Lightweight Internet Bandwidth Allocation and Isolation with Fractional Fair Shares

Marc Wyss, ETH Zurich
Yih-Chun Hu, University of Illinois at Urbana-Champaign
Vincent Lenders, University of Luxembourg
Roland Meier, armasuisse
Adrian Perrig, ETH Zurich

***Abstract*—Ensuring fair bandwidth allocations on the public Internet is challenging. Congestion control algorithms (CCAs) often fail in achieving fairness, especially when different CCAs operate simultaneously. This challenge becomes even more pronounced during volumetric distributed denial-of-service (DDoS) attacks, where legitimate traffic can be starved entirely. One approach to address this challenge is to enforce fairness by allocating bandwidth directly at routers. However, existing solutions generally fall into two categories: those that are easy to deploy but fail to provide secure in-network bandwidth isolation, and those that offer strong isolation guarantees but rely on complex assumptions that hinder real-world deployment.**

To bridge the gap between these two categories, we introduce a new fairness model based on the notion of a per-stream Fractional Fair Share (FFS). At each on-path node, a stream's FFS, represented as packet labels and updated along the forwarding path, conveys its current fair share of egress bandwidth. The combination of a packet-carried FFS and probabilistic forwarding enables effective and scalable isolation of streams with minimal overhead. FFS is the first system to combine low implementation and deployment overhead with effective bandwidth isolation, while remaining robust against source address spoofing and volumetric DDoS attacks, and delivering high performance, scalability, as well as minimal latency and jitter.

We show that FFS effectively isolates bandwidth across 15 different CCAs while keeping latency and jitter minimal. Our high-speed implementation sustains a 160 Gbps line rate on commodity hardware. Evaluated on realistic Internet topologies, FFS outperforms several of the most recent and secure bandwidth isolation systems in both median and total bandwidth allocation. In our security analysis, we prove that FFS guarantees a non-zero lower bound on bandwidth allocation for every traffic stream, ensuring that volumetric DDoS attacks, even when combined with source address spoofing, cannot prevent legitimate communication. Finally, we present an extension of FFS that provides accurate and secure rate feedback to the sender, allowing rapid rate adaptation with minimal packet loss.

I. Introduction

Fair and reliable bandwidth allocation is essential to ensure equitable access during congestion on the public Internet. Today's Internet relies on congestion control algorithms (CCAs) to manage and allocate bandwidth among competing flows, but unfortunately, many CCAs suffer from limitations that prevent them from achieving fair allocations [1], [2]:

Network and Distributed System Security (NDSS) Symposium 2026
23 - 27 February 2026 , San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.240023
www.ndss-symposium.org

- Low-RTT flows ramp up sending rates faster, achieving higher bandwidth due to quicker feedback [3], [4].
- Aggressive CCA variants can dominate bandwidth, leaving conservative ones with smaller shares [5].
- Even minimal packet loss forces CCAs to back off, degrading legitimate traffic [6]–[8].

The problems intensify in the presence of malicious end hosts, which may disregard CCA signals to claim disproportionate bandwidth. In the worst case, volumetric DDoS attacks induce heavy packet loss on legitimate flows, leading to complete starvation of benign communication [9].

Bandwidth-isolation mechanisms, which run on routers and thus enforce fairness directly inside the network, offer a complementary solution for some of the CCA shortcomings [10]. However, as we discuss in our review of related work (Section III), existing approaches exhibit a trade-off between their requirements for implementation and deployment and the properties they deliver, such as isolation, performance, scalability, and security. Systems designed for easy implementation and deployment [11] compromise significantly on secure bandwidth isolation. Often, attackers can still transmit at high rates to gain an excessive share of bandwidth or disrupt legitimate communication through volumetric DDoS attacks. Also, many systems struggle to defend against floods of traffic with spoofed source addresses [12]. Some approaches [13] rely on trust between independent entities—an unrealistic assumption given the diverse actors on the public Internet.

Conversely, systems that provide secure bandwidth isolation in the complex environment of the public Internet are often impractical to deploy due to their extensive requirements or poor scalability. Many [14], [15] rely on assumptions such as stable end-to-end paths and predictable traffic patterns, or require cryptographic operations and precise time synchronization at routers. Systems offering fine-grained bandwidth isolation [16], [17] frequently compromise scalability by requiring, for example, millions of queues at routers, per-packet state proportional to the number of on-path entities, or computational overhead that grows with network size. Due to these limitations, such systems cannot scale to the global Internet and are typically confined to small- or medium-scale deployments, such as private WANs or data centers.

Apparently, designing scalable and secure in-network bandwidth isolation systems that are easy to deploy on the public Internet, with minimal or no requirements, is challenging. The task is made even harder because diverse applications may need high throughput or low and stable latency, or both.

We address these challenges with a system called *FFS*, named after its core principle of assigning each stream a *Fractional Fair Share (FFS)*. The FFS represents a stream's minimum guaranteed share of egress capacity along its forwarding path. At any on-path node (e.g., a router), a stream's FFS at the egress is determined by (i) the stream's FFS from the previously traversed egress, (ii) a relative weight assigned to the ingress at which the stream arrives, as specified by the node's operator, and (iii) the demand of other streams sharing the same egress. Excess bandwidth from underutilizing streams is reallocated to those with higher demand.

