated machine. Traffic from each HTTP client will arrive first at an L4-LB and then at L7-LBs running at the three remaining machines. The switch-clients link is the bottleneck link because these clients share the same link. For performance isolation, we run each HTTP-client-L4-LB pipeline with dedicated CPU cores and a virtual NIC by using SR-IOV [144]. [2] Similarly, each of the other three machines has four CCLB_7 running on, with each one having its own dedicated core and virtual NIC.

Experimental Settings

We simulate real-world public-facing flows from a well-known datacenter websearch workload [7], which matches the practice in precedent research [7, 8, 120, 63, 5, 163]. In this heavy-tailed workload, the flow size varies from 8 KB to 28 MB and has a 95^{th} percentile of 4 MB. On each HTTP client, we generate flow arrival events, *i.e.,* sending GET requests, through the Poisson process. Client requests are distributed to L7-LBs by associated L4-LBs. Four L4-LBs are considered in this study: (1) HashLB, which assigns flows based on the 5-tuple hash value; (2) Cheetah_{RR}, which memorizes flow assignments in TCP Timestamps and uses round-robin for new assignments; (3) Cheetah_{LL}, a variant of Cheetah that selects the least loaded CCLB_7 based on local measurements for new assignments [22]; and (4) CCLB_4, which utilizes CLOUDCOOKIE-reported loads for flow assignment decisions. The first two are load-agnostic, while the latter two are load-aware. The target network loads vary from 0.1 to 0.8. Each experimental setting involves 10 repetitions, with

[2] In this way, pipelines can run with minimal interference because packets are switched in hardware on host.

requests emitted for 10 seconds in each run, unless otherwise specified. We primarily evaluate the performance of CLOUDCOOKIE in QUIC/IPv6 networks. The default weight coefficient α in the CCLB$_4$ score function (Eq 3.1) is 0.5 to balance loads N_{AF} and $\hat{N}_{AF}$ equally.

3.4.1 Studying The Synergy of Multiple Optimizations

Flow scheduling optimizes bandwidth allocation for network flows in local links, whereas load balancing distributes traffic to enhance overall network efficiency. To assess the synergistic benefits of CLOUDCOOKIE, we concentrate on high-load scenarios (load=0.8), comparing CCLB$_4$ against HashLB, Cheetah$_{RR}$, and Cheetah$_{LL}$, with the flow scheduler either activated or deactivated.

Figure 3.8 presents the average FCT for short flows (0, 100KB], the 99$^{\text{th}}$ percentile FCT for these flows, and the average FCT for long flows (1MB, ∞]. Without the flow scheduler, CCLB$_4$ improves performance for both short and long flows: it achieves a $4-5\times$ reduction in average FCT and a $10-11\times$ in the 99$^{\text{th}}$ percentile FCT for short, the majority, flows compared to alternatives. For long flows, CCLB$_4$ reduces the average FCT by up to 12%. After enabling the flow scheduler, the FCT reductions by CCLB$_4$, for short flows, are $2-3\times$ for the average FCT and $8-12\times$ for the 99$^{\text{th}}$ percentile. The absolute FCTs decrease for all L4-LBs. And, with fully utilizing CCLB$_4$ and the flow scheduler, the overall improvement in the 99$^{\text{th}}$ percentile FCT can reach up to $20\times$ for the majority flows. This is because the two optimizations target different optimization aspects. However, the performance gap between different L4-LBs narrows. The underlying rationale is that applying flow packet scheduling can decrease packet retransmissions and indirectly reduce the overall network load. In summary, flow packet scheduling and load balancing can collaboratively enhance the efficiency of public-facing flows.

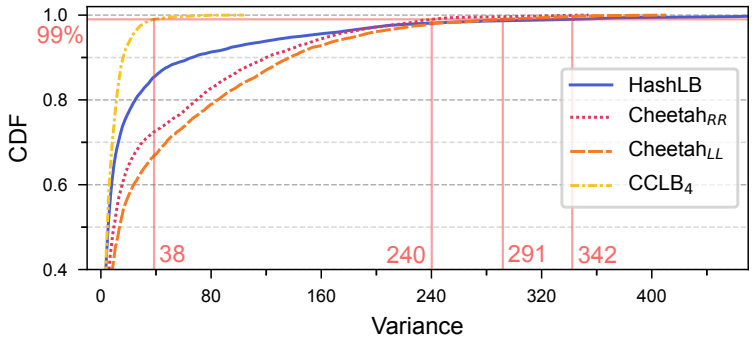


Figure 3.10: The ground-truth cross-L7-LB variance confirms the superiority of CCLB$_4$.

3.4.2 Demonstrating The Effectiveness of CCLB4 Design

From the trailer in §3.4.1, CCLB$_4$ consistently outperforms its competitors with or without the flow scheduler under high loads. Its performance is notably superior in managing both short and long flow types. We now conduct an ablation study to further understand CCLB$_4$'s effectiveness.

First, we disable the flow scheduler and compare the four L4-LBs under varying loads. Results are shown in Figure 3.9. Under light traffic load (< 0.6), all L4-LBs demonstrate comparable service levels. However, for higher load (≥ 0.6), performance gaps between L4-LBs become increasingly evident. Across various load levels, the performance ranking remains constant: CCLB$_4$ consistently outperforms all others, followed by HashLB, and then the two Cheetah variants.

We are taken aback by the underperformance of Cheetah L4-LBs, especially when compared to HashLB. But this indeed validates our earlier observation: the Cheetah's ignorance discussed in §3.3.2. According to Figure 3.9, we notice that Cheetah$_{LL}$ performs even worse than Cheetah$_{RR}$. This is understandable: Even though Cheetah$_{RR}$ is load-agnostic, it operates under the assumption that server loads are uniformly distributed.

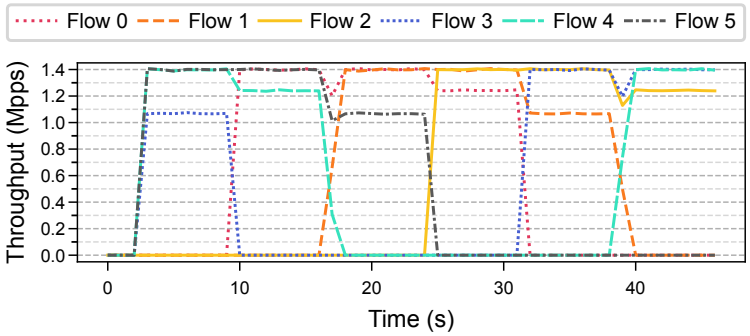


Figure 3.11: Shifts in bandwidth allocation in reaction to rotating RPTs in six flows, as carried by CloudCookie.

In contrast, Cheetah$_{LL}$ tends to result in more erroneous flow assignments, because of its biased perspective based on locally measured server load distributions. CCLB$_4$ adopts a fundamentally different strategy by fully offloading load measurement to the server, *i.e.,* CCLB$_7$, with updates facilitated by CloudCookie. Notably, CCLB$_7$ is inherently the most credible source for its own load data.

To further verify the load balancing performance, we set up Chrony [90] to elevate the time accuracy among machines, reducing it from several seconds to approximately ten microseconds. Then we periodically dump real-time active flow numbers on the 12 CCLB$_7$s and align them to calculate the variance. Figure 3.10 is the CDF of the variance collected for different L4-LBs, further confirming the superiority of CCLB$_4$ in load balancing. In summary, serving CloudCookie, stateless CCLB$_4$ can perform load-aware load balancing from an approximate global perspective.

3.4.3 Validating SRPT Flow Scheduling

In addition to results in §3.4.1, we design an experiment of 6 flows to verify that CloudCookie is effective in enabling SRPT scheduling for public-facing flows. We conducted an experiment by bypassing the L4-LB and directly transmitting 6 continuous data flows

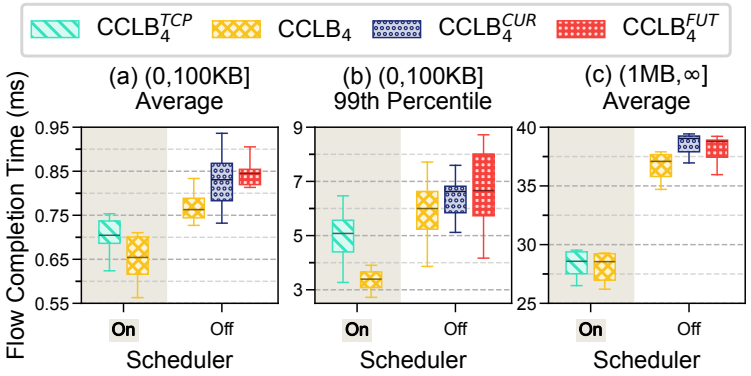


Figure 3.12: Performance of $CCLB_4$ variants under high load (0.8).

from a server to a client, operating on a best-effort basis. Each flow is uniquely identified by an RPT assigned via CloudCookie, which determines its priority. Initially, the flows, from Flow 0 to Flow 5, are arranged with RPTs in descending order, corresponding to ascending priorities. Every 5 seconds, the RPT assignments are rotated. For instance, after the first rotation, the RPTs are reordered in descending order as Flow Flow 1 to Flow 5, and Flow 0.

Figure 3.11 shows the throughput of every flow over time. Initially, only Flow 3, 4, and 5 grab shares of the bandwidth, because the CloudCookie of them carries the shortest RPTs in that period. Later, after the first rotation, Flow 3 quickly handles its bandwidth share to Flow 4, 5 and the newcomer, Flow 0, due to the RPT reranking. Similar changes apply to subsequent rotations. Besides, during each stable period, we can observe the flow throughputs are divided into 3 tiers: the two flows with the shortest RPT take the maximum throughput, followed by the third shortest RPT flow. The remaining flows do not get any bandwidth share. CloudCookie is hereby proved to be an effective and flexible signal carrier to enable SRPT flow packet scheduling.

3.4.4 More Experiments on CCLB4 Variants

CLOUDCOOKIE working in TCP networks and carrying lower-resolution signals creates the CCLB$_4^{TCP}$ variant. There are two additional variants: CCLB$_4^{CUR}$ and CCLB$_4^{FUT}$, due to different flow assignment flavors: only from the number of current active flows ($\alpha = 1, Score(i) = N_{AF}^i$), and only from the estimation of future flows ($\alpha = 0, Score(i) = \hat{N}_{FF}^i$).

Figure 3.12 shows the comparative results of these variants alongside the default CCLB$_4$. The initial comparison focuses on CCLB$_4$ targeting QUIC/IPv6 networks versus CCLB$_4^{TCP}$. Although CCLB$_4^{TCP}$ shows an 8% increase in average FCT and a 50% increase in the 99$^{\text{th}}$ percentile FCT compared to CCLB$_4$, its performance still surpasses that of CCLB$_4$ when the flow scheduler is disabled. With a properly designed coding pattern as in Figure 3.4 (b), CCLB$_4^{TCP}$ can still achieve competitive performance. The second comparison evaluates various flow assignment flavors, with the flow scheduler disabled. As anticipated, CCLB$_4$, which accounts for both the current number of active flows and future flow estimations, outperforms CCLB$_4^{CUR}$ and CCLB$_4^{FUT}$, each of which considers only one of these aspects. These results highlight the practicality of our heuristic approach in estimating future flows from current flow RPTs, thereby leveraging application layer flow awareness to enhance flow optimization.

3.4.5 Quantifying CloudCookie's Overheads

While CLOUDCOOKIE offloading flow insighting and load measurement to CCLB$_7$ from DCN middleboxes introduces additional responsibilities, the overhead remains minimal, as detailed in §3.3.1, with all operations being of constant complexity. This subsection quantifies such overhead in both Performance-Oriented and Functional PoCs.

		Idle	Vanilla L7-LB	$CCLB_7$
Mean	User	0.4%	6.1%	6.6%
	Kernel	0.4%	15.7%	16.6%
Median	User	0.0%	6.3%	6.6%
	Kernel	0.0%	15.8%	16.9%
P99	User	4.0%	7.7%	8.4%
	Kernel	2.4%	16.7%	18.4%

Table 3.1: CPU utilization of $CCLB_7$ Functional PoC, in user and kernel space, demonstrates consistent minimal overheads as of the Performance-Oriented PoC. "Idle" indicates that there is no L7-LB running.

We measure the CPU cycles required for inferring RPT and loads per outgoing packet: load retrieval consistently requires 30 cycles, being a simple memory read, while RPT computation varies between 30 (median) and 630 (99^{th} percentile) cycles. On a 3.0 GHz processor, this translates to a delay of 0.22 μs per packet at the tail, a negligible amount compared to the typical RTT of public-facing flows, *e.g.,* Facebook's reported median RTT of 39 ms [136].

Additionally, we evaluate the overhead of $CCLB_7$ in its Functional PoC, comparing it to a vanilla L7-LB without $CCLB_7$ functionality. The results in Table 3.1 show a 1.4% increase in total CPU utilization for both mean and median, and a 2.4% increase for the 99^{th} percentile. This modest rise is negligible compared to the L7-LB's core function of web service provision, accounting for approximately 20% CPU utilization. The alignment of results from both Functional and Performance-Oriented PoCs validates the minimal overheads of handling public-facing traffic with CloudCookie.

3.5 Discussion

More Applications

We have explored CloudCookie's use for flow scheduling and load balancing by utilizing RPT and load information due to their quantifiable nature. CloudCookie is versatile and can be easily integrated with different optimization signals and data center advances. We believe CloudCookie can support a wide range of applications. Here are some examples: (1) CloudCookie is a drop-in replacement for deploying private data center advances with the advantages of much easier deployment and management. For example, employing CloudCookie can free SDN systems from maintaining OOB control-plane channels, and enable more efficient network management. (2) Quality of experience (QoE) or special traffic treatments are often deployed for better user experience [162]. CloudCookie can carry fine-grained user or application demands in flow packets without client modification. (3) Network functions virtualization (NFV) offers more network programmability and enables the shift of network tasks from proprietary to general devices. NFV can embrace CloudCookie's key idea to enable more impactful NF use cases. (4) CloudCookie can be used to hide sensitive data in Internet protocol headers to circumvent auditing, censorship, and sniffing.

Limitations

CloudCookie does not support QUIC/IPv4 networks, as it requires the composite features of both protocols. Unlike IPv6 assigning a subnet of addresses, IPv4 addresses have become increasingly scarce due to the exponential growth in Internet usage, and hence cannot be used to encode CloudCookie.

Additionally, QUIC's `Connection ID` is unsuitable for encoding CloudCookie that

varies from packet to packet, because issuing a new ID incurs overhead and encoding information recognized by external devices is prohibited according to the RFC [79]. Nevertheless, this limitation only affects a portion traffic, preventing it from benefiting from the optimizations brought by CloudCookie.

3.6 Conclusion

Endorsed by the expanding footprints and increasing controllability of data center authorities, we propose CloudCookie, designed to carry flow-optimization signals within Internet protocol headers. This innovation enables private data center advances in public-facing data centers. We develop a system comprising CCLB_7, CCLB_4, and the scheduler-on switch, as the producer and consumers of flow-optimization signals. CCLB_7 gains insights at the application layer. As a credible load source, it infuses CloudCookie with flow scheduling and load balancing signals. Signals are then consumed by the stateless CCLB_4 and scheduler-on switches to achieve better performance for public-facing traffic. Additionally, our design incurs minimal overhead and requires zero client-side modification.

Our evaluation reveals that the combined activation of CCLB_4 and the SRPT flow scheduler further boosts the reduction in the 99^{th} percentile FCT of the majority flows from $2 - 11\times$ to $20\times$, compared to enabling either one individually. Also, CCLB_4, with an accurate, global perspective and future visions empowered by CloudCookie, consistently outperforms its counterparts, while maintaining its stateless nature. This superiority persists even in TCP networks where CloudCookie operates with lower signal resolution. We recognize that our exploration of CloudCookie is in its nascent stages and we remain open to the future potential of this versatile signal carrier, which holds

promise for integrating more data center advances into public-facing traffic.

Chapter 4

Leveraging Cross-Directional Dependency in Realtime Interactive Streaming

From video conferencing and cloud gaming to mixed reality (XR), realtime interactive streaming is revolutionizing digital experiences by enabling instantaneous, two-way interactions with ultra-low latency. This emerging streaming paradigm reshapes how users engage with content, transforming passive viewers into active participants. Recent years witnessed innovative applications, including holographic communication [53, 123, 100, 131, 4], digital twins [141, 111], turning sci-fi into reality. However, these applications require high-performance rather than consumer-grade networks, putting us one step away from their accessibility. In this project, we will characterize realtime interactive streaming and present an approach tailored to its unique nature.

Unfortunately, the existing network ecosystem does not suit realtime interactive streaming. Figure 4.1 illustrates a holographic communication. It follows the status quo utilizing a Selective Forwarding Unit (SFU) [158] for exchanging data among participants. First, compared to traditional audio/video (A/V) streaming, interactive streaming includes more diverse media types, such as sensor tracking. In addition, different traffic is often associated with distinct characteristics and demands. For example, sensor tracking is low-

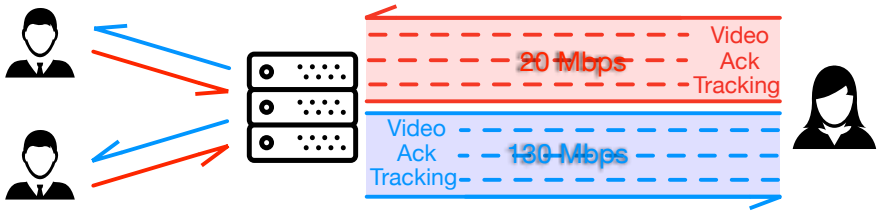


Figure 4.1: Illustration of network traffic patterns in realtime interactive streaming.

	5G [113]	**Fixed Broadband** [35]	**Satellite** [145]
Down	138-238 Mbps	467 Mbps	25-220 Mbps
Up	13-20 Mbps	71 Mbps	5-20 Mbps

Table 4.1: Network bandwidth asymmetry in the United States.

volume, high-frequency, latency sensitive but loss tolerant. This unique aspect prompts us to rethink multiplexing beyond simple A/V streaming.

Second, realtime interactive streaming involves heavy traffic in both directions. While Internet users now have ample bandwidth for bufferless 1080p or 4K streaming, this applies only to the downstream bandwidth, as upstream bandwidth is typically much lower. Recent measurements report such bandwidth asymmetry (Table 4.1) in the United States. Even in an ideal scenario, it is challenging for this upstream bandwidth to support 1080p or 4K live streaming according to the bitrate recommended by YouTube [71] not to mention $360°$, virtual reality (VR), or 3D content.