FFS **simplifies implementation and deployment** through its streamlined design and minimal requirements. At an egress interface, an FFS node requires just a single FIFO queue, constant memory (independent of the number of flows), and constant-time packet processing. FFS relies solely on basic floating-point operations and simple functions like `min()` and `max()`; it does not depend on external systems or other specialized hardware capabilities. For configuration, FFS requires only a per-node fairness matrix, where entries specify the share of an egress interface's capacity allocated to each ingress interface. Importantly, although FFS adds a small, constant-sized header to each packet, it requires no coordination between nodes, eliminating reliance on cooperation and trust among independent entities on the Internet. Moreover, FFS supports incremental deployment, where bandwidth isolation improves progressively as more nodes adopt the system.

Nevertheless, FFS **delivers effective isolation, high performance, scalability, and security** necessary for deployment on the public Internet. Its bandwidth isolation protects each stream's fair share of bandwidth, thus improving fairness among CCAs. The per-stream-stateless design, combined with low packet processing overhead and probabilistic dropping, enables fast and scalable packet forwarding with low latency and jitter. In addition, FFS defends against volumetric DDoS attacks through bandwidth isolation. Each stream is provably guaranteed a minimal rate, regardless of the communication patterns of other entities. This rate is determined by the fairness matrices of the nodes along the stream's path. FFS is also immune to source address spoofing, as congestion control decisions do not depend on IP addresses. By normalizing label values at ingress, FFS ensures robustness against manipulation by greedy or malicious entities seeking to illegitimately increase their bandwidth share.

Finally, as a useful enhancement, we demonstrate that FFS' packet-carried congestion information can be leveraged to design two endpoint algorithms, one that **minimizes data transmission time** and another that **reduces packet loss**.

Our main contributions are:

- The specification of FFS fairness.
- A highly scalable and secure in-network bandwidth isolation mechanism for enforcing this fairness notion.
- An extension of this mechanism enabling sources to rapidly discover and converge to their fair sending rates while suffering only minimal packet loss.
- A security analysis in the context of the threats present on the public Internet, supported by formal proofs that substantiate our claims.
- Implementations and evaluations showing isolation across 15 CCAs, line-rate forwarding at $160\,\mathrm{Gbps}$ on commodity hardware, larger allocations than prior systems, and consistently low latency and jitter.
- An extensive comparison to related work, emphasizing the distinctive properties and advantages of FFS.

II. Objectives

We design FFS to achieve the following core requirements:

O1 **Isolation:** Enforce fair allocations through end-to-end in-network bandwidth isolation. In the absence of congestion, all traffic should be forwarded without restriction.

O2 **Performance:** Achieve high link- and network utilization while minimizing latency, loss, and jitter.

O3 **Scalability:** Minimize overhead in terms of packet header length, computational costs, memory usage, number of queues, pre-transmission setup, and control-plane communication. Scale to multi-Gbps traffic rates typical of modern inter-domain links.

O4 **Security:** Protect streams from off-path attackers performing volumetric DDoS attacks and tampering with packet fields, including addresses and protocol labels.

O5 **Deployment:** Minimize obstacles to implementation and deployment by maintaining a simple design, avoiding reliance on external systems and specialized hardware capabilities, reducing any setup configuration overhead, eliminate the need for coordination and trust among independent entities, and enabling incremental deployment.

To the best of our knowledge, FFS is the only system that satisfies all these requirements.

III. Background and Related Work

When an egress interface becomes congested, it is necessary to decide which packets to drop and which ones to forward. This raises the question of what constitutes a fair bandwidth allocation and how this allocation can be implemented. In this section, we extensively examine the properties of various bandwidth-allocation mechanisms with respect to the required properties listed in Section II. Figure 1 provides an overview that highlights a clear trend: systems that are easy to deploy often lack secure in-network bandwidth isolation, while those offering strong isolation guarantees have complex requirements that limit their practical deployment. The remainder of this section provides a more in-depth discussion of the related work, serving as background for the design of FFS. Additional related work—though less directly relevant, as it often targets data center networks rather than the public Internet—is discussed in Appendix G.

Insufficient Isolation. Routers typically use first in, first out (FIFO) with tail-drop, where senders with higher rates, e.g., using aggressive CCAs, can claim higher shares of egress