Third, microbursts [139, 9] exacerbate this situation. Under sufficient bidirectional bandwidth, Figure 4.2 shows the bidirectional throughputs in our tested over a 0.5-second interval while streaming a 30 FPS video, with throughput calculated in 1 ms windows. Rather than being smoothly distributed over time, the throughput fluctuates significantly due to the completion of encoding new video frames, which causes the sender buffer to overfill with a significant amount of packets within a short time. This buffer behavior

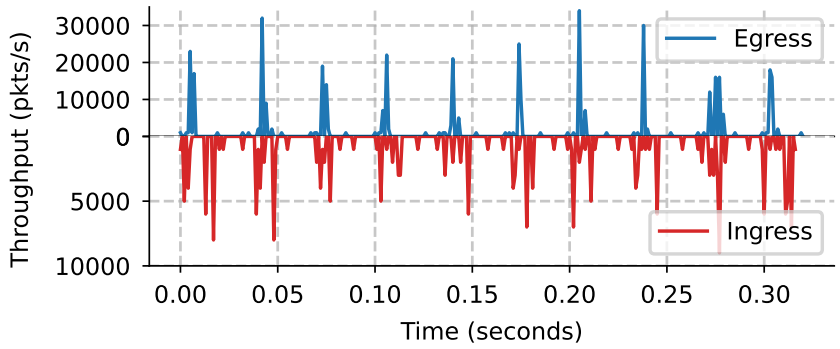


Figure 4.2: Micro-bursty throughputs in both directions of realtime interactive streaming under sufficient bidirectional bandwidth.

results in excessive packet drops and underutilization of egress bandwidth. Without lowering the encoding bitrate, counter measures [9, 85] trying to spread out heavy traffic, however, introduce latencies that hurt the realtime experience. Interestingly, we have already observed a comparable smoothing effect on the ingress throughput. Even in the absence of such countermeasures, network buffers inherently add queuing delays mitigating burstiness.

Last, state-of-the-art realtime streaming optimizations [85, 99, 12, 29, 132, 61, 1, 31, 142] focus on the single direction of the target stream but ignore the correlation of streams across directions, missing opportunities of further optimizations.

Our insights capitalize on the unique characteristic of cross-directional dependency in realtime interactive streaming. Cross-directional dependency is defined as streaming traffic in one direction depending on traffic in the other direction. Specifically, there are two types of cross-directional dependency. The first is cross-directional content dependency. For example, in holographic communication or VR streaming, a common practice is to stream rendered, perspective-projected, *view-dependent* content rather than raw 3D representations or full panoramic 360° videos [11, 3] to save bandwidth. To obtain

view-dependent content, user-side tracking inputs, such as orientations or positions, are required. Consequently, obsolete user-uploaded tracking data can result in falsely processed content being downloaded for playback. The other is cross-directional transmission dependency. For instance, in the scenario depicted in Figure 4.1, when the upstream bandwidth is saturated, a high rate of upstream video retransmissions exacerbates congestion, leading to increased packet loss, including ACKs and tracking data. Excessive upstream ACK loss would not only cause increased downstream throughputs but also introduce playback delays or freezes.

Our proposal builds upon QUIC with the datagram extension [79, 119] by leveraging cross-directional dependency. While existing applications [11, 97] often use multiple connections for different traffic types, such a design leads to self-competition and delegates traffic management to the kernel, limiting application-level optimization. In contrast, QUIC's multiplexing and partial-reliability support allow us to consolidate diverse streams into a single connection with fine-grained bidirectional control. Additionally, partial reliability is essential for realtime streaming, as it allows senders to discard overdue data, e.g., expired video frames, enabling timely and application-aware delivery [128, 119, 117]. The key philosophy of our proposal is to prioritize critical traffic in both directions and turns chaotic congestion into a predictable one. To this end, we extend QUIC's priority system to support datagram prioritization and implement a comprehensive prioritization strategy across unreliable datagrams and reliable streams, aligning transmission behavior with application-level semantics and interaction goals.

Our contributions can be summarized as follows:

- We identify the cross-directional dependency in realtime interactive streaming, introducing a new optimization direction that prioritizes critical traffic in this context.

- We extend the QUIC priority system to support effective datagram prioritization. In addition, we propose a prioritization strategy that aligns with the needs of streaming applications.

- We develop an end-to-end realtime interactive streaming emulation (RISE) framework designed for future-oriented streaming applications featuring diverse media content.

- We extensively evaluate our design in bidirectional view-dependent streaming. The results show significant improvements over the baseline, with up to $9.4\times$ and $82\times$ reductions in motion-to-photon latency and freeze frame rates, respectively.

- We open source the extended QUIC library and the RISE framework to facilitate broader research on next-generation interactive streaming.[1]

4.1 Background

4.1.1 QUIC and QUIC Datagram

QUIC is an emerging transport protocol that provides efficient and secure data delivery [79, 86], serving as the foundation of HTTP/3 [24]. Because it operates in the user space, QUIC offers significant flexibility in transmission control, allowing it to be tailored to application-specific requirements. Additionally, QUIC incorporates several features, including multiplexing, 0-RTT handshake, and mobility support, making it particularly attractive for streaming applications. To date, QUIC has been widely adopted by leading streaming services [82, 91] and content delivery networks (CDNs) [2, 81].

[1]QUIC library: `https://github.com/posoo/quinn-dg-priority`; RISE framework: `https://github.com/posoo/RISE-framework`.

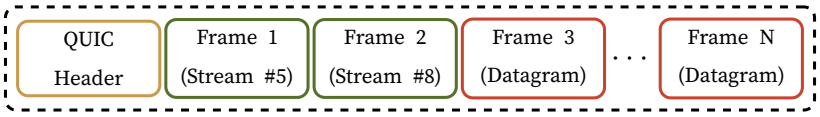


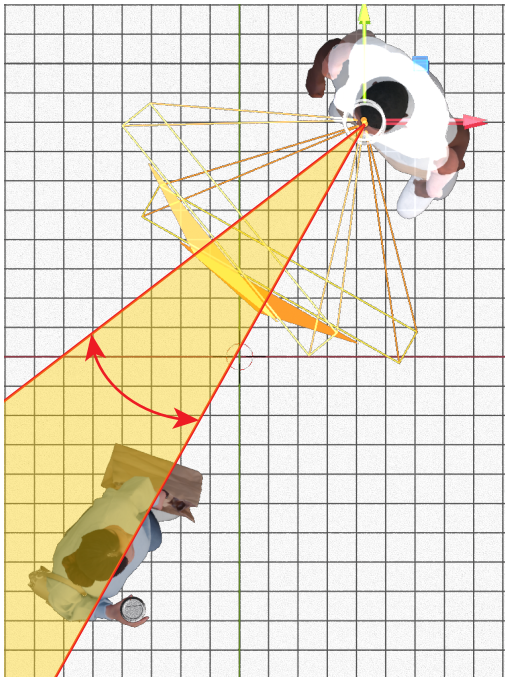
Figure 4.3: The structure of a QUIC packet. Rectangles in green and red refer to *reliable* stream frames and *unreliable* datagram frames. The yellow rectangle denotes the QUIC header.

	MsQuic [102]	**quinn** [126]	**quiche** [34]	**quic-go** [125]	**Chromium** [65]
Stream	✗	✓	✓	✗	✓
Datagram	✗	✗	✗	✗	✗

Table 4.2: Priority supports in the most popular QUIC libraries.

Originally, QUIC is designed as a fully reliable transport protocol to replace TCP and TLS. Recently, an extension introducing unreliable QUIC datagrams [119] has reshaped QUIC into a transport with mixed reliability [128, 117]. This is essential for realtime interactive streaming. First, realtime streaming protocols such as WebRTC [10] and RTP [137] require unreliable transport to enable precise content-aware transmission control. For instance, they may discard outdated content and avoid unnecessary retransmissions. Second, a mixed-reliability scheme facilitates better multiplexing of different types of streaming traffic. Figure 4.3 illustrates a QUIC packet containing both reliable stream frames and unreliable datagram frames. If this packet is lost, the QUIC transmitter has full control to compose a new packet that includes both new data and retransmissions, unlike TCP, which retransmits the entire lost packet.

Although both QUIC streams and datagrams can theoretically be prioritized [79, 119], our survey of the most popular QUIC libraries (Table 4.2) reveals that only a few implement priority control for reliable streams, and none support priority control for unreliable datagrams. Furthermore, there is a lack of discussion on how to implement datagram prioritization in the context of either QUIC or UDP.



(a) Top view

(b) First person PoV before moving

(c) First person PoV after moving

Figure 4.4: Viewport content changes after moving the body.

4.1.2 View-Dependent Streaming

View-dependent streaming has emerged as a critical technique in realtime interactive immersive media delivery systems, addressing the fundamental challenges of computation and bandwidth constraints in XR experiences. On the one hand, user clients, especially mobile devices like VR headsets, typically have limited computational resources. Therefore, a remote server running the application relies on user tracking and inputs to update states, as in cloud gaming [97, 74]. On the other hand, user realtime positional and orientation data can be leveraged to selectively render and transmit only the content within the user's current field of view, substantially reducing the bandwidth requirements compared to full 360-degree or full-scene 3d content streaming [73, 11, 166]. Figure 4.4 exemplifies how the first person point-of-view (PoV) content changes after the viewer rotating his body.

In the context of realtime interactive streaming, the efficiency of view-dependent approaches hinges on ultra-low-latency streaming of user tracking data. Motion-to-photon (MTP) latency is the most direct metric for quantifying this efficiency. Specifically, MTP latency refers to the time elapsed between a user's movement and the corresponding visual update reflecting that movement. Ideally, this latency is supposed be zero, as in the case of local rendering without any processing delay. In realtime interactive streaming systems, MTP latency is considered excellent if it is close to the network round-trip time (RTT). In addition, this efficiency can be quantified at the content level by measuring position mismatch, which results from MTP latency. In a three-degrees-of-freedom (3DoF) system, position mismatch can be defined as the angular difference between the user's current orientation and the orientation used to render the current playback frame. Assuming Figures 4.4b and 4.4c represent these two orientations, the position mismatch is illustrated as the highlighted angle in Figure 4.4a.

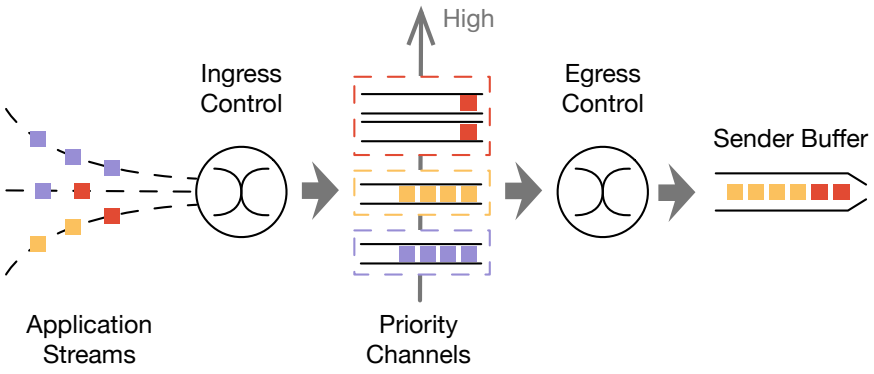


Figure 4.5: The proposed datagram processing pipeline with prioritization in QUIC.

Algorithm 1 QUIC Datagram Priority Control

procedure INGRESSCONTROL(d, $\hat{C}$)
  $p \leftarrow \text{priority}(\hat{C})$ ▷ get the channel priority
  $S \leftarrow \sum_{\forall C} |C|$ ▷ get virtual sender buffer size
  if $S < B$ **then**
    $\hat{C} \leftarrow \hat{C} \cup \{d\}$
  else
    $C_{\text{min}} \leftarrow \arg\min_{\forall C} \text{priority}(C) \mid C \neq \emptyset$
    if $p > \text{priority}(C_{\text{min}})$ **then**
      $\text{dequeue}(P_{\text{min}})$
      $\hat{C} \leftarrow \hat{C} \cup \{d\}$
    else
      $\text{drop}(d)$
procedure EGRESSCONTROL(R)
  $p_{\text{max}} \leftarrow \max\{\text{priority}(C) \mid \forall C, |C| > 0\}$
  $\mathbb{C} \leftarrow \{C \mid \forall C, \text{priority}(C) = p_{\text{max}}, |C| > 0\}$
  if $|\mathbb{C}| = 1$ **then**
    $C' \leftarrow \text{first}(\mathbb{C})$
  else
    $C' \leftarrow \text{RR}(\mathbb{C})$ ▷ round-robin selection
  if $\text{size}(\text{first}(C')) \leq R$ **then**
    return $\text{dequeue}(C')$
  else
    return null

4.2 Design

4.2.1 Priority Control for QUIC Datagram

There is currently no established convention for implementing priority control for unreliable datagrams. In contrast, priority control for reliable streams is well-defined: High-priority stream data is more likely to be included in the next populated packet, while low-priority streams still get delivered eventually in a later stage. However, realtime applications demand unreliable transport to enable flexible retransmission at the application layer.

Our design focuses on establishing datagram priority control in QUIC to deal with streaming microbursts. We propose *priority channels*, which decouple unreliable application streams with specific priorities, enabling flexible and dynamic priority management. Ingress and egress controls work together to enforce priority control. Additionally, ingress control is responsible for capping the size of a *virtual* sender buffer to prevent bloating of the *actual* sender buffer and hence minimize latency. Egress control further helps mitigate starvation by ensuring fair scheduling among channels.

The pipeline of QUIC datagram transmission with priority control is illustrated in Figure 4.5, with the ingress and egress control procedures detailed in Algorithm 1. At the application layer, multiple unreliable streams may request transmitting datagrams with different priorities. A set of priority channels can be initialized either in advance or dynamically on demand. It is worth noting that an application stream may use different channels to send datagrams with varying priorities. For example, a video stream may prioritize key frame datagrams over predicted ones.

Ingress control is the first stage that determines whether to accept or drop an incoming datagram, and where to place it if accepted. We extend QUIC's datagram send API to include the associated priority channel ($\hat{C}$) for each datagram (d). After retrieving the channel priority (p) and the current size of the virtual sender buffer (S), if the buffer size does not exceed the buffer capacity (B), the datagram is immediately enqueued (L4–5). Otherwise, the datagram may still be accepted if its associated channel is not the one with the lowest priority. In that case, the lowest-priority channel will drop its oldest datagram, and the current datagram will be enqueued in the higher-priority channel (L6–10). If the current datagram belongs to the lowest-priority channel, it will be dropped without evicting any existing datagrams (L11–12).

Egress control is triggered when the underlying UDP socket becomes available. The highest-priority datagram that can be accommodated within the next populated QUIC packet is dequeued and moved to the actual sender buffer. Among channels with the same priority, datagrams are dequeued in a round-robin fashion to avoid starvation within equal-priority flows (L13–19). If the remaining space (R) is insufficient to fit the selected datagram, the current egress round is skipped, deferring transmission in anticipation of more available space in the next QUIC packet (L20–23).

While working locally at the sender side, our priority control mechanism not only proactively drops less important datagrams to protect critical ones, but also maintains an ordered priority queue to increase the likelihood of successful delivery of high-priority datagrams in subsequent networks. Importantly, we have no control over sender buffers in the kernel networking stack, network interfaces, or network middleboxes along the path, any of which can become congestion bottlenecks. While the optimal buffer size has long been a subject of debate, both academia and industry generally favor shallow sender buffers under modern Internet protocols and infrastructure [75, 161, 14, 98, 21]. As a result, allowing critical datagrams to enter the buffer earlier increases their chances of surviving subsequent congestion points.

4.2.2 Integration in Realtime Interactive Streaming

Extending QUIC with a datagram priority system enables cross-layer optimization by aligning transport-layer behaviors more closely with application-layer demands. It also extends QUIC's multiplexing concepts to cover QUIC datagrams. To illustrate the integration of this mechanism in realtime interactive streaming, we consider a holographic communication system based on bidirectional view-dependent streaming, as shown in Figure 4.1. In this system, each user continuously uploads tracking data (e.g., orientations),

which is used by the communicating peer to render view-dependent video content, such as perspective-projected 360° video. Simultaneously, each user downloads video data and responds with acknowledgments (ACKs). The process is *symmetric* between interacting users, implying that each user simultaneously acts as both a *streamer* (encoding and sending video) and a *receiver* (sending tracking data and rendering received video). Our optimization goal is to minimize the MTP latency, which is the time between a tracking input and the corresponding video frame being displayed, and frame freezing.

Before diving into the analysis, we define the video encoding frequency as f_v, the motion sensor tracking frequency as f_t, and the round-trip time (RTT) between two interacting users as 2τ. In practice, $f_t \geq f_v$ is required to ensure smooth playback responsive to user movements. For the ease of explanation, we assume $f_t = 2f_v = \frac{1}{\tau}$ and zero processing delay. [2] In our proposed system, QUIC datagrams carry all three types of traffic, with tracking and video data treated as critical traffic.

Without datagram prioritization, the sender buffer behaves as a FIFO queue. Assuming the buffer has infinite capacity, the expected MTP latency is the sum of the RTT and the expected queuing delay for a tracking input (derivation omitted for brevity):

$$\mathbb{E}[L_{\text{MTP}}|\text{FIFO}] = 2\tau + \frac{R_v^2}{2B^2 f_v} \tag{4.1}$$

where R_v denotes the video bitrate, B the available bandwidth, and f_v the video frame rate. For example, streaming a 30 FPS video with $R_v = \frac{2}{3}B$ and $\tau = 10$ ms yields an expected MTP latency of 27 ms. However, an infinite buffer is impractical in real-world

[2] For instance, video conferencing applications like Zoom [148] typically encode video at 30 frames per second (FPS). Devices such as the Meta Quest 2 headset have a tracking frequency around 60 Hz. The median RTT values in the United States are approximately 28 ms for cellular and 13 ms for fixed broadband users [143].

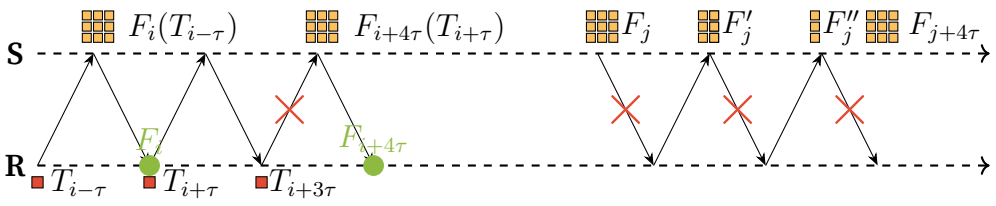


Figure 4.6: Using QUIC datagram prioritization to solve realtime interactive streaming challenges. **S** and **R** stand for Streamer and Receiver respectively. Green dots indicate playback events.

scenarios, as it increases latency for all types of traffic. Additionally, due to the nature of video compression, frame sizes vary significantly, even under constant bitrate (CBR) encoding. For instance, a key frame can be up to two orders of magnitude larger than a predicted frame, increasing the burstiness. Furthermore, realtime streaming favors fresh data, and hence a bounded shallow buffer is preferred.