	Requires															Provides						
	Path / traffic stability	Time synchronized routers	Crypt. operations at routers	Duplicate suppression system	Probabilistic monitoring system	Control communication	Pre-transmission setup	Packet header length	State overhead atrouter interface	Key derivation at router	Inter-domain coordination	Trust among deploying entities	Number of queues at egress	Changes to end hosts	Fairness configuration	Fairness mechanism	CCA Isolation (in benign setting)	Minimize loss	Minimize latency	Constant per-packet overhead	Protect against volumetric DDoS	Robust to address spoofing
FIFO	No	No	No	No	No	No	No	O(1)	O(1)	No	No	No	O(1)	No	-	CCA	No	No	No	Yes	No	Yes
L4S	No	No	No	No	No	No	No	O(1)	O(1)	No	No	No	O(1)	Yes	-	CCA	No	Low	Low	Yes	No	Yes
FQ	No	No	No	No	No	No	No	O(1)	O(f)	No	No	Yes	O(f)	No	-	Per flow	Yes	No	Low	Yes	No	No
AFD	No	No	No	No	No	No	No	O(1)	O(1)	No	No	Yes	O(1)	No	-	FQ	Yes	No	Low	Yes	No	No
CSFQ	No	No	No	No	No	(Weights)	No	O(1)	Edge: O(f) Core: O(i)	No	(Weights)	Yes	O(1)	No	(Weights)	(W)FQ	Yes	No	No	Yes	No	No
HCSFQ	Stable paths	No	No	No	No	(Weights)	No	O(d)	O(T)	No	(Weights)	Yes	O(1)	No	(Weights)	H(W)FQ	Yes	No	No	O(d)	No	No
PSP	Traffic History	No	No	No	No	Yes	No	O(1)	O(i*i)	No	No	No	O(1)	No	Allocation matrix	Ing.-Egr. Isolation	No	No	No	Yes	Limi ted	Yes
RCS 2020	No	No	No	No	No	No	No	O(1)	O(i)	No	No	No	O(i)	No	Weights	Ing.-Egr. Isolation	No	No	No	Yes	No	Yes
RCS 2024	No	No	No	No	No	Yes	No	O(1)	O(T)	No	Yes	Yes	O(T)	No	Weights	HWFQ	Yes	No	Low	O(d)	No	No
RCP	No	No	No	No	No	No	Yes	O(1)	O(1)	No	No	Yes	O(1)	Yes	-	Per flow	No	Yes	Low	Yes	No	No
XCP	No	No	No	No	No	No	Yes	O(1)	O(1)	No	No	Yes	O(1)	Yes	-	Flexible	No	Yes	Low	Yes	No	No
FFS	No	No	No	No	No	No	No	O(1)	O(i)	No	No	No	O(1)	No	Fairness matrix	FFS	Yes	Yes	Yes	Yes	Yes	Yes
Z-Lane	Traffic History	Yes	Yes	Yes	No	No	No	O(h)	O(√a)	Yes	No	(only inside group)	O(1)	Yes	Per-egress allocations	Per AS groups (weighted)	Yes	No	Yes	GW: O(h) Router: Yes	Yes	Yes
GLWP	Stable paths	Yes	Yes	Yes	Yes	Yes	Yes	O(h)	O(1)	Yes	Yes	No	O(1)	Yes	Allocation matrix	GMA	Yes	Yes	Yes	RS: O(h) Router: Yes	Yes	Yes
COLIBRI	Stable paths	Yes	Yes	Yes	Yes	Yes	Yes	O(h)	O(1)	Yes	Yes	No	O(1)	Yes	Allocation matrix	N-Tube	Yes	Yes	Yes	GW: O(h) Router: Yes	Yes	Yes
Helia (Flyovers)	Stable paths	Yes	Yes	Yes	No	No	Yes	O(h)	O(a)	Yes	No	No	O(1)	Yes	Allocation matrix	Per active AS	Yes	Yes	Yes	RS: O(h) Router: Yes	Yes	Yes
Hummingbird	Stable paths	Yes	Yes	Opt.	No	Yes	Yes	O(h)	O(r)	Yes	No	No	O(1)	Yes	Per-interface bandwidth	Bandwidth market	Yes	Yes	Yes	Source: O(h) Router: Yes	Yes	Yes

Notation: f: number of flows, i: number of interfaces, T: tree for HFQ, d: depth of tree, h: number of on-path hops, r: number of reservations, a: number of currently sending ASes, GW: gateway, RS: reservation service.

Figure 1: Detailed comparison of FFS and related systems along fine-grained criteria derived from the objectives in Section II.

capacity. ECN [18] signals congestion to end hosts but lacks isolation. L4S [11] uses ECN with an additional egress queue for low latency, relying on CCAs to respond correctly to congestion signals. XCP [19] extends ECN by signaling precise congestion window adjustments to flows, avoiding RTT bias and scaling independently of flow count by embedding congestion data in packet headers. However, it lacks authenticated feedback and relies on source compliance and a reactive monitoring system with per-flow state, vulnerable to address spoofing. RCP [4] signals source transmission rates but lacks isolation in adversarial settings.

Interface-based Isolation. PSP [20] and RCS-2020 [10] isolate traffic between ingress-egress pairs. However, they fail to reliably isolate CCAs, as congestion may not occur at the first common egress but later, when flows share the same interface. In such cases, only the aggregate traffic is controlled, giving more bandwidth to sources with aggressive CCAs.

Fair Queuing (FQ). To ensure robustness against aggressive CCAs or malicious hosts, and to enable diverse CCAs to coexist, bandwidth isolation can be implemented at routers. FQ achieves this by allocating resources based on flow demands. While standard FQ requires per-flow state or queues, CSFQ [13] is stateless at core routers, requiring only per-flow rate estimation at the edges, with rates stored in packet headers. However, CSFQ assumes a fully trusted network environment. AFD [21] assumes that a small number of flows are responsible for the majority of traffic, allowing it to reduce state overhead by tracking only those high-volume flows. FQ approximates max-min fairness, allocating a rate $r_i = \min(d_i, \tau)$ for flow f_i, where τ satisfies $C = \sum_i r_i$. However, greedy end hosts can exploit FQ by generating extra flows, gaining unfairly higher rates.

Weighted Fair Queuing (WFQ). FQ can be extended to WFQ by assigning weights to flows, with rates computed as $r_i = \min(d_i, w_i \cdot \tau)$. CSFQ also supports WFQ. Like FQ, WFQ can be exploited by greedy end hosts creating extra flows.

Hierarchical Weighted Fair Queuing (HWFQ). HWFQ organizes flows hierarchically, assigning weights to both individual flows and flow groups. This ensures fair resource allocation at multiple levels, capping the total rate for a host while allowing fair allocation among its individual flows. Implementing HWFQ efficiently is challenging. HCSFQ [17], an extension of

CSFQ, supports HWFQ on programmable switches but relies on a trusted network, limiting its use to datacenters. Also, scalability is affectedbut is hard to implement efficiently by the requirement to carry hierarchy node lists within packets. RCS-2024 [16] uses Deficit Round Robin, but its computational overhead grows with the hierarchy depth, and the number of queues increases with the hierarchy size, potentially reaching millions on the Internet. RCS-2024 also requires trust between operators for hierarchy weight propagation and is vulnerable to address spoofing, enabling a malicious host to deplete a benign host's fair share of bandwidth.

Secure Inter-domain Bandwidth Reservation Systems. Secure inter-domain bandwidth reservation systems like COLIBRI [22], GLWP [23], and Helia [14] implement fairness based on N-Tube [24], GMA [25], and per-AS allocations. These systems reserve dedicated channels along a path, guaranteeing a specific rate for a set period, regardless of other traffic. Unused reserved bandwidth is dynamically reallocated to best-effort traffic. Bandwidth reservations minimize latency and eliminate the need for CCAs, though they require an explicit setup phase. Z-Lane [15] eliminates the need for a setup overhead for short-lived communication. Hummingbird [26] enables economic fairness by allowing hosts to bid for available bandwidth. However, reservation systems have significant implementation and deployment challenges, including the need for cryptographic operations at routers, time synchronization between routers and end hosts, and path stability, which is hard to maintain with BGP. Extended packet headers reduce payload size, and key exchange and duplicate suppression mechanisms add complexity.

Lessons Incorporated into FFS. For the design of FFS, we draw inspiration from several systems. We assign fair shares to streams and avoid overuse as in bandwidth reservation systems, while preventing underuse through max-min fairness with a τ-based mechanism like (W)FQ. To ensure robustness against address spoofing, we adopt PSP-like local isolation, making forwarding probabilities address-independent. Inspired by HWFQ, we extend this to end-to-end bandwidth isolation via recursive fairness across multiple hops. As in CSFQ, we use probabilistic forwarding to reduce state and queue requirements. Finally, to avoid loss during fair share discovery, we provide explicit feedback to source endpoints, as in XCP.

IV. Model and (Non-)Assumptions

This section introduces the terminology used throughout the description of FFS and highlights its minimal assumptions, requirements, and reliance on external systems.

Node, Interface, and Capacity. The primary entities in our model are nodes (Figure 2a), which represent points of potential network traffic congestion. A node n has a non-empty set of interfaces I^n, where a (possibly empty) subset of interfaces L ($\subset$ I) are local interfaces. While a local interface represents a communication endpoint (source or destination of traffic), a non-local interface connects the node via some link to a non-local interface of another node.[1] We assume nodes are connected, meaning each has at least one non-local interface. For a given node n, we denote the capacity of an interface $i \in \mathrm{I}^n$ as C_i^n. Connected nodes have equal interface capacities in both directions.[2] We consider a node's interface j as congested if $\sum_{i\in\mathrm{I}^n} \mathrm{R}_{(i,j)}^n > \mathrm{C}_j^n$, where $\mathrm{R}_{(i,j)}$ represents the aggregate rate of traffic originating from interface i towards interface j. When the context focuses on a single node, we omit the reference to n for simplicity, e.g., we write C_j instead of C_j^n. We further omit the reference to j when it is clear from the context, e.g., we simply write C instead of C_j.

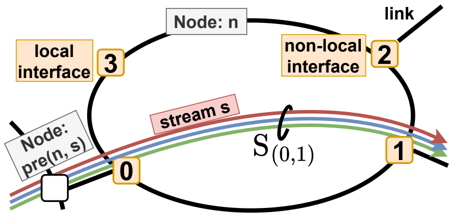
A practical instantiation of a node is for example an autonomous system (AS), where node interfaces correspond to AS interfaces. A node may also represent an individual router—whether home, edge, internal, or border—or the networking stack of a physical or virtual end host. In this case, node interfaces may refer to network interface card (NIC) interfaces, application sockets, IP prefixes, or specific IP addresses. Nodes can be nested, with, e.g., an AS as a single node and its routers as nested nodes.

Streams. Instead of the traditional notion of a flow, which is commonly defined as a 5-tuple (source IP, destination IP, source port, destination port, protocol), we follow prior work [16] and use the notion of a *stream*. This allows us to formally reference the fairness-matrix entries associated with the nodes a packet traverses, which determine its forwarding probability, and generalizes a transport flow: it may correspond to a single flow or an aggregate of flows. We also adopt the term stream to clearly distinguish our setting from per-flow fairness mechanisms, which can be exploited by artificially creating additional flows to obtain a larger share of bandwidth.