Packet loss is a major culprit behind excessive MTP latency. As illustrated in Figure 4.6, at time $t = i$, the streamer generates and encodes a frame F_i based on tracking data received τ earlier. In this ideal case, at playback time $t = i + \tau$, the MTP latency is 2τ. Although the next tracking data ($T_{i+\tau}$) is successfully delivered, the subsequent tracking data ($T_{i+3\tau}$) is lost. As a consequence, frame $F_{i+4\tau}$ must use outdated tracking data ($T_{i+\tau}$), resulting in mismatched playback at $t = i + 5\tau$ and doubling the MTP latency to 4τ.

Similarly, failing to prioritize video datagrams can trigger a cascading effect that severely degrades streaming performance. Figure 4.6 depicts a scenario where the streamer encodes and transmits a frame F_j, which experiences partial loss during transmission. After one RTT, the streamer retransmits the lost data (F'_j), further increasing bandwidth usage and may exacerbating network congestion. The situation worsens if retransmitted data (F'_j) is also lost. Consequently, at time $t = j + 4\tau$, the streamer must simultaneously send both a new frame ($F_{j+4\tau}$) and retransmitted data (F''_j), intensifying network congestion. In addition, failure to decode frame F_j in a timely manner results in frame

freezing. Furthermore, if the streamer decides to abandon frame F_j, then frame $F_{j+4\tau}$ must be encoded as a key frame to prevent error propagation, significantly increasing traffic consumption.

Based on the aforementioned insights, we assign the highest priority to tracking data, followed by video data, and the lowest to ACKs. Tracking is prioritized over video due to its extreme latency sensitivity and relatively small traffic volume. Reversing this order would risk starving tracking packets. Although ACKs are also low in volume, they are less latency-sensitive and can therefore be safely deprioritized. Counterintuitively, deprioritizing ACKs can actually improve streaming performance. Our prioritization orders datagrams by traffic type, taming congestion into a predictable process. Instead of being dropped randomly during buffer overflows, ACKs are dequeued right after bursty video traffic is drained, giving them a higher chance of successful delivery. While delayed ACKs may trigger occasional key frame insertions, they reduce the likelihood of frame freezing. Notably, this prioritization remains effective beyond the sender buffer, ensuring a higher success rate of delivery in downstream network buffers (§4.2.1).

4.3 Evaluation

4.3.1 Evaluation Setup

Testbed

Realtime interactive streaming is an emerging streaming paradigm, yet, to the best of our knowledge, the research community currently lacks a standardized testbed for exploring technological advancements in this field. Traditional applications, such as video conferencing and cloud gaming, typically do not exhibit symmetric network demands

or diverse parallel streams. Advanced applications, including holographic communication and digital twins, mostly remain experimental or proprietary due to commercial considerations, limiting open research collaboration. To bridge this gap, we develop the **R**ealtime **I**nteractive **S**treaming **E**mulation (RISE) framework, an end-to-end experimental testbed. RISE is a high-performance asynchronous streaming application implemented in Rust. It implements a research-oriented WebRTC protocol focusing on the streaming logic introduced by ringmaster [103], a streaming platform widely used in prior research [132, 55, 159]. Furthermore, RISE features bidirectional streaming, parallel streams, VR perspective projection, tracking replay capabilities, and multiple transport protocols. RISE is open-sourced, along with our QUIC extension implementation based on quinn [126], aiming to promote broader research efforts in advancing next-generation realtime interactive streaming technologies.

Settings

We deploy RISE endpoints on two servers to simulate realtime interactive streaming peers. Each endpoint simultaneously acts as both the streamer (encoding and sending video frames) and the receiver (sending tracking data, receiving, decoding, and playing video frames). For clarity, we refer to these endpoints as "Host" and "Guest" in the following sections.

The Host endpoint operates in a commercial data center, emulating a user connected behind a SFU server, while the Guest endpoint runs in a campus network environment, simulating a typical user connecting to the SFU. Both servers are equipped with a single AMD EPYC CPU with 8+ cores operating at 3+ GHz, and at least 16 GB of RAM. According to our measurements, this configuration adequately supports perspective projection and encoding of 4K H.264 video streams at 30 FPS, ensuring that the bottleneck for

streaming is in the network. The round-trip time (RTT) between the two servers is 21 ms. We use Linux tc-htb [45] to throttle the Guest's upstream bandwidth, simulating realistic 5G/satellite (20 Mbps) and fixed broadband (70 Mbps) network conditions, as outlined in Table 4.1.

For generating the video stream, we utilize a 4K 360-degree VR gaming video from prior research [156]. Additionally, we synthesize three-degree-of-freedom (3DoF) user movement tracking data at 60 Hz under three different speeds: slow, moderate, and fast. When a new video frame becomes available, each RISE endpoint performs perspective projection using the peer's most recent tracking information with a point-of-view (PoV) of 1 radian and subsequently encodes the projected frames at different bitrates. Specifically, videos are encoded at 10, 30, and 40 Mbps for three settings: 5G/FHD/Low, FB/4K/Low, and FB/4K/High. Here, FHD and 4K denote 1080P and 2160P; Low and High indicate frame rates of 30 and 60 FPS. Each frame has a 1-second deadline. Bitrates are chosen based on YouTube's recommendations [71]. The encoder configuration follows WebRTC standards.

In summary, we primarily evaluate two transport schemes: our design with prioritization (w/ Pri) and baseline QUIC without prioritization (w/o Pri) under these three settings. Unless otherwise noted, results are reported from the Guest's perspective. We focus on QUIC to isolate the effect of datagram prioritization, as SCTP and WebRTC prioritization lack native datagram support and preemptive control, leaving comparative evaluation for future work.

Forward error correction (FEC), error concealment, and foveated rendering features are disabled to minimize unrelated variables. Accordingly, the decoder follows an all-or-nothing best-effort policy, decoding a frame only when all its packets are received and skipping incomplete frames to sustain realtime playback. Our primary baseline for

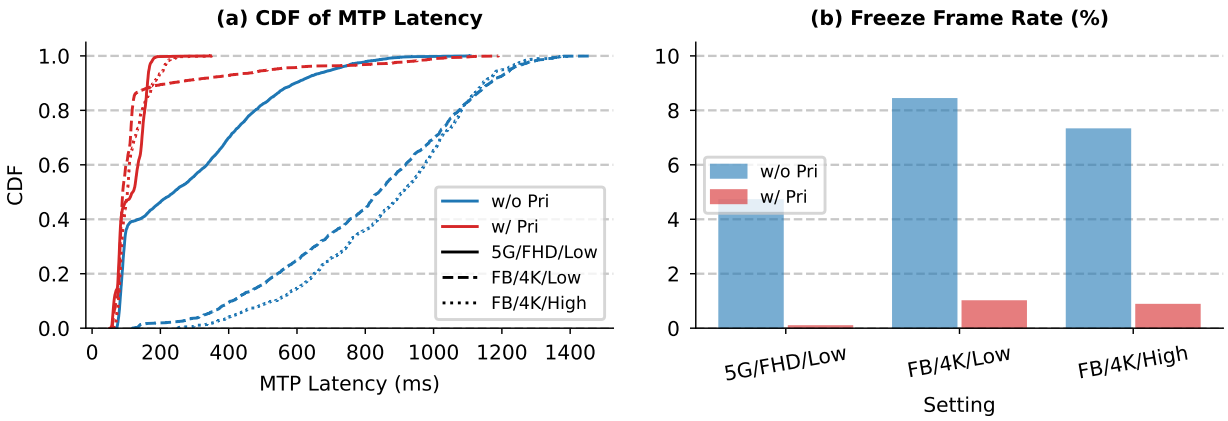


Figure 4.7: Transport-level results.

comparison is the vanilla QUIC without datagram prioritization. Each experimental setting is repeated at least five times.

4.3.2 Transport-Level Results

We first evaluate transport-level metrics by comparing our proposed approach against the baseline. By prioritizing tracking traffic, we achieve a median $2\times$ reduction in MTP latency for streaming FHD/Low video over 5G networks. In fixed broadband networks, the median MTP latency is reduced by $9.4\times$ for 4K/Low video and $8.8\times$ for 4K/High video. Specifically, the $9.4\times$ MTP latency improvement translates to a 730 ms reduction, allowing approximately 45 additional position updates at a tracking sensing frequency of 60 Hz. For the 95th and 99th percentile MTP latencies, our prioritization strategy yields reductions ranging from $2.3\times$ to $5.7\times$ and from $1.3\times$ to $5.3\times$, respectively, across the three evaluated scenarios. This significantly lower MTP latency ensures more responsive graphical updates reflecting user movements.

Additionally, video traffic is prioritized to ensure smoother video playback. Playback smoothness is assessed by measuring freeze frame rates, following the definition

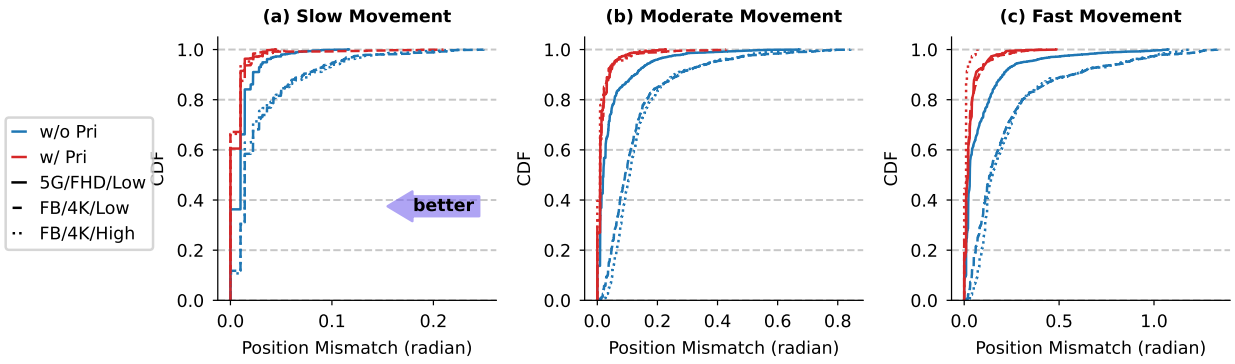


Figure 4.8: CDFs of position mismatch under different movement speeds.

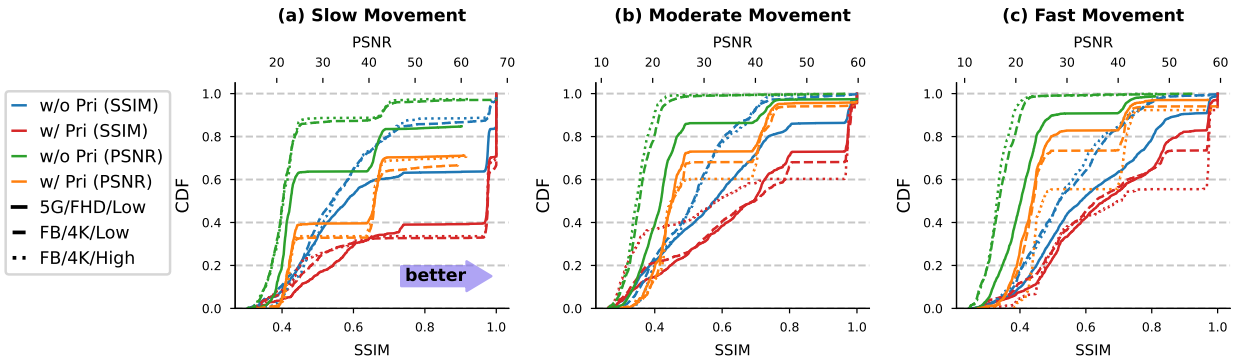


Figure 4.9: CDFs of similarity scores under different movement speeds.

in WebRTC's Statistics API [154]. When streaming FHD/Low video over 5G networks, we observe an average $82\times$ reduction in freeze frame rate. For the other two scenarios over fixed broadband networks, the reductions exceed $8\times$ on average. Across all tested conditions, prioritizing video traffic consistently maintains average freeze frame rates below 1%. These results confirm that assigning second-tier priority to video traffic effectively guarantees smoother video playback experience.

4.3.3 Content-Level Results

We have confirmed that our priority scheme significantly reduces MTP latency. We now examine how this improvement translates into enhanced quality-of-experience (QoE) by evaluating content-level metrics.

We first assess position mismatch, defined in 3DoF systems as the angular difference between user orientations at tracking and decoding time. As shown in Figure 4.8, our scheme achieves zero median mismatch under slow movement and reduces 95th and 99th percentile mismatches by $2.5 - 7.8\times$ and $1.8 - 6.4\times$, respectively. For moderate and fast movement, gains increase further: median reductions reach $2.1 - 11\times$ in 5G networks and $1.5 - 15\times$ in fixed broadband networks; 95th and 99th percentile improvements range from $3 - 7\times$ and $2.5 - 4\times$ for moderate movement, and $2.5 - 28\times$ and $4 - 18\times$ for fast movement. These results underscore a butterfly effect, where small MTP latency differences can lead to amplified content-level discrepancies, even at low movement speeds.

We further evaluate per-frame similarity between decoded and reference videos, where the latter are rendered from ground-truth user positions at decoding time. We evaluate two similarity metrics: Structural Similarity Index Measure (SSIM) [157] and Peak Signal-to-Noise Ratio (PSNR). Notably, each video pair is temporally synchronized, ensuring that observed differences stem only from perspective projection effects. As shown in Figure 4.9, our scheme improves SSIM by $12\% - 68\%$ in 5G networks and $12\% - 80\%$ in fixed broadband networks, and improves PSNR by $10\% - 80\%$ and $37\% - 93\%$ in the two networks, respectively. However, the improvements diminish at higher movement speeds. This reflects a fundamental limitation of SSIM and PSNR: they are designed to quantify quality degradation of identical content, not perceptual differences arising from changes in perspective. We therefore view position mismatch as a more appropriate QoE metric—but all three metrics validate the effectiveness of prioritization on user experience.

4.3.4 Ablation Study

To gain a deeper understanding of how datagram prioritization leverages cross-directional dependency to optimize realtime interactive streaming, we conduct an ablation study to

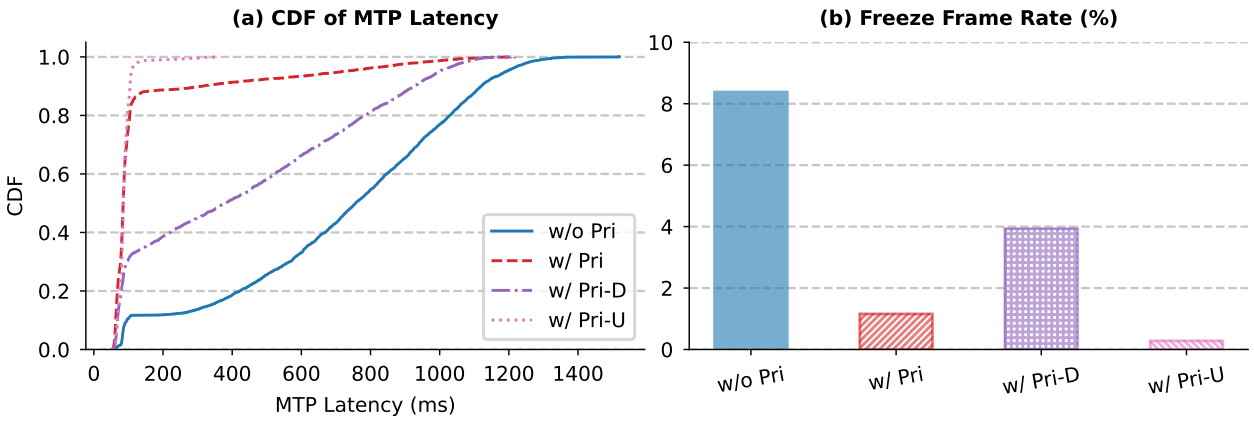


Figure 4.10: Transport-level results under different prioritization directions. "Pri-D", "Pri-U", and "Pri" denote enabling prioritization on downstream, upstream, and both directions.

analyze the impact of prioritization directions and the effective range of prioritization. In this study, we focus on the streaming setting of FB/4K/Low.

Directional Impacts

In addition to duplex prioritization (w/ Pri) and the baseline (w/o Pri), we evaluate two asymmetric schemes: upstream-only (w/ Pri-U) and downstream-only (w/ Pri-D) datagram prioritization.[3] As shown in Figure 4.10, enabling prioritization solely on the upstream, the bottleneck direction, yields the most significant gains, reducing median MTP latency and average freeze frame rate by $9\times$ and $32\times$, respectively. In contrast, downstream-only prioritization achieves $2\times$ and $2.4\times$ reductions. These results confirm that microbursting issues primarily occur in the constrained upstream. Interestingly, downstream prioritization still helps despite the downstream link's ample bandwidth (1 Gbps vs. 30 Mbps demand), indicating that burst-induced buffer overflows can still occur under low bandwidth pressure. However, the mechanisms of improvement differ between upstream and downstream prioritization. To understand this distinction, Figure 4.11 presents retransmission and key frame rates under all schemes.

[3]w/ Pri-U and w/ Pri-D enable prioritization only on the Guest and Host, respectively. These asymmetric schemes are unfair in symmetric setups, evaluated only for ablation.

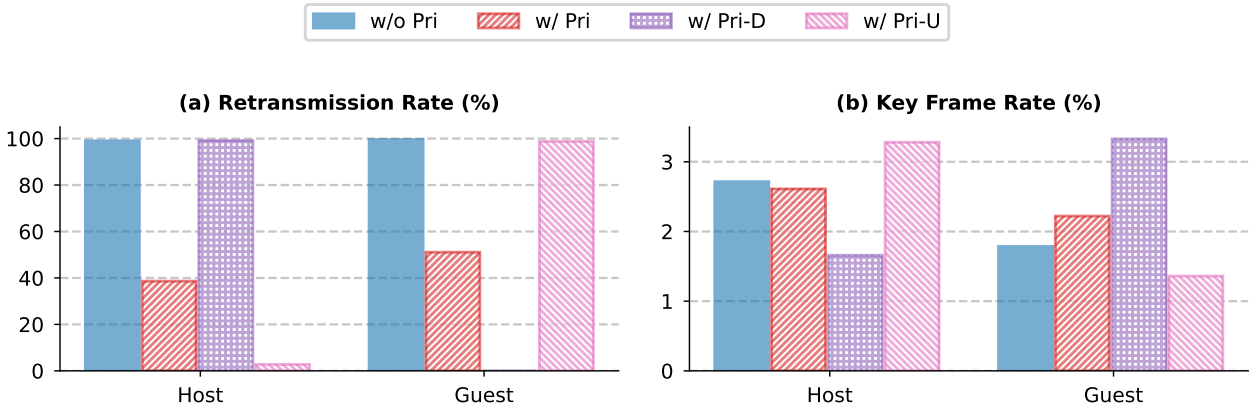


Figure 4.11: Retransmission rates and key frame rates under different prioritization directions.