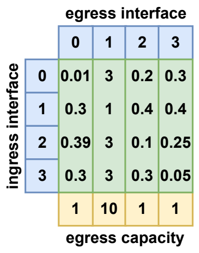
A stream s of length ℓ is a sequence of connected nodes and their corre interfaces: $s = [(n^0, i^0, j^0), (n^1, i^1, j^1), \dots, (n^\ell, i^\ell, j^\ell)]$. All packets of the same stream follow the same forwarding path, where i^0 and j^ℓ are local interfaces, and all other interfaces are non-local. We assume that no two streams start at the same interface.[3] The first and last node in a stream typically refer to an edge router, an end host, an application, or a single port. We define $\mathrm{pre}(n, s)$ as the preceding on-path node of stream s relative to node n. Additionally, we define $\mathrm{S}_{(i,j)}^n$ as the set of streams entering through ingress interface i and exiting via egress interface j at node n.

Fairness Matrices. Each node n has a fairness matrix M^n, where the entry in the *i*-th row and *j*-th column indicates the fair capacity share for ingress interface i at egress interface j. Importantly, the matrix is thus defined at the level of interfaces rather than individual flows. For a given egress j, the sum of all entries in its row equals its capacity: $\sum_{i\in\mathrm{I}} \mathrm{M}_{(i,j)} = \mathrm{C}_j$. The sum of entries for any ingress i is not constrained by its capac-

[1] Representing endpoints as interfaces simplifies the later notion of streams.
[2] For simplicity; the model extends to differing ingress and egress capacities.
[3] This assumption ensures stream isolation; otherwise all traffic sent over the corresponding interfaces will be treated as a single stream. Streams starting at the same interface can be assigned separate local interfaces to comply.



(a)



ingress interface \ egress interface	0	1	2	3
0	0.01	3	0.2	0.3
1	0.3	1	0.4	0.4
2	0.39	3	0.1	0.25
3	0.3	3	0.3	0.05
egress capacity	1	10	1	1

(b)

Figure 2: Core concepts: (a) abstract node model and (b) example fairness matrix (in Gbps).

ity. In Figure 2b, the entry for ingress 1 and egress 2 ($\mathrm{M}^n_{(1,2)}$) is 0.4 Gbps, a large share of the 1 Gbps egress capacity. Diagonal entries, like $\mathrm{M}^n_{(1,1)}$, enable packet reflection, such as for ICMP traceroute replies. A basic FFS configuration assigns each ingress an equal share of egress capacity. However, a node may use its fairness matrix to prioritize certain neighbors by assigning larger matrix entries to those with greater importance (e.g., higher-bandwidth agreements between ASes), similar to defining weights and allocation matrices in related work [14], [16], [20]. For example, consider an egress with a physical capacity of 10 Gbps. An AS might then allocate matrix entries such as 1 Gbps each for six default-contract neighbors, and 4 Gbps for a premium-contract neighbor. FFS uses matrix entries to *guide allocation* but does not strictly isolate traffic between interface pairs. Unused bandwidth from underutilized streams is reallocated to all streams sharing the same egress, not just those on the same interface pair. Unlike bandwidth reservation systems, an FFS node can adjust its matrix anytime without risking over-allocation. An FFS node requires only the fairness matrix for configuration.

(Non-)Assumptions. We assume that entities agree on how FFS labels are encoded into packet headers.[4] Beyond this, FFS imposes no additional assumptions, requirements, or reliance on external systems. This minimalism stands in stark contrast to many prior approaches and significantly simplifies FFS deployment. FFS avoids common assumptions in the literature, such as the presence of only one bandwidth bottleneck along a path [27], congestion being limited to specific locations like the last mile [28], volumetric DDoS being a solved problem in today's Internet [11], or trust among independent entities [13]. It further eliminates the need for cryptographic operations at source endpoints or routers, avoiding assumptions about the security of cryptographic primitives [14] and thus rendering the system inherently quantum-safe. In addition, FFS operates without reliance on external components such as duplicate suppression mechanisms [15] or dedicated monitoring infrastructure [23]. To better illustrate the system's full capabilities, we assume in the following sections that all potential congestion points (i.e., nodes) implement FFS. However, this is not a requirement; FFS also supports incremental deployment, as discussed in Section XIII.

[4]Even without this assumption, i.e., if nodes use their own encoding schemes that are not understood by others, FFS still provides local isolation, though it would no longer guarantee CCA independence.

V. Fairness According to FFS

This section describes fairness according to FFS, which serves as the foundation for achieving scalable and secure bandwidth isolation in later sections. An overview of the key terminology is provided in Table I.