In the downstream-only case, Host's retransmission rate remains similar to the baseline, as retransmission signals (ACKs) from Guest are neither prioritized nor deprioritized. However, Host encodes only half as many key frames, since its video datagrams are prioritized over ACKs and thus delivered more promptly. This also contributes to reduced MTP latency at the Guest. Meanwhile, Guest experiences almost no retransmissions but shows elevated key frame rates. This is because ACKs from Host are deprioritized, leading to frequent loss or delay of retransmission feedback and forcing Guest to encode more key frames. As a result, burstiness on the bandwidth-limited Guest-to-Host link worsens, partially offsetting the gains in MTP latency.

In the upstream-only case, Guest's MTP latency drops directly due to the prioritization of its latency-sensitive tracking data. Freeze frame rates also decrease, as Guest's ACKs are mostly delayed rather than dropped (§4.2.2), This is evidenced by Host's near-zero retransmission rate and similar key frame rate. Although Host's key frame rate slightly increases, Guest's ample downstream bandwidth absorbs the additional traffic from Host.

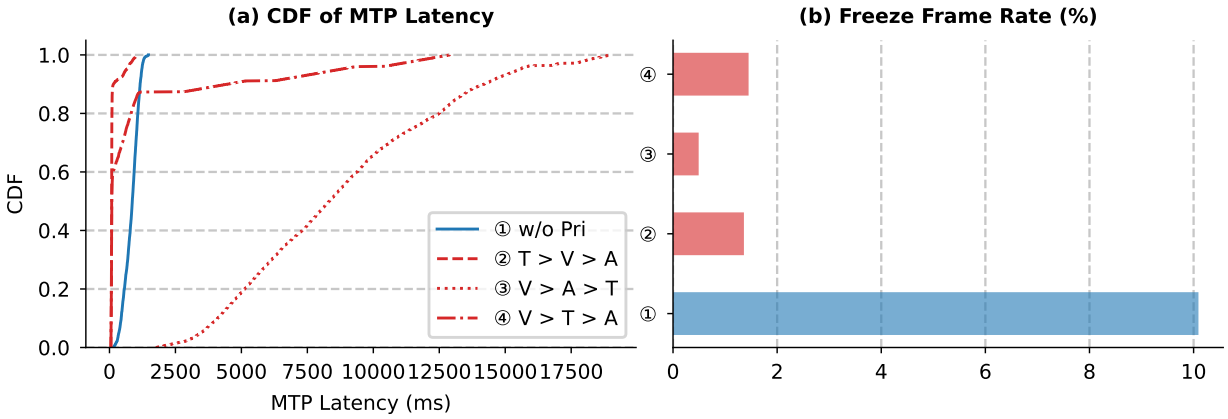


Figure 4.12: Transport-level results under different priority assignments. "T", "V", and "A" refer to Tracking, Video, and ACK datagrams.

Alternative Priority Assignments

In addition to our proposed priority scheme (②: **T**racking > **V**ideo > **A**CK), we evaluate two alternative assignments: ③ (V > A > T) and ④ (V > T > A). Figure 4.12 presents the results of MTP latency and freeze frame rates under these different assignments. From the analysis of directional impacts, we have learned that deprioritizing ACKs implies fewer retransmissions and more abandoned frames for the peer, but it has minimal impact on the streaming direction with sufficient bandwidth. On the other hand, realtime interactive streaming can benefit significantly from prioritizing video traffic for its delivery efficiency improvements during micro-bursts. Results of ③ further validate this claim, enlarging the freeze frame rate reduction to $21\times$ — approximately twice the improvement observed in ②. Clearly, ③ is not a practical choice due to its bloating MTP latency.

Priority assignment ④ explores whether video or tracking traffic should be given higher priority. ② and ④ yield similar reductions in freeze frame rates, but ④ performs significantly worse in minimizing MTP latency. While ④'s median MTP latency is similar to that of ②, its 95th and 99th percentile MTP latencies are even $10\times$ and $7\times$ higher than the baseline (①), respectively. Because tracking data has a much smaller volume than

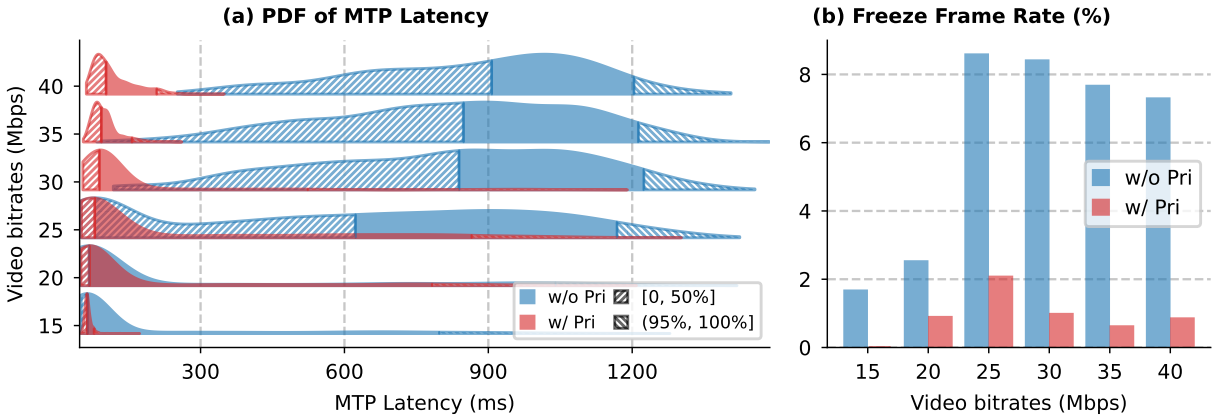


Figure 4.13: Transport-level results under different video encoding bitrates.

video, prioritizing it over video has little effect on video streaming, whereas the reverse significantly harms the delivery of latency-sensitive tracking data. These findings from the alternative priority assignments ultimately justify the effectiveness of our proposed priority assignment (②).

Effective Range

In fixed broadband networks (70 Mbps upstream bandwidth), we vary the video encoding bitrates from 15 Mbps to 40 Mbps to investigate the effective range in which datagram prioritization is able to improve realtime interactive streaming performance. As shown in Figure 4.13, we observe a growing trend in MTP latency differences between QUIC with and without datagram prioritization, as the supply-demand bandwidth gap increases. When video bitrates exceed 25 Mbps, the benefits of prioritization become more pronounced. Specifically, MTP latency improvements range from $7\times$ to $9\times$ in the median and from $1.3\times$ to $10\times$ in the 95th percentile. Average freeze frame rate reductions range from $4\times$ to $12\times$. Additionally, among all tested bitrates, datagram prioritization consistently maintains the median MTP latency below 103 ms and average freeze frame rates below 2%, highlighting its robustness under varying loads. Moreover, even at extremely low bitrates, datagram prioritization effectively mitigates micro-bursting issues.

For instance, at 15 Mbps, it yields a $10\times$ speedup in the 95th percentile MTP latency. This underscores the necessity of enabling datagram prioritization, as merely lowering the encoding bitrate, with reduced video quality, cannot effectively mitigate burstiness.

4.4 Discussion

Limitations

In the current design, deprioritizing ACKs may cause the peer to frequently skip frames due to a lack of retransmissions. The situation can be eased with several realtime streaming best practices. First, forward error correction (FEC) can be employed to encode video datagrams with parity information, reducing reliance on retransmissions. Second, a dynamic priority ladder can be introduced to elevate the priority of retransmitted video datagrams, due to their tighter deadlines, and key frame datagrams, due to their greater impact on playback continuity. We believe that these techniques can further reduce freeze frame rates. Conversely, we intentionally avoid using NACKs to replace ACKs. On the one hand, if NACKs are not prioritized, retransmissions become less likely to happen. On the other hand, prioritizing NACKs would increase the likelihood of dropping video datagrams. Therefore, using NACKs does not provide a clear advantage in this context.

Reactions in Network Buffers

We have extended the QUIC priority system to support datagram prioritization. However, subsequent network buffers, including sender buffers in the kernel networking stack, network interfaces, and ingress queues in network middleboxes, remain beyond the control. These buffers may still experience packet reordering or even overflow caused by micro-bursts, as noted in the introduction. Our prioritization system already introduces a degree of prioritization into these buffers, as outgoing packets are emitted in an or-

der determined by their datagram priorities, providing priority-aware scheduling before transmission. To further reinforce this effect, existing prioritization primitives in these network buffers, such as DiffServ bits or more advanced cross-layer coordination techniques [92], can be leveraged to propagate and enforce packet priorities along the entire network path, ensuring consistent prioritization across buffers and maximizing the benefits of cross-directional dependency in realtime interactive streaming.

4.5 Related Work

Prioritization Support

At the transport layer, both SCTP [146] and QUIC [79] support partial reliability and priority control, but their mechanisms differ. SCTP implements timed reliability, where a lifetime parameter can be attached to a stream to discard overdue (re)transmissions [128]. Additionally, the latest SCTP extensions include a priority-based scheduler under this context [147]. However, SCTP's communication model remains stream-based and lacks native datagram support, making it less suitable for realtime applications. As a result, WebRTC [10] adopts SCTP primarily for data channels rather than media channels. The priority control in QUIC is well-defined and implemented for its reliable streams, but not for its unreliable datagrams (§4.1.1).

At the application layer, WebRTC exposes a four-level prioritization API, which can influence delivery via DSCP marking or proportional bandwidth allocation at the sender [10, 155]. However, these mechanisms lack fine-grained and preemptive control. In addition, DSCP marking is known to be fragile on the Internet [92].

Realtime Streaming Optimization

Forward Error Correction (FEC) has long been a key focus in realtime streaming optimization. Recent advances include adaptive redundancy schemes, such as FRACTaL [85] and Hairpin [99]; learning-based approaches, such as Tooth [12] and RL-AFEC [29]; and advanced coding designs, such as Tambur [132]. Additionally, bandwidth demands can be reduced through network-aware codecs, such as Salsify [61]; adaptive bitrate control, such as Mowgli [1]; and neural codecs, such as GRACE [31] and Gemino [142]. However, these techniques primarily target traditional one-way streaming, where the server pushes video content to a passive client. Recent efforts have begun exploring the realtime interactive streaming space. For example, LoopTailor [88] optimizes the VSync pipeline in cloud gaming. Our work complements these efforts by leveraging cross-directional dependency.

4.6 Conclusion

In this project, we identify the cross-directional dependency unique to the emerging paradigm of realtime interactive streaming. This insight leads us to a new direction for optimizing user experience by prioritizing critical traffic in both directions, thereby mitigating micro-bursting issues without introducing additional latency. To achieve this, we extend the QUIC priority system to support datagram prioritization, advancing QUIC multiplexing to a new level. Additionally, we develop RISE, an end-to-end testbed tailored for realtime interactive streaming, which features diverse bidirectional traffic patterns, and essential functionality for evaluating this emerging streaming paradigm. We conduct extensive evaluations on the Internet with RISE across various configurations. Results reveal up to a $9.4\times$ speedup in MTP latency and an $82\times$ reduction in freeze frame rates. Even under low bandwidth pressure, our prioritization strategy consistently maintains its superiority

against the vanilla QUIC. Future work will integrate additional streaming advances, such as FEC and priority ladder mechanisms, to further leverage the cross-directional dependency for more demanding realtime interactive streaming applications.

Chapter 5

Virtual Multipath Transport: Ubiquitous Multipath Without Multihoming

5.1 Introduction

Multipath transport protocols, such as Multipath TCP (MPTCP) [59], which allow a single connection to simultaneously utilize multiple paths between endpoints, represent a major advancement in Internet communications. By exploiting path diversity, multipath transport offers several key benefits. First, it aggregates bandwidth across multiple paths, thereby raising the achievable throughput of a single connection. Second, it enables connection mobility: as long as one subflow remains active, the overall connection persists. For example, when a smartphone moves out of Wi-Fi coverage, an MPTCP connection can seamlessly continue over cellular, whereas a traditional TCP connection would terminate. Third, it enhances resilience by automatically redistributing traffic to other available paths when one path experiences congestion or channel degradation, with the latter being particularly common in wireless networks. Finally, distributing traffic across multiple paths improves security by reducing the extent of observation possible on any single path. Over the past decade, MPTCP has been standardized [58, 59], integrated into major operating systems, including Linux and Darwin (macOS and iOS), and adopted in widely

used services, such as Apple Siri [153]. In parallel, multipath transport has spurred significant research efforts, resulting in emerging protocols such as MPQUIC [96], XLINK [165], and CellFusion [107].

Since the concept of multipath transport was first proposed, the establishment of multiple paths has been tightly coupled with multihoming, which narrows its applicability and brings certain deployment challenges. In principle, converting a TCP socket to an MPTCP one is as simple as changing the socket declaration, as the kernel transparently falls back to TCP when MPTCP probing fails on either endpoint. However, both clients and servers lack sufficient motivation to enable MPTCP. On the client side, the primary obstacle lies in the multihoming assumption, which presumes that multiple interfaces can concurrently reach the peer, a condition that often does not hold in practice. While smartphones equipped with both Wi-Fi and cellular interfaces satisfy this assumption, other mobile and stationary devices such as laptops, desktops, and IoT terminals typically do not. Counterintuitively, even smartphones exhibit limited support for MPTCP: iOS exposes MPTCP only through its high-level URLSession API with special entitlement [44], restricting its use to HTTP traffic only, while Android takes an even more conservative stance by disabling MPTCP entirely despite its underlying Linux kernel having implemented it. This conservatism largely stems from well-recognized concerns. First, maintaining concurrent Wi-Fi and cellular connectivity increases power consumption [108, 133]. Second, diverting traffic over the cellular network may incur additional data charges [41]. Third, exposing multiple user-identifiable IP addresses to the server introduces privacy and security risks [134, 18].

Due to these barriers, MPTCP adoption on the client side remains minimal, which in turn discourages server-side deployment. A recent Internet-wide measurement study [17,

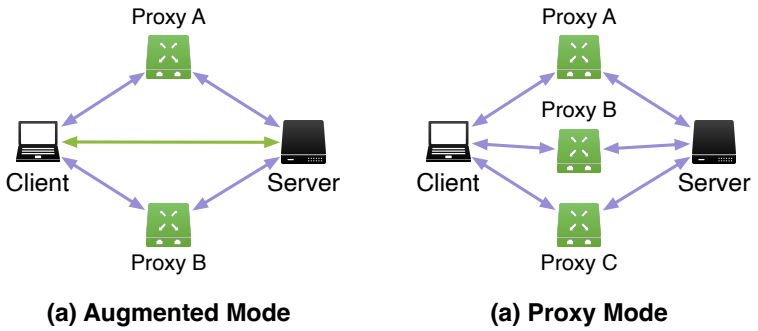


Figure 5.1: Two operational modes of Virtual Multipath. Purple and green lines represent virtual and direct paths, respectively.

140] found around 0.024% of responsive IPv4 hosts supported MPTCP as of December 2021.[1] Moreover, MPTCP traffic accounted for merely 0.002% and 0.04% of total TCP bytes observed on a Tier-1 ISP backbone in North America and a university uplink in Japan as of 2019 and 2020 [17]. In our own investigation, we found that popular cloud providers, including AWS, Azure, and Cloudflare, do not enable MPTCP for their major services such as object storage. The low activation rate on both clients and servers, together with the wide gap between MPTCP capability and real-world usage, underscores the difficulty of adopting multipath transport protocols. This raises a fundamental question: *Can we overcome these limitations and make multipath transport ubiquitous, given its well-recognized benefits?*

We observe that multipath connectivity is, in fact, more ubiquitous than multihoming in today's Internet ecosystem. The Virtual Private Network (VPN) industry has grown rapidly, driven by rising demand for privacy, remote access, censorship circumvention, and network neutrality protection. As of 2024, approximately 46% of Americans report using VPN services [36]. Commercial VPNs typically grant users access to a large set of proxy nodes. For example, leading providers such as ExpressVPN and NordVPN operate

[1] As of December 2021, approximately 13,500 out of 52 million and 16,500 out of 74 million responsive IPv4 hosts supported MPTCPv0, while only about 100–150 hosts supported MPTCPv1 [140].

over 7,400 and 3,000 proxy nodes, respectively, distributed across more than one hundred countries [152]. However, state-of-the-art VPN protocols—including WireGuard [47], IKEv2 [83], and OpenVPN [121]—fail to exploit the full potential of these distributed infrastructures. A client can either select a single proxy node for a connection or distribute different connections across multiple nodes [54, 149], but cannot leverage multiple nodes concurrently within a single connection. Our key insight is that these proxy nodes, whose primary role is to relay traffic, can serve as ideal anchors for establishing multiple network paths, without relying on multiple physical interfaces. More importantly, creating such virtual multiple paths does not trigger the aforementioned concerns regarding power consumption, data charges, or privacy risks. Building on this insight, we propose *Virtual Multipath*, a technique that enables multipath transport capabilities for virtually all connected devices on the Internet.

Virtual Multipath operates in two modes: augmented mode and proxy mode, as illustrated in Figure 5.1. In augmented mode, Virtual Multipath functions similarly to traditional multipath transport protocols, where a single connection aggregates both the direct-link bandwidth and additional bandwidth available through proxy nodes. The virtual paths offer several advantages. First, they aggregate bandwidth across multiple paths, thus improving the achievable throughput of a single connection. This is particularly beneficial in commodity networks that lack dedicated or optimized routes to the communication peer. Second, Virtual Multipath inherits the resilience of traditional multipath transport by automatically redistributing traffic among available paths when one becomes congested or experiences degradation. Third, because a virtual path is always available on the client side, connection mobility is preserved even when the physical interface switches to a new access network. Finally, enhanced security and privacy are achieved by dispersing traffic across multiple proxy paths [42, 19], combining the bene-

fits of multipath transport and secure VPN tunneling. In proxy mode, Virtual Multipath operates more like a conventional VPN service, where all traffic of a single connection is relayed through multiple proxy nodes, completely concealing the client's network footprint.

Virtual Multipath is not a "reinvention of the wheel" but rather an artful reuse of existing kernel functionality through strategic "trickery". We exemplify this idea with virtual MPTCP (VMPTCP). In VMPTCP, a set of virtual network interfaces are artificially instantiated and registered to convince the kernel that the conditions for establishing an MPTCP connection are satisfied. From the kernel's perspective, it can then initiate MPTCP connections and naturally distribute traffic across these interfaces using its existing, well-optimized multipath mechanisms. Specifically, we propose two designs: VMPTCP-Native (VMPTCP-N) and VMPTCP-Tunneling (VMPTCP-T). VMPTCP-N repurposes obsolete or underutilized TCP header fields to encode control metadata and employs an eBPF [62] program that intercepts and manipulates outbound packets. The program performs two key tasks: (1) rewriting packet addresses to reflect the selected proxy node while embedding the metadata, and (2) reassigning packets to the physical network interface for transmission. Inbound packets undergo the reverse transformation. VMPTCP-N introduces negligible overhead and thus achieves the near-native performance. In contrast, VMPTCP-T encapsulates MPTCP packets within IP-over-UDP tunnels between the client and proxy nodes, thereby enabling connection mobility across a broader range of network transitions. All operations in both designs occur either before or after the kernel's networking stack processing, ensuring that the internal multipath logic remains unmodified. From the proxy's viewpoint, VMPTCP packets require only lightweight address rewriting and no additional memory. From the network's viewpoint, VMPTCP traffic is indeed TCP or UDP traffic and is therefore readily deployable across the Internet. From

the server's viewpoint, VMPTCP traffic appears indistinguishable from standard MPTCP traffic, allowing existing MPTCP servers to support VMPTCP transparently and without modification.

We summarize the contributions of this project as follows:

- We reimagine multipath transport protocols beyond multihoming, establishing virtual paths through widely deployed VPN proxy nodes.
- Virtual multipath makes multipath transport protocols ubiquitous and overcomes concerns of traditional multipath transport protocols.
- Virtual multipath provides aggregated bandwidth, path resilience, connection mobility, and enhanced security.
- We design VMPTCP, reusing existing kernel multipath implementation and compatible with standard MPTCP servers.
- We develop VMPTCP-N and VMPTCP-T and conduct extensive evaluation to demonstrate their effectiveness and efficiency.

5.2 Background

5.2.1 Motivation for Multipath Virtual Paths

Even with a single network access, establishing multiple virtual paths can improve connection performance, when the bottleneck is not at the client's edge. If a TCP connection were perfectly efficient—experiencing no losses and maintaining full window utilization—its maximum throughput would equal the bandwidth–delay product (BDP) divided

by the RTT [28], where the BDP reflects the in-flight portion of the sender's window:

$$\text{Throughput}_{\text{max}} = \frac{\text{TCP Sender Window}}{\text{RTT}} \tag{5.1}$$

In the mainline Linux kernel, the maximum send buffer size is set to 4 MB by default, which yields an ideal upper bound of 640 Mbps for a 50 ms RTT. In practice, losses, jitter, and protocol inefficiencies reduce the achievable throughput considerably. Aggregating multiple virtual paths allows better utilization of available capacity and mitigates single-flow bottlenecks. This is particularly beneficial in commodity networks that lack optimized or dedicated routes—for example, home broadband or cellular uplinks used by typical live streamers. By contrast, professional broadcasters often rely on managed or specialized networks (e.g., satellite links, private backhaul, or overlay infrastructures) to achieve higher and more predictable upstream performance[51].

Furthermore, a single virtual path or VPN tunnel rarely remains optimal over time. Network performance fluctuates due to congestion, transient failures, and provider-driven route changes, often triggered by high-frequency TE in cloud backbones [129]. This volatility forces frequent manual reselection of VPN nodes, introducing delay and operational complexity. By contrast, a multipath architecture inherently provides resilience: subflows over different virtual paths are continuously monitored, and traffic is adaptively redistributed when one path degrades. Virtual multipath transport inherits this resilience from multipath protocols such as MPTCP, automatically adjusting traffic allocation across paths based on real-time performance measurements. This dynamic load balancing not only maintains stable throughput but also masks transient failures, offering stronger robustness for latency-sensitive applications such as live streaming and real-time interaction.

5.2.2 Connection Mobility

One of the most important advantages of multipath transport protocols such as MPTCP is connection mobility: when a subflow fails due to path unavailability, the connection remains intact, and traffic is reassigned to available paths. While such migration is seamless on multihomed devices, MPTCP also supports connection mobility on single-homed devices. However, this requires several additional round trips after reconnecting to a new network access, because the client must probe the new path and establish a new MPTCP subflow. Moreover, the new subflow is inherently a new TCP connection whose sender window grows linearly, requiring extra time to reach its maximum throughput. As a result, MPTCP's connection mobility on single-homed devices is fundamentally limited for latency-sensitive or high-throughput applications.

Connection mobility, however, can also be achieved through mechanisms beyond multipath transport. QUIC, for instance, maintains a set of connection IDs rather than relying on the traditional 5-tuple for connection identification. This design naturally supports migration between different IP addresses, enabling a connection to survive network changes without requiring simultaneous multipath availability. An ongoing standardization effort further extends QUIC to support true multipath capabilities, known as MPQUIC [95]. Whereas QUIC supports mobility natively, porting existing TCP-based applications to QUIC often requires non-trivial client- and server-side refactoring, along with corresponding deployment changes.

Modern VPN protocols also offer a form of connection mobility. WireGuard, one of the most advanced VPNs, creates a virtual network interface on the client host. All outbound traffic is routed through this virtual interface, using its IP address as the source and the proxy's IP as the destination. When the client switches from Wi-Fi to cellular access,

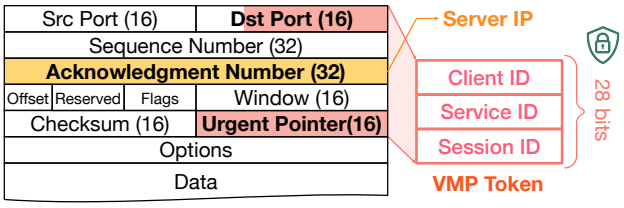


Figure 5.2: In-and proxy signaling through repurposing TCP header fields in vMPTCP-N.

the kernel still observes traffic flowing between the same virtual and proxy addresses, allowing the connection to persist without reset. The WireGuard daemon then encapsulates packets into UDP and transmits them over the current physical interface. While WireGuard provides connection continuity, it does so by relaying all traffic through a single proxy node.

In contrast, Virtual Multipath is designed to maximize both flexibility and availability. Like QUIC and WireGuard, it does not depend on physical multihoming. However, unlike WireGuard, it does not require proxying all traffic. Instead, Virtual Multipath can dynamically activate proxy-enabled paths only when needed—for example, to maintain connectivity during a direct-path outage—and later hand over traffic back to a newly available direct path.

5.3 Native Virtual MPTCP

Virtual MPTCP works by tricking the kernel into initiating and maintaining MPTCP connections, utilizing its native, well-optimized implementation, even on single-homed devices. In contrast to QUIC porting, converting any TCP application into a MPTCP one is much easier. On Linux, creating an MPTCP socket is as simple as specifying the MPTCP protocol in the socket declaration:

```
socket(AF_INET(6), SOCK_STREAM, IPPROTO_MPTCP)
```

However, by default, the kernel initiates an MPTCP connection only when certain multihoming conditions are satisfied: (1) the device possesses multiple network interfaces (NIs); (2) these NIs are registered as MPTCP endpoints; and (3) packets from these interfaces can be routed to the destination server. The first two conditions are straightforward. Linux offers several types of software NIs, such as dummy, TAP, and TUN devices [114, 115], collectively referred to as virtual NIs (VNIs). Once assigned a valid IP address, a VNI can be registered as an MPTCP endpoint. The third condition, however, is more challenging. It requires agreement among all involved parties on how connection packets are forwarded and routed, while remaining compatible with intermediate middleboxes. To address this challenge, we introduce VMPTCP-N, a design that allows virtual MPTCP traffic to be recognized by the kernel and the network as standard MPTCP packets, without tunneling, thus achieving near-native performance.

5.3.1 In-Band Proxy Signaling

MPTCP packets are essentially TCP packets carrying MPTCP-specific options in their headers, and MPTCP subflows are individual TCP connections established over distinct paths. Building on this structure, VMPTCP-N repurposes obsolete or underutilized TCP header fields to encode critical metadata used to establish forwarding and routing logic. To remain fully compatible with the kernel networking stack, we design an in-band proxy signaling mechanism, rather than relying on out-of-band messages (Figure 5.2). This in-band signaling serves two purposes: (1) maintaining the session table on the proxy node and (2) routing packets between the VNIs and the physical NI (PNI) that connects to the Internet.

For packets sent from the client to the proxy node, the destination IP address and port number must be rewritten to point to the proxy node. Consequently, the proxy node

must be notified of the original server's IP address and port number to create a corresponding session table entry. VMPTCP-N encodes the server's IPv4 address into the acknowledgment number field of the initial SYN packet.[2] Because the first SYN packet carries an ACK flag set to zero, the acknowledgment number field will not be evaluated by the TCP stack [46], making this reuse safe.

Besides the server's IP address, VMPTCP-N must also convey the server's service port number and support connection mobility when the client's address changes. To achieve this, we design a *VMP token* that carries three identifiers: the client, the service, and the session. VMPTCP-N encodes this token using a total of 28 bits available within the TCP header. First, it exploits the obsolete urgent pointer field, which provides a 16-bit message space [46]. Because the urgent pointer field has been officially deprecated [84, 49], it can be safely repurposed for encoding partial VMP token.

Additionally, VMPTCP-N identifies another 12-bit coding space from *semantic partitioning* of the destination port number. Originally, the destination port number of packets sent from the client to the proxy node corresponds to the proxy's listening port, serving merely as a flat service identifier. This design underutilizes the 16-bit port space, since each port value corresponds to only one binding and leaves no room for additional semantics. VMPTCP-N repurposes this field by semantically partitioning the port number into three components. The most significant bit is fixed to one, preserving all port numbers below 32768 for existing services on the proxy node. The next 3 bits are used to represent the listening port number, while the remaining 12 bits are utilized to partially encode the VMP token. VMPTCP-N redefines the listening port number as a Proxy Semantic Listening Port (PSLPort), embedding both service and token semantics within the same 16-bit

[2] VMPTCP-N currently supports only IPv4 servers.

field.

The feasibility of semantic partitioning of the port number stems from two conditions being met. First, since the PSLPort remains fixed for each connection—because the VMP token is unchanged throughout the entire lifecycle of an MPTCP subflow, *i.e.,* a TCP connection, the PSLPort remains Network Address Translation (NAT) friendly. Second, packets with the PSLPort are forwarded directly on the proxy node, without being processed by the kernel's TCP stack, so the kernel is not confused by such packets.

The first component of the VMP token is the Client ID, a unique identifier assigned to each client in advance. Similar to other VPN architectures, the Client ID serves as a service contract token between the client and the VPN provider. Although it is intended to remain consistent throughout the client's lifetime, a client may hold multiple Client IDs, and any of them can be revoked by the provider at any time. As a contractual token, it can also facilitate advanced management functions such as differentiated service. Obviously, the Client ID is much shorter than a conventional UUID [39]. For instance, a 20-bit Client ID can represent up to one million clients globally, whereas large VPN providers may serve a few millions of active users. To address this scalability limitation, we consider three practical approaches. First, the provider can allocate additional IP addresses to proxy nodes, effectively replicating the address space per node. Compared to computational and networking costs, IP addresses are inexpensive, the average monthly lease price for an IPv4 address is approximately $0.05 [78]. And when the proxy node runs on IPv6, the marginal cost is negligible. Second, colocated proxy nodes can be partitioned into groups, with each Client ID authorized to access only a subset of proxies at a given site. Third, the 3-bit listening port field can be utilized more aggressively by activating all eight possible listening ports per proxy node, effectively providing the equivalent of eight

duplicated IP addresses per node. In practice, providers may combine all three strategies to maximize scalability.

The other two components of the VMP token are the Service ID and the Session ID. Given the limited coding space, the Service ID serves as a compact substitute for the full service port number, mapping to a predefined set of commonly used ports. This mapping can be customized for each Client ID and is established during the negotiation phase when the Client ID is assigned. The Session ID is designed to decouple the client's ephemeral port number from its MPTCP subflow through the proxy node, enabling connection mobility when the client's port changes behind different NATs. The combination of the Client ID and Service ID alone is insufficient to uniquely identify an MPTCP subflow, as multiple parallel sessions may coexist. In this project, we adopt an empirical bit allocation of 20:4:4 for the Client ID, Service ID, and Session ID, respectively. In practice, providers may choose alternative allocations to balance the trade-off between scalability and per-client flexibility. For security, we recommend encrypting the VMP token using a modern stream cipher that preserves the plaintext length, such as ChaCha20 [109]. The encryption key, along with the mapping between Service IDs and service port numbers, is provisioned together with the Client ID during registration.

5.3.2 VMPTCP-N In Action

VMPTCP-N employs a set of VNIs to create a multihoming illusion that convinces the kernel to initiate MPTCP connections. Next, we detail the mechanisms on both the client and proxy sides. Figure 5.3 illustrates the workflow of bidirectional packet pipelines between the client and the proxy node. Although only one VNI is shown in the figure, in practice VMPTCP-N typically sets up multiple VNIs to establish concurrent tunnels to different proxy nodes, thereby maximizing connection resilience.

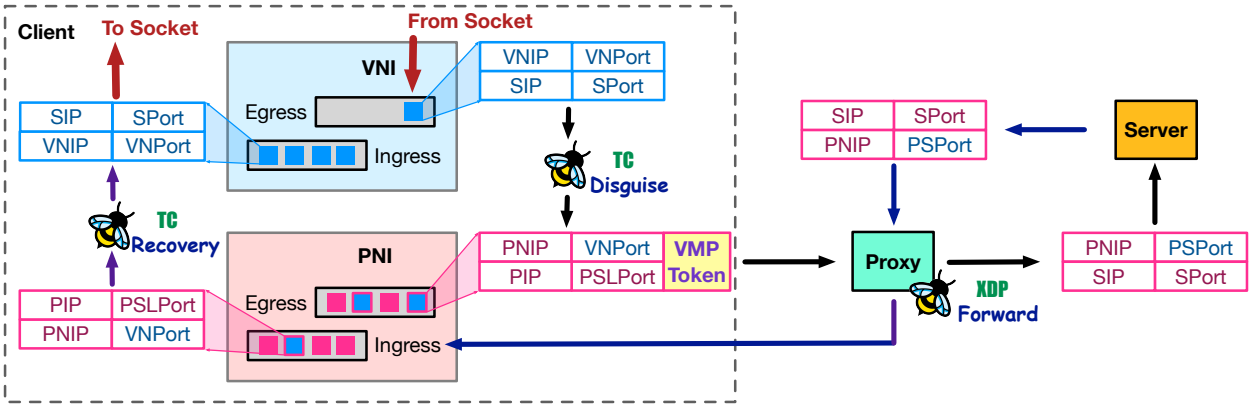
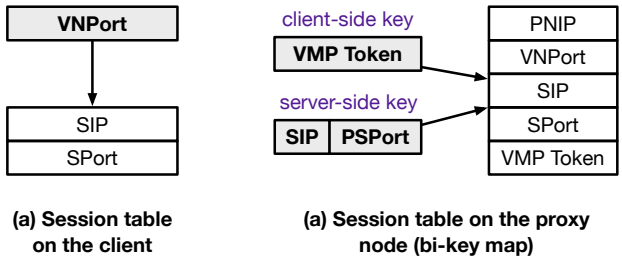


Figure 5.3: Workflow of the VMPTCP-N design, illustrating the bidirectional packet pipelines between the client's VNI and the server. Each packet is represented by its source and destination addresses and ports, while packets sent from the client's PNI to the proxy node additionally carry the VMP token.



(a) Session table on the client

(a) Session table on the proxy node (bi-key map)

Figure 5.4: The session table structure on the client and the proxy node in VMPTCP-N.

On the client side, VMPTCP-N creates multiple dummy interfaces [114], which are originally designed as loopback-like VNIs. By default, a dummy interface does not transmit packets. To enable communication between a VNI and the proxy node, we attach two eBPF Traffic Control (TC) hooks that implement the aforemented in-band proxy signaling technique. When a VNI is registered as an MPTCP subflow endpoint, it initiates a TCP connection starting with a SYN packet that carries either the `MP_CAPABLE` or `MP_JOIN` option, depending on whether it is the first or a subsequent subflow. Each packet placed in the VNI's egress queue is intercepted by an eBPF TC hook. This hook rewrites the packet's source address from the VNI's IP (VNIP) to the PNI's IP (PNIP), and the destination address and port from the server's IP (SIP) and service port (SPort) to the proxy node's IP (PIP) and PSLPort, while attaching the VMP token. For SYN packets, the hook additionally encodes the server's IP address into the acknowledgment number field and creates a new entry in the session table for recovering packets in the reverse direction (Figure 5.4 (a)). Finally, the packet is reassigned to the PNI's egress queue for further transmission. These rewritten packets are thus disguised as routable PNI packets on the PNI's egress queue and are transmitted together with other outbound traffic.

On the proxy node, an eXpress Data Path (XDP) eBPF program processes VMPTCP packets before they enter the kernel's networking stack. Upon receiving a SYN packet, the proxy creates a new session table entry for the corresponding VMPTCP subflow (Figure 5.4 (b)). Notably, this session table differs from that of traditional VPN architectures, as it is implemented as a bi-key map to handle the asymmetric headers of packets received from both directions. Additionally, a proxy-selected port (PSPort) replaces the packet's source port to prevent port conflicts among clients, following practices common in VPN architectures. For subsequent packets of the same subflow, the session entry is retrieved from the session table using the client-side key, which corresponds to the VMP token em-

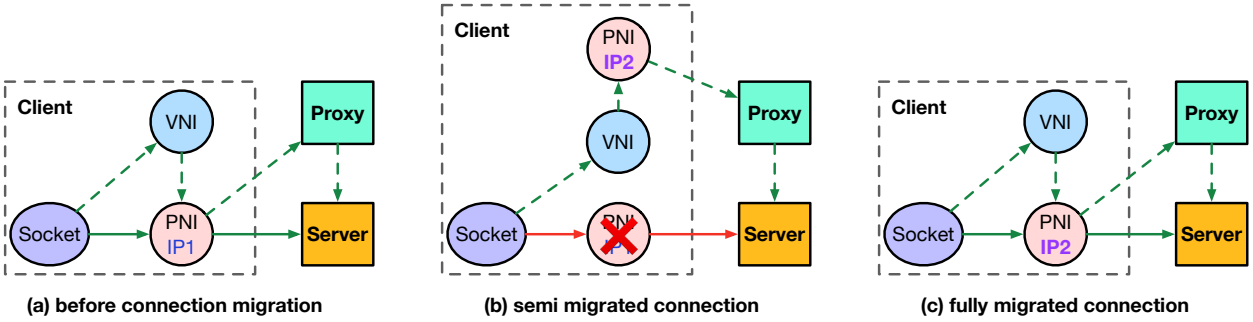


Figure 5.5: Connection mobility of VMPTCP. Solid and dashed lines represent direct-link and virtual paths, respectively. Red and green lines denote available and unavailable paths, respectively.

bedded in the packet. For each packet, the source port is rewritten to the PSPort, and the destination address and port are rewritten to the server's SIP and SPort for onward delivery. From the server's perspective, packets relayed by the proxy appear indistinguishable from standard MPTCP packets originating from the client. Because the design is fully transparent, the server cannot differentiate VMPTCP traffic from conventional MPTCP connections.