Fractional Fair Shares. For a stream s going through interfaces i and j of node n, we associate a rate f^n_s, referred to as the *Fractional Fair Share (FFS)*. The FFS represents the stream's current fair share of egress capacity and is computed iteratively (i.e., fractionally) as:

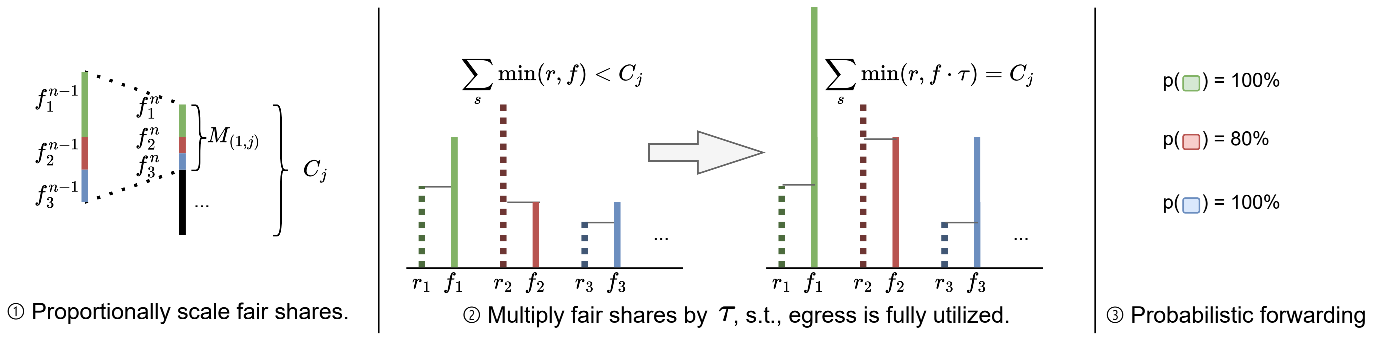
$$\mathrm{f}^n_s = \frac{\mathrm{f}^{\mathrm{pre}(n,s)}_s}{\mathrm{F}^{\mathrm{pre}(n,s)}} \times \mathrm{M}^n_{(i,j)}, \tag{1}$$

where $\mathrm{F}^{\mathrm{pre}(n,s)} = \sum_{s \in \mathrm{S}^n_{(i,j)}} \mathrm{f}^{\mathrm{pre}(n,s)}_s$. Thus, a stream's FFS is *normalized*, meaning it represents a share of $\mathrm{M}^n_{(i,j)}$ distributed among all streams passing the same interface-pair, weighted by their FFS at the most recently traversed egress. Therefore, if $\mathrm{S}^n_{(i,j)}$ is non-empty, the sum of all normalized fractional fair shares from i to j at node n is always equal to $\mathrm{M}^n_{(i,j)}$, i.e., $\mathrm{F}^n_{(i,j)} = \sum_{s \in \mathrm{S}^n_{(i,j)}} \mathrm{f}^n_s = \mathrm{M}^n_{(i,j)}$, and zero otherwise. For a stream's source node n^0, we have $\mathrm{f}^{n^0}_s = \mathrm{M}^{n^0}_{(i^0,j^0)}$.

Reallocating Underused Shares. If an interface is not experiencing congestion, it forwards all traffic without restriction. When congestion occurs at an interface j of node n, each stream s could, in principle, be assigned a rate equal to its FFS, f^n_s, since the total fair shares directed toward interface j cannot exceed its capacity: $\sum_{i \in \mathrm{I}^n} \sum_{s \in S^n_{(i,j)}} \mathrm{f}^n_s \le \sum_{i \in \mathrm{I}^n} \mathrm{M}^n_{(i,j)} = C^n_j$. However, this represents the worst-case (i.e., minimum) allocation, which is only necessary if all streams are sending at or above their FFS. If some streams are sending below their FFS, their unused share can be reallocated to other streams. In our fairness model, we redistribute all unused bandwidth fairly among streams with demand exceeding their FFS, again weighted according to their FFS. We define r^n_s as the rate of a stream s after it passes the egress interface of on-path node n. Therefore, the rate r^n_s of a stream s after passing the egress interface of node n is given by:

$$\mathrm{r}^n_s = \min(\mathrm{r}^{\mathrm{pre}(n,s)}_s,\ \mathrm{f}^n_s \times \tau) \tag{2}$$

The fairness parameter τ is defined as the unique solution to the equation $C = \sum_{i \in \mathrm{I}^n} \sum_{s \in S^n_{(i,j)}} \mathrm{r}^n_s$, and we refer to $(\mathrm{f}^n_s \times \tau)$ as the *Current Fair Rate (CFR)*. In the absence of congestion, fairness enforcement is not required, and we set $\tau = \infty$. The node itself does not determine the relative weights of the streams in $\mathrm{R}^n_{(i,j)}$; rather, this is established by the previous node. Thus, our fairness model follows a hierarchical approach, where each node focuses solely on enforcing local fairness according to its own policies. The flexible granularity and independence of streams make FFS ideal for multi-path protocols and inherently enable traffic protection in both directions. We prove a lower bound on



f_1^{n-1}
f_2^{n-1}
f_3^{n-1}
f_1^{n}
f_2^{n}
f_3^{n}
$M_{(1,j)}$
C_j
① Proportionally scale fair shares.
$\sum_s \min(r, f) < C_j$
$\sum_s \min(r, f \cdot \tau) = C_j$
r_1 f_1 r_2 f_2 r_3 f_3
② Multiply fair shares by τ, s.t., egress is fully utilized.
p(🟩) = 100%
p(🟥) = 80%
p(🟦) = 100%
③ Probabilistic forwarding

Figure 3: Illustration of the key steps in Algorithm 1. Here, r denotes a stream's sending rate and f its fair share.

an FFS stream's allocation, which holds irrespective of other network traffic, in Lemma 2 in Section XI.