The reverse direction operates symmetrically. A VMPTCP packet arriving at the proxy node is identified through a lookup in the session table using the server-side key, and its 5-tuple is rewritten to mirror that of the client-to-proxy packet. This symmetric view ensures that intermediate network middleboxes between the client and proxy recognize the packets as legitimate MPTCP traffic and process them accordingly. When the packet reaches the client's PNI ingress queue, its header is recovered by another TC hook, which rewrites the source address and port back to SIP and SPort, and the destination address and port to VNIP and VNPort. The packet is then enqueued into the VNI's ingress queue and delivered to the socket. From the socket's perspective, this header symmetry is critical to convince the kernel that the VNI legitimately functions as an MPTCP subflow endpoint.

5.4 Connection Mobility with Virtual MPTCP

5.4.1 Mobility Mechanism and Challenges

VMPTCP is designed with connection mobility at its core. Figure 5.5 illustrates the connection migration process. Before migration, multiple subflows are active between the client and the server, including those on virtual path subflows through the VNI and proxy nodes, while traffic distribution depends on the kernel's MPTCP scheduler. When the previous network access is lost, the direct-path subflow becomes immediately unavailable. However, the virtual-path subflows remain active from the kernel's perspective. They are still recognized as valid MPTCP connections. Consequently, the application-layer socket maintains its connection and continues transmission, even though packets sent through the VNI are dropped because the PNI is temporarily disconnected. Once the PNI reconnects to the network, possibly with a different IP address, packets over virtual subflows can resume delivery immediately. We refer to this state as semi-migrated. The new PNI is then registered as an MPTCP subflow endpoint, initiating a new direct-path subflow through the MPTCP handshake. Once the new direct-path subflow is established, the VMPTCP connection is considered fully migrated. Notably, for proxy-mode VMPTCP, connection mobility is simpler because there is no direct-path subflow to reestablish.

On the proxy node, the session table (Figure 5.4 (b)) is designed to be updatable when the client's IP address, *i.e.,* PNIP, changes. In VMPTCP-N, the VMP token enables address updates so that packets belonging to an existing VMPTCP subflow, even with a new source address and port, can be correctly associated with their existing session entries. However, VMPTCP-N supports connection mobility only under limited scenarios due to the complex NAT and strict firewall behaviors of today's Internet. Most NATs allo-

cate a new mapping only upon seeing an outbound SYN packet and retain it in a transitory, *i.e.,* partially-open, state until the TCP three-way handshake completes [60]. This behavior restricts VMPTCP-N's mobility to two practical scenarios. The first occurs when the client's new network assigns a public IP address to its PNI, which is often the case in IPv6 environments. The second arises when the client is located behind a one-to-one NAT or a permissive NAT that does not enforce the above restrictions.

5.4.2 Tunneling Virtual MPTCP

To enable universal connection mobility, we design VMPTCP-T, a tunneling-based variant. VMPTCP-T sets up a set of TUN devices as VNIs for the VMPTCP kernel trickery. Packets from a VNI's egress queue are encapsulated with the VMP token and transmitted through IP-over-UDP tunnels to the proxy node. This approach follows the practice of modern VPN protocols such as WireGuard, but VMPTCP-T actively establishes multiple tunnels in parallel. In addition, VMPTCP-T integrates a path monitor that continuously probes the readiness of the PNI, prevents application-level panics caused by socket errors, and seamlessly migrates the connection to the direct-path once the PNI reconnects. Moreover, unlike the VMPTCP-N client, relying on eBPF programs to intercept and inject packets at the VNI, the VMPTCP-T client employs a user-space daemon to interact with the TUN device.

Since the VMP token is carried in the UDP payload rather than the TCP header, it has no length restriction, offering better scalability and flexibility. Moreover, because Internet NATs and firewalls typically treat UDP packets more permissively and create new mappings for any unseen 5-tuple, VMPTCP-T's design naturally benefits from such robust traversability. This insight resonates with the mobility mechanisms of QUIC and WireGuard.

The proxy node in VMPTCP-T operates slightly differently from that in VMPTCP-N. When it decapsulates a packet received through the UDP tunnel, the proxy node either creates a new session table entry for the corresponding VMPTCP subflow or updates the existing entry if the client's PNIP has changed. The proxy node then sets up a TUN device and configures MASQUERADE Source NAT (SNAT). Encapsulated packets are injected into the TUN's egress queue, where the kernel forwards them to the destination server on the Internet. In the reverse direction, packets received from the server are placed into the TUN's ingress queue with a 5-tuple mirroring that of the outgoing packet. Each packet is then encapsulated with the VMP token and sent back to the client through the same UDP channel.

While VMPTCP-T achieves ubiquitous connection mobility and eliminates the scalability and flexibility constraints of VMPTCP-N, its performance is typically lower for several reasons. First, VMPTCP-T incurs additional overhead, including unavoidable memory copies between kernel and user space, whereas VMPTCP-N manipulates kernel memory directly. Second, frequent UDP socket operations introduce I/O latency and are generally less efficient than TCP sockets. Third, due to the additional space reserved for the VMP token and tunneling headers, the MTU must be reduced, resulting in less efficient channel utilization. It is worth noting that VMPTCP-T and VMPTCP-N are not mutually exclusive. Both can coexist on the same client and proxy nodes, allowing users to flexibly select the design that best fits their needs.

5.5 Evaluation

To validate the concept of virtual multipath transport and evaluate the benefits of VMPTCP, we deployed multiple VMPTCP proxy nodes, as well as MPTCP clients and servers, across

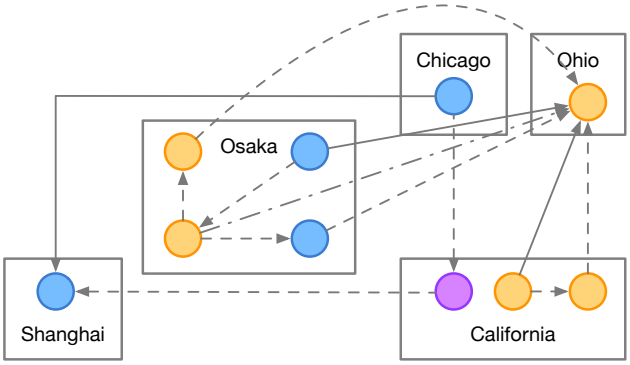


Figure 5.6: Connectivity among testing nodes in performance evaluation. Blue, purple, and yellow nodes represent machines deployed in best-effort public backbones, premium public backbones, and private (AWS) backbones, respectively. While solid and dashed arrows denote direct-path and virtual-path (sub)flows, dashdotted arrows denote flows on both path.

the globe. Specifically, we evaluate three core benefits of virtual multipath transport: connection mobility, resilience to network dynamics, and performance gains. In addition, we examine both VMPTCP-N and VMPTCP-T, corresponding to the augmented and proxy modes, respectively, depending on the evaluation scenario.

All testing nodes run the mainline Linux kernel version 6.17. Both clients and servers use the in-tree MPTCP implementation provided by the kernel. Currently, the in-tree MPTCP only supports a single non-overlapping path scheduler, which is lowest RTT based. And no coupled congestion control is supported, so the MPTCP connection inherits the cubic congestion control algorithm for TCP for its subflows individually. Although each client is single-homed, both physical and virtual NIs are registered as MPTCP subflow endpoints, actively establishing subflows over available paths by sending `MP_JOIN` requests.

5.5.1 Performance Gains

While virtual multipath transport fundamentally differs from traditional multipath transport protocols—which aggregate bandwidth across multiple physical links—it can still achieve significant performance improvements by more effectively saturating the avail-

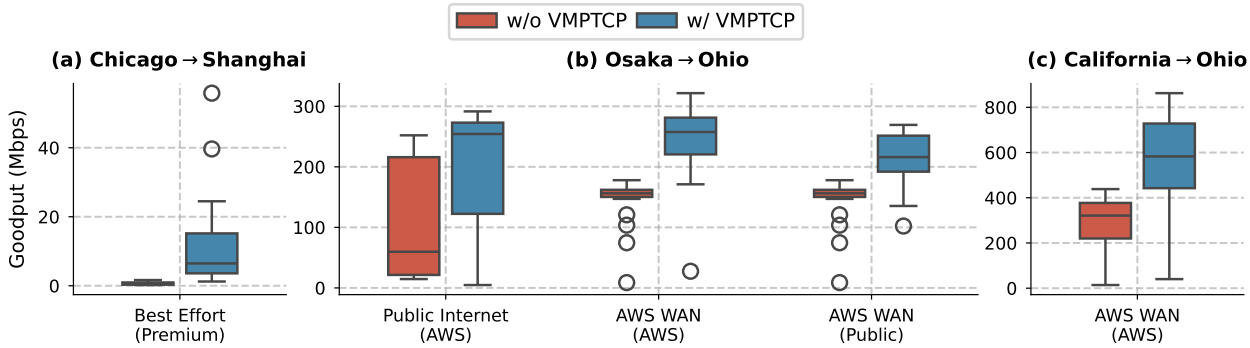


Figure 5.7: Comparison of goodput performance with and without VMPTCP under different scenarios.

able bandwidth of a single network access. To demonstrate this, we deploy VMPTCP-N on multiple clients and proxy nodes worldwide (Figure 5.6), running VMPTCP in the augmented mode with a single virtual path through a proxy. We conduct iperf [3] tests 20 times for each scenario and measure the goodput between clients and servers, both with and without VMPTCP.

The first scenario involves a cross-continent connection between the United States and China, where a virtual path through a premium public backbone is used to accelerate the connection (Figure 5.7(a)). China Telecom operates two major international backbones: ChinaNet (AS4134)[150] and CN2 (AS4809) [151]. While both are public networks, ChinaNet provides best-effort Internet connectivity, whereas CN2 is a premium MPLS-based backbone with QoS guarantees. In this scenario, a client in a residential network in Chicago, USA, transmits traffic to a server hosted in a public cloud in Shanghai, China. The proxy node for the virtual path is deployed in a CN2-connected datacenter in California, USA. Without VMPTCP, a single TCP connection achieves a median goodput of only 0.5 Mbps, making even basic web browsing sluggish. With VMPTCP enabled, the median goodput increases fourteenfold to 7 Mbps, with an average goodput of 12 Mbps.

[3] iperf [52] has supported MPTCP since version 3.19.

The second scenario evaluates a cross-continent connection between the U.S. and Japan using a private cloud backbone as the virtual path (Figure 5.7(b)). Major cloud providers operate extensive global backbones; for example, AWS has deployed over six million kilometers of fiber-optic cables worldwide, enabling faster data transfer and reduced latency[138]. In all cases, the server is deployed in AWS's Ohio datacenter, while clients and proxies are located in different cities in Osaka, Japan.

In the first configuration, the client resides in a public Internet datacenter and the proxy is hosted in AWS. Leveraging AWS's private backbone as the virtual path, VMPTCP achieves a fourfold improvement in median goodput from 60 Mbps to 250 Mbps, and a twofold improvement in average goodput, from 111 Mbps to 180 Mbps. In the second configuration, both client and proxy are deployed within AWS. Although the direct path already traverses AWS's inter-datacenter WAN, without VMPTCP, the connection remains limited by single-path bottlenecks, observing a median goodput of 156 Mbps and an average of 142 Mbps. Enabling VMPTCP increases the median goodput by 60% to 257 Mbps and the average by 70% to 245 Mbps. In the third configuration, the client remains in AWS, but the proxy is placed in a public datacenter. Similarly, VMPTCP improves the median goodput by 38% from 156 Mbps to 216 Mbps, and the average by 50% from 142 Mbps to 214 Mbps.

The performance gains from VMPTCP are not limited to challenging cross-continent connections; they also manifest in domestic settings. We deploy the server in AWS's Ohio datacenter, and both the client and proxy in AWS's California datacenter (Figure 5.7 (c)). The direct connection via AWS's domestic backbone achieves a median goodput of 321 Mbps and an average of 277 Mbps. This single-flow throughput is constrained by kernel defaults that cap the send buffer to 4 MB. After enabling VMPTCP through a colo-

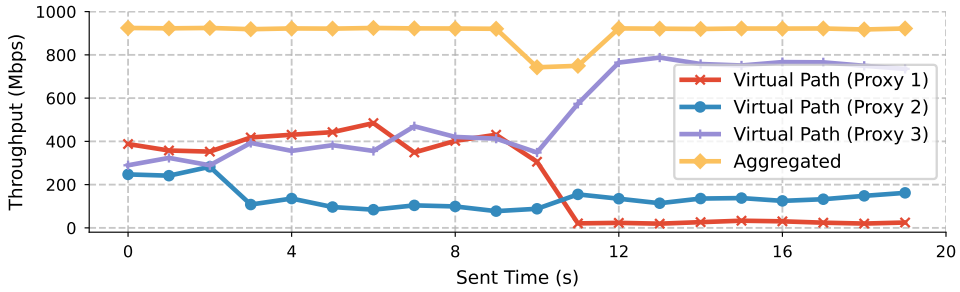


Figure 5.8: Distribution of the throughputs of subflows within a VMPTCP connection adapting to network dynamics.

cated proxy, the median goodput doubles to 582 Mbps, and the average rises to 556 Mbps. While initiating two parallel TCP connections can achieve similar aggregate throughput, VMPTCP enhances a single connection's goodput transparently, benefiting bursty high-throughput applications without requiring application-level traffic splitting.

5.5.2 Resilience to Network Dynamics

In contrast to traditional VPN protocols that bind a connection to a single selected path, virtual multipath transport imposes no restriction on the number of concurrent paths. It is explicitly designed to establish multiple virtual paths simultaneously, maximizing connection resilience and enabling adaptation to network dynamics. In this experiment, we again run iperf workloads, but with VMPTCP operating in proxy mode. Due to the difficulty of emulating network dynamics within an XDP BPF program, we evaluate VMPTCP-T instead of VMPTCP-N. Network dynamics are simulated on the proxy nodes using tc netem. The client connects through three colocated proxy nodes. Figure 5.8 illustrates the throughput distribution of each subflow and the aggregate throughput over time.

At the beginning of the experiment, the three subflows share the traffic roughly equally, resulting from similar RTTs across all paths. After approximately two seconds,

we introduce a 0.2% packet loss on the subflow via Proxy 2. The path scheduler immediately detects this degradation and shifts more traffic to the other two subflows. The aggregate throughput remains stable, continuing to saturate the available bandwidth. After another seven seconds, we inject an additional 6 ms delay with 1 ms jitter on the subflow via Proxy 1, which is again detected by the client. Consequently, the scheduler adjusts the traffic allocation, and the throughput of the affected subflow drops sharply. However, the unaffected subflow via Proxy 3 does not immediately compensate, leading to a temporary reduction in aggregate throughput. Although each subflow maintains its own congestion control independently, head-of-line blocking still occurs. After roughly one second, the throughput of the subflow via Proxy 3 increases, restoring the aggregate throughput to its original level.

These results demonstrate that VMPTCP can effectively adapt to network dynamics by redistributing traffic among available virtual paths in real time. Even under transient impairments such as packet loss or delay spikes, the aggregate throughput remains largely stable and quickly recovers once alternative paths compensate for degraded ones. Unlike traditional VPNs that rely on explicit and often disruptive path or node reselection, VMPTCP achieves seamless in-band adaptation at the transport layer, ensuring continuous operation and preserving high end-to-end resilience.

5.5.3 Connection Mobility Support

Virtual multipath transport enables connection mobility by leveraging persistent virtual paths to sustain communication when network access changes. We replicate a similar experiment from a previous MPTCP study [40], but with a single-homed client. The client is a laptop equipped with a single Wi-Fi interface. It initially connects to a home router and then switches to a mobile hotspot over a 5G network during transmission. The client

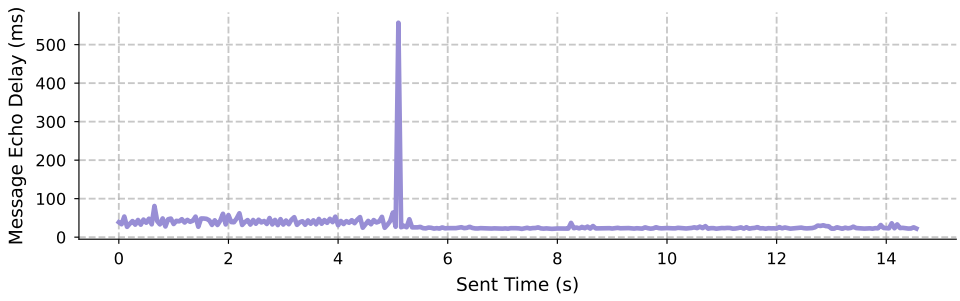


Figure 5.9: Message echo delay of a VMPTCP connection during connection migration.

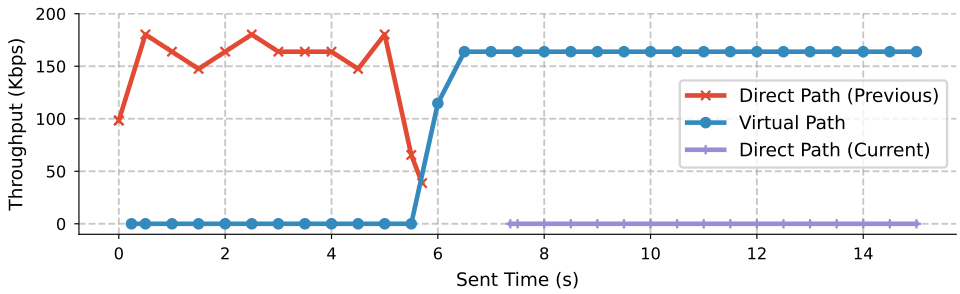


Figure 5.10: Distribution of the throughputs of subflows within a VMPTCP connection during connection migration.

sends a 1024-byte message to the server every 50 ms, and the server immediately echoes each message upon receipt.