Comparison to Related Work. FFS can be broadly interpreted as a *dynamic adaptation of GMA* [25]. While GMA determines static fair shares based on a worst-case scenario (assuming maximum congestion at all interfaces), FFS redistributes unused bandwidth to streams with higher demand. FFS can also be viewed as *WFQ [13] with adaptive weights* ($w = f_s^n$) that carry information from prior nodes to achieve hierarchical fairness. Although FFS appears *similar to HWFQ due to its hierarchical approach*, it operates differently (Table II). In interface-based HWFQ, unused bandwidth is first allocated to other streams on the same ingress with unmet demand. In contrast, FFS redistributes unused bandwidth among all streams sharing the same egress, regardless of their ingress. While FFS streams align more closely with their fair shares, HWFQ may be more economically suitable [16]. In data centers, for example, with tenants at the first level and flows within each tenant at the second level of the hierarchy, HWFQ may be a better fit [17]. FFS' key advantage is its streamlined design and minimal requirements, delivering effective isolation, high performance, scalability, and security, making it well-suited for Internet deployment.

VI. Bandwidth Isolation

We now present an algorithm to enforce FFS fairness, translating the target rates outlined earlier into efficient packet-level operations *without requiring nodes to identify individual streams or keep per-stream state*. It only adds a small, constant packet size overhead, requires constant per-packet computation, and maintains constant per-interface state.

Overview. Each packet p is extended with two labels, $\boxed{\text{p.r}}$ and $\boxed{\text{p.f}}$, denoting it's stream's current rate (r) and FFS (f). Labels are initialized at the stream's source endpoint or the first on-path node, both initialized to the stream's rate. Algorithm 1 describes the fairness enforcement algorithm at a node's egress. It updates a packet's FFS according to Equation (1) (Line 1) and computes its forwarding probability γ (Line 2), detailed further below. Then, it adjusts the stream's encoded rate based on γ, reducing r to $r \cdot \gamma$ (Line 3). It optionally invokes Algorithm 2 (Line 4), which enables the source to

Table I: Overview of key terminology. The metrics *actively monitored* at egress j are emphasized at the bottom.

Term	Description
$S_{(i,j)}^n$	Streams from ingress i to egress j at node n.
$M_{(i,j)}^n$	Fair capacity share for ingress i at egress j.
pre(n,s)	Upstream node for stream s relative to node n.
C_j^n	Capacity of interface i at node n.
f_s^n	FFS for stream s going through node n.
$f_s^n \times \tau$	Current Fair Rate (CFR)
$F_{(i,j)}^n$	$\sum_{s \in S_{(i,j)}^n} f_s^n$
r_s^n	Rate of stream s after passing node n.
$R_{(i,j)}^n$	Aggregate traffic rate from interface i to j.
$R_{(*,j)}^n$	Total traffic rate directed to egress j from all ingress interfaces, equal to $\sum_i R_{(i,j)}$.
$F_{(i,j)}^{\text{pre}(n,s)}$	Total fair share sum from ingress i to egress j, computed based on weighted packet sizes.
A_j	Rate of outgoing, i.e., accepted, traffic.
τ_j	Fairness parameter per Equation (2), computed as in Appendix A.

Table II: HWFQ vs. FFS allocations for three streams from two ingresses sharing a 15 Gbps egress.

Stream	Rate	Fair Share	HWFQ	FFS
s_1 (i_1)	3 Gbps	6 Gbps	3 Gbps	3 Gbps
s_2 (i_1)	4 Gbps	1 Gbps	4 Gbps	$1.\overline{3}$ Gbps
s_3 (i_2)	16 Gbps	8 Gbps	8 Gbps	$10.\overline{6}$ Gbps

discover its fair rate as described in Section VIII. Lastly, it forwards the packet with probability γ (Line 5). Figure 3 presents an illustration of the key steps. We next detail the derivation of $F_{(i,j)}^n$, the computation of γ, and its rationale.

Sum of Fair Shares. FFS fairness requires that for $R_{(i,j)}^n > 0$, the sum of fractional fair shares equals $M_{(i,j)}^n$, ensuring $F_{(i,j)}^n = M_{(i,j)}^n$. While $M_{(i,j)}^n$ is constant, $F_{(i,j)}^n$ depends on $F_{(i,j)}^{\text{pre}(n,s)}$, which is challenging to estimate. A packet's label reveals its stream's FFS, but identifying the stream and calculating total FFS would require per-stream state. To

Algorithm 1: Packet forwarding at egress j of node n, for a packet p arriving from some ingress i. We use $\boxed{\cdot}$ to refer to packet labels and $\leftarrow$ for assignments.

```
1 $\boxed{\text{p.f}} \leftarrow (\boxed{\text{p.f}} \;/\; \text{F}^{\text{pre}(n,s)}_{(i,j)}) \cdot \text{M}^{n}_{(i,j)}$
2 $\gamma \leftarrow \min(1,\; (\boxed{\text{p.f}} \;/\; \boxed{\text{p.r}}) \cdot \tau)$
3 $\boxed{\text{p.r}} \leftarrow \boxed{\text{p.r}} \times \gamma$
4 Call Algorithm 2 (for rate feedback)
5 Forward packet with probability $\gamma$, otherwise drop it.
```