Figure 5.9 shows the message echo delay of the VMPTCP connection during migration. We observe a noticeable latency spike when the client switches to the hotspot at approximately 5 seconds after the start of transmission. Within less than one second, however, the delay rapidly decreases, indicating that data transmission continues seamlessly. Although the client's IP address changes, the server consistently recognizes incoming packets and maintains the ongoing connection.

To further analyze connection mobility, Figure 5.10 visualizes the subflow throughput distribution observed at the server, using a 500 ms sampling window. At around 5 seconds, the throughput of the previous direct-path subflow drops sharply as in-flight packets are consumed. Roughly half a second later, no packets from the dead path remain, while a sharp throughput spike appears on the virtual path, which subsequently replaces the original subflow. By approximately 7 seconds, traffic from the new direct path emerges, indicating that the VMPTCP connection has fully migrated, now operating over both available paths. For this lightweight workload, the path scheduler maintains traffic primarily on the virtual path without rebalancing to the new direct path.

These results demonstrate that VMPTCP effectively supports connection mobility by leveraging persistent virtual paths to maintain continuity during network transitions. When the client switches to a new access network, the connection sustains stable throughput and latency. This yields a migration experience that is as seamless as MPTCP on multihomed devices. Most importantly, VMPTCP achieves mobility with negligible deployment overhead, in contrast to QUIC, which requires non-trivial client- and server-side application modifications to provide similar functionality.

5.6 Discussion

Security implications

While most Internet traffic is end-to-end encrypted, as more than 90% of HTTP traffic is protected by HTTPS [66], the use of VPNs provides enhanced security guarantees. Most VPNs additionally encrypts the client-proxy traffic for several reasons. First, relying solely on client-server connection encryption is insufficient. For example, SNI is de facto mandatory for modern HTTPS/TLS because servers depend on it to select the correct certificate when multiple domains share the same IP address, and it inevitably exposes the requested hostname in plaintext [130]. Second, DNS queries are often unencrypted, leaking the requested hostname as well. Third, the server's IP address is always visible and can be used to infer the destination hostname. Last, advanced traffic analysis techniques can classify connections based on their traffic patterns [13]. VPNs eliminate these threats under the assumption that the VPN provider is fully trustworthy. In contrast, VMPTCP allows virtual subflows to be delegated to different VPN providers, each of which only has partial visibility into the connection traffic. Therefore, no single provider needs to be fully trusted.

Comparison with Parallel Sessions

For TCP-based applications, a common alternative to circumvent the theoretical single-flow throughput limitation is to establish multiple parallel TCP sessions. For example, network speed testers often create a set of parallel TCP connections to saturate the available bandwidth [25]. However, this approach requires substantial extra application-level logic and operational overhead. Specifically, the application must partition outgoing data across parallel TCP connections and reassemble it at the receiver to preserve the original byte-stream semantics. While network speed testers can ignore ordering constraints,

general-purpose applications must additionally handle cross-session ordering and reassembly, which significantly increases implementation complexity. On the other hand, managing multiple TCP sockets inevitably introduces nontrivial system overhead, including additional user–kernel boundary crossings, extra memory copies, and more frequent context switches. Fortunately, VMPTCP, leveraging the kernel MPTCP, achieves the same functionality without these concerns.

5.7 Conclusion

This project introduces virtual multipath transport, a new approach to making multipath transport ubiquitous without relying on multihoming. By leveraging widely available VPN proxy nodes, virtual multipath transport enables any connected device, even those with a single network interface, to benefit from the advantages of multipath communication. We design and implement VMPTCP, which reuses the kernel's mature MPTCP implementation and operates transparently with unmodified MPTCP servers. Two variants, VMPTCP-N and VMPTCP-T, demonstrate complementary trade-offs between near-native performance and universal mobility. Through global deployment and extensive evaluation, we show that VMPTCP achieves substantial goodput improvements, up to 4x internationally and 2x domestically, using a single virtual path. Furthermore, VMPTCP maintains robust performance under dynamic network conditions and provides seamless connection mobility, all with negligible deployment overhead.

Virtual multipath transport reimagines not only how multiple paths are established, but also how VPN protocols can be designed. It redefines the relationship between multipath transport and network virtualization, bridging the gap between transport-layer innovation and real-world deployability. We believe this paradigm opens a promising di-

rection toward resilient, adaptive, and universally deployable transport protocols, and inspires future research on integrating multipath transport semantics into next-generation VPNs and Internet architectures.

Chapter 6

Conclusion

Over the past decades, computation has advanced at a pace once captured by Moore's Law, reshaping what applications can do and expect. In contrast, network evolution has proceeded much more slowly, leaving a widening gap between what emerging applications demand and what the Internet can realistically deliver. Ironically, within the networking community, innovation itself progresses rapidly—new transport protocols, programmable data planes, and congestion control algorithms emerge every year. Yet few of these innovations reach deployment at Internet scale, constrained by the stacked layers of legacy protocols, middleboxes, and operational inertia that define today's Internet. The result is a two-sided tension: the network struggles to catch up with the accelerating demands above, while being anchored by the ossified infrastructure below.

This dissertation explores how to bridge these gaps and improve Internet user experience through a series of complementary cross-layer optimizations across provider-side, user-side, and provider–user collaborative dimensions. In the constrained Internet environment, I identify opportunities to judiciously coordinate information and control across the stack while preserving protocol compatibility.

Provider-Side Optimization

The first project, *CloudCookie,* addresses the challenge of bringing data center innovations to *public-facing* Internet traffic. By embedding optimization signals directly into standard Internet headers and leveraging the auto-echo feature, CloudCookie enables public-facing data centers to deploy advanced techniques—such as SRPT-based flow scheduling and predictive load balancing—that were previously confined to private environments. It demonstrates that coordinated cross-layer signaling can yield substantial performance improvements without requiring client-side modifications or violating Internet semantics. This work shows that even within existing protocol boundaries, it is possible to unlock new optimization opportunities through deployable in-network signaling.

User-Side Optimization

The second project focuses on *real-time interactive streaming*, where traffic in opposite directions is tightly coupled. By identifying and analyzing this *cross-directional dependency*, the work extends QUIC's priority framework to datagrams and proposes a prioritization strategy that aligns transport-level behavior with application-level demands, and transforms chaotic congestion into a predictable process. The proposed system achieves significant reductions in motion-to-photon latency and freeze-frame rate in bidirectional view-dependent streaming, further validating the effectiveness of application-guided transport behavior.

Provider–User Collaboration

The final project introduces *virtual multipath transport,* a collaborative approach that makes multipath transport benefits ubiquitously available, including to single-homed devices. I develop an MPTCP-compatible protocol, *VMPTCP*, which deliberately tricks the kernel into reusing its mature MPTCP implementation, establishing multiple virtual paths

through widely available VPN proxy nodes to enhance connection performance, resilience, and mobility. This work reimagines the design of both multipath transport and VPN protocols, further pushing the boundaries of cross-layer network optimization.

Together, these projects establish a unified perspective on cross-layer optimization. By recognizing and leveraging the inherent interdependent nature of protocols running within individual layers, this dissertation uncovers multiple opportunities for optimizing Internet traffic. It further presents the principles and methodologies for designing deployable cross-layer optimization systems that ultimately benefit Internet endusers.

References

[1] Neil Agarwal et al. “Mowgli: Passively Learned Rate Control for Real-Time Video”. In: *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. 2025, pp. 579–594.

[2] Akamai. *QUIC Native Platform Support for Media Delivery Products*. Apr. 2019. URL: https://community.akamai.com/customers/s/article/FAQ-QUIC-Native-Platform-Support-for-Media-Delivery-Products.

[3] Volga Aksoy, Irad Ratmansky, and Amanda Watson. *How does Oculus Link Work? The Architecture, Pipeline and AADT Explained*. Nov. 2019. URL: https://developers.meta.com/horizon/blog/how-does-oculus-link-work-the-architecture-pipeline-and-aadt-explained/.

[4] Akihito Akutsu et al. “2020 Public Viewing–Kirari! Immersive Telepresence Technology”. en. In: *NTT Technical Review* 14.12 (Dec. 2016), pp. 30–35.

[5] Albert Gran Alcoz, Alexander Dietmüller, and Laurent Vanbever. “SP-PIFO: Approximating Push-In First-Out Behaviors using Strict-Priority Queues”. In: *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 59–76. ISBN: 978-1-939133-13-7.

[6] Alexa. *The top 500 sites on the web*. URL: https://www.alexa.com/topsites.

[7] Mohammad Alizadeh et al. “Data center TCP (DCTCP)”. In: *Proceedings of the ACM SIGCOMM 2010 Conference*. 2010, pp. 63–74.

[8] Mohammad Alizadeh et al. “pfabric: Minimal near-optimal datacenter transport”. In: *ACM SIGCOMM Computer Communication Review* 43.4 (2013), pp. 435–446.

[9] Mark Allman and Ethan Blanton. “Notes on burst mitigation for transport protocols”. In: *ACM SIGCOMM Computer Communication Review* 35.2 (2005), pp. 53–60.

[10] Harald T. Alvestrand. *Transports for WebRTC*. RFC 8835. Jan. 2021. DOI: 10.17487/RFC8835. URL: https://www.rfc-editor.org/info/rfc8835.

[11] alvr-org. *ALVR - Air Light VR*. 2025. URL: https://github.com/alvr-org/ALVR.

[12] Congkai An et al. “Tooth: Toward Optimal Balance of Video QoE and Redundancy Cost by Fine-Grained FEC in Cloud Gaming Streaming”. In: *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. 2025, pp. 635–651.

[13] Blake Anderson and David McGrew. “Identifying encrypted malware traffic with contextual flow data”. In: *Proceedings of the 2016 ACM workshop on artificial intelligence and security*. 2016, pp. 35–46.

[14] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. “Sizing router buffers”. In: *ACM SIGCOMM Computer Communication Review* 34.4 (2004), pp. 281–292.

[15] Apple. *About iCloud Private Relay*. 2023. URL: https://support.apple.com/en-us/102602.

[16] Todd Arnold et al. “Cloud Provider Connectivity in the Flat Internet”. In: *Proceedings of the ACM Internet Measurement Conference*. IMC ’20. New York, NY, USA: Association for Computing Machinery, Oct. 2020, pp. 230–246. ISBN: 978-1-4503-8138-3. DOI: 10.1145/3419394.3423613.

[17] Florian Aschenbrenner et al. “From Single Lane to Highways: Analyzing the Adoption of Multipath TCP in the Internet”. In: *Proceedings of the 2021 IFIP Networking Conference*. Aalto, Finland, 2021.

[18] Marcelo Bagnulo. *Threat Analysis for TCP Extensions for Multipath Operation with Multiple Addresses*. RFC 6181. Mar. 2011. DOI: 10.17487/RFC6181. URL: https://www.rfc-editor.org/info/rfc6181.

[19] Marcelo Bagnulo, Amelia Andersdotter, and Christoph Paasch. *Privacy threats and possible countermeasures for Multipath-TCP (MPTCP)*. Internet-Draft draft-bagnulo-mptcp-privacy-00. Work in Progress. Internet Engineering Task Force, July 2019. 6 pp. URL: https://datatracker.ietf.org/doc/draft-bagnulo-mptcp-privacy/00/.

[20] Wei Bai et al. “Information-Agnostic Flow Scheduling for Commodity Data Centers”. In: *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, May 2015, pp. 455–468. ISBN: 978-1-931971-218.

[21] Wei Bai et al. “One more config is enough: Saving (DC) TCP for high-speed extremely shallow-buffered datacenters”. In: *IEEE/ACM Transactions on Networking* 29.2 (2020), pp. 489–502.

[22] Tom Barbette et al. “A High-Speed Load-Balancer Design with Guaranteed Per-Connection-Consistency”. In: *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 667–683. ISBN: 978-1-939133-13-7.

[23] Theophilus Benson, Aditya Akella, and David A Maltz. “Network traffic characteristics of data centers in the wild”. In: *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. 2010, pp. 267–280.

[24] Mike Bishop. *HTTP/3*. RFC 9114. June 2022. DOI: 10.17487/RFC9114.

[25] Netflix Technology Blog. *Building fast.com*. Aug. 2016. URL: https://netflixtechblog.com/building-fast-com-4857fe0f8adb.

[26] David Borman et al. *TCP Extensions for High Performance*. RFC 7323. Sept. 2014. DOI: 10.17487/RFC7323.

[27] David A. Borman, Robert T. Braden, and Van Jacobson. *TCP Extensions for High Performance*. RFC 1323. May 1992. DOI: 10.17487/RFC1323.

[28] Neal Cardwell et al. “Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time”. In: *Queue* 14.5 (2016), pp. 20–53.

[29] Ke Chen et al. “RL-AFEC: adaptive forward error correction for real-time video communication based on reinforcement learning”. In: *Proceedings of the 13th ACM Multimedia Systems Conference*. 2022, pp. 96–108.

[30] Xiaoqi Chen. “Implementing AES encryption on programmable switches via scrambled lookup tables”. In: *Proceedings of the Workshop on Secure Programmable Network Infrastructure*. 2020, pp. 8–14.

[31] Yihua Cheng et al. “GRACE: Loss-Resilient Real-Time Video through Neural Codecs”. In: *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 2024, pp. 509–531.

[32] Yi-Ching Chiu et al. “Are we one hop away from a better internet?” In: *Proceedings of the 2015 Internet Measurement Conference*. 2015, pp. 523–529.

[33] Chromium. *HSTS Preloaded list*. URL: https://cs.chromium.org/chromium/src/net/http/transport_security_state_static.json.

[34] Cloudflare. *quiche*. 2025. URL: https://github.com/cloudflare/quiche.

[35] Federal Communications Commission. *Measuring Fixed Broadband - Thirteenth Report*. Aug. 2024. URL: https://www.fcc.gov/reports-research/reports/measuring-broadband-america/measuring-fixed-broadband-thirteenth-report.

[36] Brett Cruz. *2024 VPN Trends, Statistics, and Consumer Opinions*. 2024. URL: https://www.security.org/resources/vpn-consumer-report-annual.

[37] Andrew R Curtis, Wonho Kim, and Praveen Yalagandula. “Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection”. In: *2011 Proceedings IEEE INFOCOM*. IEEE. 2011, pp. 1629–1637.

[38] Ana Custura, Gorry Fairhurst, and Raffaello Secchi. *Considerations for Assigning a New Recommended Differentiated Services Code Point (DSCP)*. RFC 9435. July 2023. DOI: 10.17487/RFC9435.

[39] Kyzer R. Davis, Brad Peabody, and P. Leach. *Universally Unique IDentifiers (UUIDs)*. RFC 9562. May 2024. DOI: 10.17487/RFC9562. URL: https://www.rfc-editor.org/info/rfc9562.

[40] Quentin De Coninck and Olivier Bonaventure. “Multipath quic: Design and evaluation”. In: *Proceedings of the 13th international conference on emerging networking experiments and technologies*. 2017, pp. 160–166.

[41] Quentin De Coninck and Olivier Bonaventure. “Tuning multipath TCP for interactive applications on smartphones”. In: *2018 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE. 2018, pp. 1–9.

[42] Wladimir De la Cadena et al. “Trafficsliver: Fighting website fingerprinting attacks with traffic splitting”. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 2020, pp. 1971–1985.

[43] Dr. Steve E. Deering and Bob Hinden. *IP Version 6 Addressing Architecture*. RFC 4291. Feb. 2006. DOI: 10.17487/RFC4291.

[44] Apple Developer. *Improving network reliability using Multipath TCP*. 2025. URL: https://developer.apple.com/documentation/foundation/improving-network-reliability-using-multipath-tcp.

[45] Martin Devera and Bert Hubert. *tc-htb(8) - Linux manual page*. 2002. URL: https://man7.org/linux/man-pages/man8/tc-htb.8.html.

[46] Peter A Dinda. "Exploiting packet header redundancy for zero cost dissemination of dynamic resource information". In: *Proceedings of the 6th Workshop on Languages, Compilers, and Run-time Systems for Scalable Computer (LCR 2002)*. 2002.

[47] Jason A Donenfeld. "WireGuard: Next Generation Kernel Network Tunnel." In: *NDSS*. 2017, pp. 1–12.

[48] Vojislav Đukić et al. "Is advance knowledge of flow sizes a plausible assumption?" In: *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. Boston, MA: USENIX Association, Feb. 2019, pp. 565–580. ISBN: 978-1-931971-49-2.

[49] Wesley Eddy. *Transmission Control Protocol (TCP)*. RFC 9293. Aug. 2022. DOI: 10.17487/RFC9293. URL: https://www.rfc-editor.org/info/rfc9293.

[50] Daniel E. Eisenbud et al. "Maglev: A Fast and Reliable Software Network Load Balancer". In: *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. Santa Clara, CA: USENIX Association, Mar. 2016, pp. 523–535. ISBN: 978-1-931971-29-4.

[51] Encompass Digital Media. *Contribution / Distribution*. 2025. URL: https://www.encompass.tv/solutions/tv-networks/contribution-distribution/.

[52] ESnet. *iperf3: A TCP, UDP, and SCTP network bandwidth measurement tool*. 2025. URL: https://software.es.net/iperf/.

[53] *Evercoast - 4D Spatial Volumetric Studio*. 2025. URL: https://evercoast.com.

[54] ExpressVPN. *Fast is getting faster: introducing Lightway Turbo*. 2025. URL: https://www.expressvpn.com/blog/introducing-lightway-turbo.

[55] Hao Fang et al. “Streaming Media over LEO Satellite Networking: A Measurement-Based Analysis and Optimization”. In: *ACM Transactions on Multimedia Computing, Communications and Applications* (2024).

[56] Marwan Fayed et al. “The Ties that un-Bind: Decoupling IP from web services and sockets for robust addressing agility at CDN-scale”. In: *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 2021, pp. 433–446.

[57] Sally Floyd, Dr. K. K. Ramakrishnan, and David L. Black. *The Addition of Explicit Congestion Notification (ECN) to IP*. RFC 3168. Sept. 2001. DOI: 10.17487/RFC3168.

[58] Alan Ford et al. *TCP Extensions for Multipath Operation with Multiple Addresses*. RFC 6824. Jan. 2013. DOI: 10.17487/RFC6824. URL: https://www.rfc-editor.org/info/rfc6824.

[59] Alan Ford et al. *TCP Extensions for Multipath Operation with Multiple Addresses*. RFC 8684. Mar. 2020. DOI: 10.17487/RFC8684. URL: https://www.rfc-editor.org/info/rfc8684.

[60] Bryan Ford et al. *NAT Behavioral Requirements for TCP*. RFC 5382. Oct. 2008. DOI: 10.17487/RFC5382. URL: https://www.rfc-editor.org/info/rfc5382.

[61] Sadjad Fouladi et al. “Salsify: Low-Latency Network Video through Tighter Integration between a Video Codec and a Transport Protocol”. In: *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 2018, pp. 267–282.

[62] The eBPF Foundation. *eBPF - Introduction, Tutorials & Community*. 2025. URL: https://ebpf.io/.

[63] Peter X Gao et al. “phost: Distributed near-optimal datacenter transport over commodity network fabric”. In: *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. 2015, pp. 1–12.

[64] Petros Gigis et al. “Seven Years in the Life of Hypergiants’ off-Nets”. In: *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. SIGCOMM ’21. New York, NY, USA: Association for Computing Machinery, Aug. 2021, pp. 516–533. ISBN: 978-1-4503-8383-7. DOI: 10.1145/3452296.3472928.

[65] Google. *QUIC in The Chromium Projects*. 2025. URL: https://www.chromium.org/quic/playing-with-quic/.

[66] Google Transparency Report. *HTTPS encryption on the web*. URL: https://transparencyreport.google.com/https/overview.

[67] Albert Greenberg et al. "VL2: A scalable and flexible data center network". In: *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*. 2009, pp. 51–62.

[68] Jesse Gross, Ilango Ganga, and T. Sridhar. *Geneve: Generic Network Virtualization Encapsulation*. RFC 8926. Nov. 2020. DOI: 10.17487/RFC8926.

[69] Daniel B. Grossman. *New Terminology and Clarifications for Diffserv*. RFC 3260. Apr. 2002. DOI: 10.17487/RFC3260.

[70] Sangjin Han et al. *SoftNIC: A Software NIC to Augment Hardware*. Tech. rep. UCB/EECS-2015-155. EECS Department, University of California, Berkeley, May 2015.

[71] Youtube Help. *Choose live encoder settings, bitrates, and resolutions*. 2025. URL: https://support.google.com/youtube/answer/2853702?hl=en.

[72] Chi-Yao Hong, Matthew Caesar, and P Brighten Godfrey. "Finishing flows quickly with preemptive scheduling". In: *ACM SIGCOMM Computer Communication Review* 42.4 (2012), pp. 127–138.

[73] Mohammad Hosseini and Viswanathan Swaminathan. "Adaptive 360 VR video streaming: Divide and conquer". In: *2016 IEEE International Symposium on Multimedia (ISM)*. IEEE. 2016, pp. 107–110.

[74] Chun-Ying Huang et al. "GamingAnywhere: An open cloud gaming system". In: *Proceedings of the 4th ACM multimedia systems conference*. 2013, pp. 36–47.

[75] Geoff Huston. "Sizing the buffer". In: *APNIC Blog* (2019). URL: https://blog.apnic.net/2019/12/12/sizing-the-buffer/.

[76] IAB. *IAB/IESG Recommendations on IPv6 Address Allocations to Sites*. RFC 3177. Sept. 2001. DOI: 10.17487/RFC3177.

[77] Intel. *The Intel Tofino series of P4-programmable Ethernet switch ASICs*. URL: https://www.intel.com/content/www/us/en/products/details/network-io/intelligent-fabric-processors/tofino.html.

[78] IPXO. *IPv4 Lease Price Overview 2024 and Insights*. 2025. URL: https://www.ipxo.com/blog/ipv4-lease-price-overview-2024-and-insights/.

[79] Jana Iyengar and Martin Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. May 2021. DOI: 10.17487/RFC9000.

[80] Sushant Jain et al. "B4: Experience with a globally-deployed software defined WAN". In: *ACM SIGCOMM Computer Communication Review* 43.4 (2013), pp. 3–14.

[81] Nick Jones. "Get a head start with QUIC". In: *Cloudflare Blog* (Sept. 2018). URL: https://blog.cloudflare.com/head-start-with-quic/.

[82] Matt Joras and Yang Chi. *How Facebook is bringing QUIC to billions*. Oct. 2020. URL: https://engineering.fb.com/2020/10/21/networking-traffic/how-facebook-is-bringing-quic-to-billions/.

[83] Charlie Kaufman et al. *Internet Key Exchange Protocol Version 2 (IKEv2)*. RFC 7296. Oct. 2014. DOI: 10.17487/RFC7296. URL: https://www.rfc-editor.org/info/rfc7296.

[84] Mike Kosek et al. "MUST, SHOULD, DON'T CARE: TCP Conformance in the Wild". In: *International Conference on Passive and Active Network Measurement*. Springer. 2020, pp. 122–138.

[85] Balázs Kreith, Varun Singh, and Jörg Ott. "FRACTaL: FEC-based rate control for RTP". In: *Proceedings of the 25th ACM international conference on Multimedia*. 2017, pp. 1363–1371.

[86] Adam Langley et al. "The quic transport protocol: Design and internet-scale deployment". In: *Proceedings of the conference of the ACM special interest group on data communication*. 2017, pp. 183–196.

[87] Tony Li et al. *Generic Routing Encapsulation (GRE)*. RFC 2784. Mar. 2000. DOI: 10.17487/RFC2784.

[88] Yang Li et al. "Dissecting and Streamlining the Interactive Loop of Mobile Cloud Gaming". In: *22nd USENIX Symposium on Networked Systems Design and Implementation (NSDI 25)*. 2025, pp. 595–611.

[89] Yuliang Li et al. "HPCC: High precision congestion control". In: *Proceedings of the ACM Special Interest Group on Data Communication*. Association for Computing Machinery, 2019, pp. 44–58.

[90] Miroslav Lichvar and Red Hat. *The Chrony Project*. URL: https://chrony-project.org/.

[91] Etienne Liebetrau. “How Google’s QUIC Protocol Impacts Network Security and Reporting”. In: *Fastvue* (June 2018). URL: https://blog.cloudflare.com/head-start-with-quic/.

[92] Sen Lin, Jianfeng Wang, and Aleksandar Kuzmanovic. “Optimizing Traffic in Public-Facing Data Centers Amid Internet Protocols”. In: *2024 IEEE 32nd International Conference on Network Protocols (ICNP)*. IEEE. 2024, pp. 1–12. DOI: 10.1109/ICNP61940.2024.10858510.

[93] Sen Lin et al. “Leveraging Cross-Directional Dependency in Realtime Interactive Streaming”. In: *Proceedings of the ACM Multimedia Asia (MMAsia ’25)*. Kuala Lumpur, Malaysia: ACM, 2025. DOI: 10.1145/3743093.3771029.

[94] Linux Foundation. *Data Plane Development Kit (DPDK)*. URL: http://www.dpdk.org.

[95] Yanmei Liu et al. *Managing multiple paths for a QUIC connection*. Internet-Draft draft-ietf-quic-multipath-17. Work in Progress. Internet Engineering Task Force, Oct. 2025. 38 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-quic-multipath/17/.

[96] Yanmei Liu et al. *Multipath Extension for QUIC*. Internet-Draft draft-ietf-quic-multipath-14. Work in Progress. Internet Engineering Task Force, Apr. 2025. 37 pp. URL: https://datatracker.ietf.org/doc/draft-ietf-quic-multipath/14/.

[97] LizardByte. *Sunshine - Self-hosted game stream host for Moonlight*. 2025. URL: https://github.com/LizardByte/Sunshine.

[98] Nick McKeown, Guido Appenzeller, and Isaac Keslassy. “Sizing router buffers (redux)”. In: *ACM SIGCOMM Computer Communication Review* 49.5 (2019), pp. 69–74.

[99] Zili Meng et al. “Hairpin: Rethinking packet loss recovery in edge-based interactive video streaming”. In: *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 2024, pp. 907–926.

[100] Meta. *Codec Avatars*. 2025. URL: https://about.meta.com/realitylabs/codecavatars/.

[101] Rui Miao et al. “Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics”. In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 2017, pp. 15–28.

[102] Microsoft. *MsQuic*. 2025. URL: https://github.com/microsoft/msquic.

[103] Microsoft. *RingMaster*. Oct. 2024. URL: https://github.com/microsoft/ringmaste.

[104] Behnam Montazeri et al. "Homa: A receiver-driven low-latency transport protocol using network priorities". In: *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 2018, pp. 221–235.

[105] David Naylor et al. "The cost of the" s" in https". In: *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. 2014, pp. 133–140.

[106] NetSys. *Berkeley Extensible Software Switch (BESS)*. URL: https://github.com/NetSys/bess.

[107] Yunzhe Ni et al. "Cellfusion: Multipath vehicle-to-cloud video streaming with network coding in the wild". In: *Proceedings of the ACM SIGCOMM 2023 Conference*. 2023, pp. 668–683.

[108] Ashkan Nikravesh et al. "An in-depth understanding of multipath TCP on mobile devices: Measurement and system design". In: *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*. 2016, pp. 189–201.

[109] Yoav Nir and Adam Langley. *ChaCha20 and Poly1305 for IETF Protocols*. RFC 8439. June 2018. DOI: 10.17487/RFC8439. URL: https://www.rfc-editor.org/info/rfc8439.

[110] Erik Nordmark, Dr. Steve E. Deering, and Bob Hinden. *IPv6 Global Unicast Address Format*. RFC 3587. Aug. 2003. DOI: 10.17487/RFC3587.

[111] Nvidia. *Omniverse Digital Twin*. 2025. URL: https://docs.omniverse.nvidia.com/digital-twins/latest/index.html.

[112] Vladimir Olteanu et al. "Stateless Datacenter Load-balancing with Beamer". In: *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. Renton, WA: USENIX Association, Apr. 2018, pp. 125–139. ISBN: 978-1-939133-01-4.

[113] Opensignal. *Mobile Network Experience Report*. Jan. 2025. URL: https://www.opensignal.com/reports/2025/01/usa/mobile-network-experience.

[114] Linux Kernel Organization. *drivers/net/dummy.c*. 2025. URL: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/drivers/net/dummy.c.

[115] Linux Kernel Organization. *Universal TUN/TAP Device Driver*. 2002. URL: https://www.kernel.org/doc/Documentation/networking/tuntap.txt.

[116] *P4 Programming Language*. URL: https://p4.org/.

[117] Mirko Palmer et al. "VOXEL: Cross-layer optimization for video streaming with imperfect transmission". In: *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*. 2021, pp. 359–374.

[118] Parveen Patel et al. "Ananta: Cloud scale load balancing". In: *ACM SIGCOMM Computer Communication Review* 43.4 (2013), pp. 207–218.

[119] Tommy Pauly, Eric Kinnear, and David Schinazi. *An Unreliable Datagram Extension to QUIC*. RFC 9221. Mar. 2022. DOI: 10.17487/RFC9221. URL: https://www.rfc-editor.org/info/rfc9221.

[120] Jonathan Perry et al. "Fastpass: A centralized" zero-queue" datacenter network". In: *Proceedings of the 2014 ACM conference on SIGCOMM*. 2014, pp. 307–318.

[121] OpenVPN Project. *OpenVPN: Open Source VPN*. 2024. URL: https://openvpn.net/.

[122] The Tor Project. *Tor*. 2025. URL: https://www.torproject.org/.

[123] *Proto Hologram - Holographic Communications Platform*. 2025. URL: https://protohologram.com.

[124] Enric Pujol et al. "Steering hyper-giants' traffic at scale". In: *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*. 2019, pp. 82–95.

[125] quic-go. *quic-go*. 2025. URL: https://github.com/quic-go/quic-go.

[126] quinn-org. *quinn*. 2025. URL: https://github.com/quinn-rs/quinn.

[127] Idris A Rai et al. "Performance analysis of LAS-based scheduling disciplines in a packet switched network". In: *ACM SIGMETRICS Performance Evaluation Review* 32.1 (2004), pp. 106–117.

[128] Michael A. Ramalho et al. *Stream Control Transmission Protocol (SCTP) Partial Reliability Extension*. RFC 3758. May 2004. DOI: 10.17487/RFC3758. URL: https://www.rfc-editor.org/info/rfc3758.

[129] Waleed Reda et al. "Path persistence in the cloud: A study of the effects of inter-region traffic engineering in a large cloud provider's network". In: *ACM SIGCOMM Computer Communication Review* 50.2 (2020), pp. 11–23.

[130] Eric Rescorla and Tim Dierks. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. Aug. 2008. DOI: 10.17487/RFC5246. URL: https://www.rfc-editor.org/info/rfc5246.

[131] Microsoft Research. *Holoportation*. 2025. URL: https://www.microsoft.com/en-us/research/project/holoportation-3/.

[132] Michael Rudow et al. "Tambur: Efficient loss recovery for videoconferencing via streaming codes". In: *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 2023, pp. 953–971.

[133] Swetank Kumar Saha et al. "Multipath TCP in smartphones: Impact on performance, energy, and CPU utilization". In: *Proceedings of the 15th ACM International Symposium on Mobility Management and Wireless Access*. 2017, pp. 23–31.

[134] Michael Scharf and Alan Ford. *Multipath TCP (MPTCP) Application Interface Considerations*. RFC 6897. Mar. 2013. DOI: 10.17487/RFC6897. URL: https://www.rfc-editor.org/info/rfc6897.

[135] Brandon Schlinker et al. "Engineering egress with edge fabric: Steering oceans of content to the world". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 2017, pp. 418–431.

[136] Brandon Schlinker et al. "Internet performance from facebook's edge". In: *Proceedings of the Internet Measurement Conference*. 2019, pp. 179–194.

[137] Henning Schulzrinne et al. *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550. July 2003. DOI: 10.17487/RFC3550. URL: https://www.rfc-editor.org/info/rfc3550.

[138] Amazon Web Services. *AWS Global Network*. 2025. URL: https://aws.amazon.com/about-aws/global-infrastructure.

[139] Taveesh Sharma et al. “Estimating webrtc video qoe metrics without using application headers”. In: *Proceedings of the 2023 ACM on Internet Measurement Conference*. 2023, pp. 485–500.

[140] Tanya Shreedhar. *Analyzing MPTCP adoption in the Internet*. 2022. URL: https://blog.apnic.net/2022/08/23/analyzing-mptcp-adoption-in-the-internet/.

[141] Siemens. *Digital Twin*. 2025. URL: https://www.siemens.com/global/en/products/automation/topic-areas/digital-enterprise/digital-twin.html.

[142] Vibhaalakshmi Sivaraman et al. “Gemino: Practical and robust neural compression for video conferencing”. In: *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 2024, pp. 569–590.

[143] Speedtest. *United States Median Country Speeds*. Jan. 2025. URL: https://www.speedtest.net/global-index/united-states.

[144] Intel LAN Access Division. *PCI-SIG SR-IOV Primer: An Introduction to SR-IOV Technology (Revision 2.5)*. Jan. 2011.

[145] Starlink. *Specification*. 2025. URL: https://www.starlink.com/legal/documents/DOC-1400-28829-70.

[146] Randall R. Stewart, Michael Tüxen, and karen Nielsen. *Stream Control Transmission Protocol*. RFC 9260. June 2022. DOI: 10.17487/RFC9260. URL: https://www.rfc-editor.org/info/rfc9260.

[147] Randall R. Stewart et al. *Stream Schedulers and User Message Interleaving for the Stream Control Transmission Protocol*. RFC 8260. Nov. 2017. DOI: 10.17487/RFC8260. URL: https://www.rfc-editor.org/info/rfc8260.

[148] Zoom Support. *Accessing meeting and phone statistics*. Dec. 2024. URL: https://support.zoom.com/hc/en/article?id=zm_kb&sysparm_article=KB0070504.

[149] Surfshark. *Surfshark Nexus: a cutting-edge technology to change VPNs*. 2025. URL: https://surfshark.com/blog/surfshark-nexus.

[150] China Telecom. *ChinaNet (AS 4134)*. 2025. URL: https://www.ctamericas.com/company/global-network/chinanet/.

[151] China Telecom. *CN2 (AS 4809)*. 2025. URL: https://www.ctamericas.com/company/global-network/cn2/.

[152] Attila Tomaschek and Moe Long. *Best VPN Service for 2025: Our Top Pick in a Tight Race*. 2025. URL: https://www.cnet.com/tech/services-and-software/best-vpn.

[153] *Use Multipath TCP to create backup connections for iOS*. 2023. URL: https://support.apple.com/en-us/101905.

[154] W3C. *Identifiers for WebRTC's Statistics API*. Mar. 2025. URL: https://www.w3.org/TR/webrtc-stats/.

[155] W3C. *WebRTC Priority Control API*. Mar. 2021. URL: https://www.w3.org/TR/webrtc-priority/.

[156] Shibo Wang et al. "SalientVR: Saliency-driven mobile 360-degree video streaming with gaze information". In: *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*. 2022, pp. 542–555.

[157] Zhou Wang et al. "Image quality assessment: from error visibility to structural similarity". In: *IEEE transactions on image processing* 13.4 (2004), pp. 600–612.

[158] Magnus Westerlund and Stephan Wenger. *RTP Topologies*. RFC 7667. Nov. 2015. DOI: 10.17487/RFC7667. URL: https://www.rfc-editor.org/info/rfc7667.

[159] Kichang Yang et al. "Logan: Loss-tolerant Live Video Analytics System". In: *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*. 2024, pp. 1314–1329.

[160] Kok-Kiong Yap et al. "Taking the edge off with espresso: Scale, reliability and programmability for global internet peering". In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 2017, pp. 432–445.

[161] Sharada Yeluri. "Sizing Router Buffers - Small is the New Big". In: *Juniper TechPost* (2023). URL: https://community.juniper.net/blogs/sharada-yeluri/2023/02/22/sizing-router-buffers.

[162] Yiannis Yiakoumis, Sachin Katti, and Nick McKeown. "Neutral net neutrality". In: *Proceedings of the 2016 ACM SIGCOMM Conference*. 2016, pp. 483–496.

[163] Zhuolong Yu et al. "Programmable packet scheduling with a single queue". In: *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 2021, pp. 179–193.

[164] Chaoliang Zeng et al. “Tiara: A scalable and efficient hardware acceleration architecture for stateful layer-4 load balancing”. In: *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 2022, pp. 1345–1358.

[165] Zhilong Zheng et al. “Xlink: Qoe-driven multi-path quic transport in large-scale video services”. In: *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 2021, pp. 418–432.

[166] Wenjie Zhu et al. “View-dependent dynamic point cloud compression”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 31.2 (2020), pp. 765–781.