address these challenges, we express $\text{F}^{\text{pre}(n,s)}_{(i,j)}$ as $\text{F}^{\text{pre}(n,s)}_{(i,j)} = \sum_{s \in \text{S}^{n}_{(i,j)}} \text{f}^{\text{pre}(n,s)}_{s} = \sum_{s \in \text{S}^{n}_{(i,j)}} \text{r}^{\text{pre}(n,s)}_{s} \cdot \frac{\text{f}^{\text{pre}(n,s)}_{s}}{\text{r}^{\text{pre}(n,s)}_{s}}$, interpreting it as a weighted rate: Similarly to how $\text{R}^{n}_{(i,j)} = \sum_{s \in \text{S}^{n}_{(i,j)}} \text{r}^{\text{pre}(n,s)}_{s}$ is measured using each packet's length, call it p.len, we measure the weighted rate $\text{F}^{\text{pre}(n,s)}_{(i,j)}$ using p.len $\times$ ($\boxed{\text{p.f}}$ / $\boxed{\text{p.r}}$). Therefore, just as the rate $\text{R}^{n}_{(i,j)}$ can be computed based on the size of incoming packets (e.g., using exponential averaging), $\text{F}^{\text{pre}(n,s)}_{(i,j)}$ can be derived using the size of incoming packets, where each packet's size is weighted by the label $\boxed{\text{p.f}}$ and divided by the label $\boxed{\text{p.r}}$.

Forwarding Probability. The computation of γ in Line 2 is derived from Equation (2): $\text{r}^{n}_{s} = \min(\text{r}^{\text{pre}(n,s)}_{s},\ \text{f}^{n}_{s} \times \tau) = \text{r}^{\text{pre}(n,s)}_{s} \cdot \min(1,\ \text{f}^{n}_{s} \times \tau \;/\; \text{r}^{\text{pre}(n,s)}_{s}) = \text{r}^{\text{pre}(n,s)}_{s} \cdot \gamma$.

Measurements. Equation (2) can be interpreted as WFQ with packet-carried weights. Therefore, we can reuse existing methods for estimating τ in WFQ, such as the constant per-packet-overhead approach from (H)CSFQ [13], [17], as detailed in Appendix A. The metrics tracked at some egress j are listed and highlighted in Table I. The rates $\text{R}^{n}_{(i,j)}$, $\text{R}^{n}_{(*,j)}$, and $\text{F}^{\text{pre}(n,s)}_{(i,j)}$ are monitored using the state of packets prior to the execution of Algorithm 1, while A_j is tracked based on the packets being forwarded after the algorithm's execution.

VII. Bandwidth Isolation Evaluation

We evaluate FFS for CCA isolation and network utilization.[5]

A. CCA Isolation

We implement FFS and evaluate its fairness, link utilization, RTT, and jitter for competing CCAs, comparing it to a standard FIFO queue and (H)CSFQ. We assess two FFS variants: (i) FFS-C100, where $\sum_{i \in \text{I}} \text{M}(i,j) = 1.00 \cdot \text{C}j$, and (ii) FFS-C85, where $\sum_{i \in \text{I}} \text{M}(i,j) = 0.85 \cdot \text{C}_j$. The goal of FFS-C100 is to maximize throughput, whereas the goal of FFS-C85 is to reduce latency by limiting queue buildup.

Implementation. We implement an FFS node in Python using netfilterqueue [29] to intercept and process system traffic. For packet parsing, we avoid using scapy [30] and instead implement our own parser, which is about 40 times faster.

[5]We focus on aspects that are particularly relevant for our protocol. As prior studies have already demonstrated key properties, we avoid repeating them. E.g., [17] confirms that probabilistic forwarding, the principle underlying FFS, remains effective across large topologies with multiple hops and varying RTTs.

Table III: Performance of different scheduling schemes.

Scheme	**F**	**U**	**L(2)**	**L(50)**	**J(2)**	**J(50)**
FIFO	30.7%	80.4%	0%	7.1%	0%	7.1%
(H)CSFQ	72.1%	91.5%	0%	38.1%	0.9%	90.5%
FFS-100	71.3%	93.9%	0%	36.9%	0.5%	89.3%
FFS-85	88.5%	79.3%	100%	100%	99.1%	100%

We use a custom Mininet [31] topology with two sources connected via FFS nodes over a common bottleneck link to two destinations, configured with a $10\,\text{ms}$ one-way delay. The FFS node's fairness matrix entries are set equally for both sources to ensure equal bandwidth allocation. We evaluated all TCP CCAs pre-installed on Ubuntu 24.04.1 LTS with default settings, using iperf3 [32] for traffic generation. The evaluated CCAs are bbr, bic, cdg, cubic, highspeed, htcp, hybla, illinois, lp, nv, scalable, vegas, veno, westwood, and yeah, where some of them are known to produce particularly bursty traffic patterns. We exclude dctcp, because it is designed for data centers. Each one-minute measurement records the RTT every second using ping, enabling consistent assessment of FFS across CCAs. To compute traffic rates at the FFS node, we implement exponential averaging [13]. Instead of calculating the accepted egress rate (A_j) based solely on the size of accepted packets, we update it using the expected value of all egress packets, i.e., multiplying the packet size by the forwarding probability, yielding more stable and accurate rate estimates. Packets are forwarded with probability γ by uniformly sampling a random number $n \in [0, 1]$ and forwarding the packet if $n < \gamma$, otherwise dropping it.

Fairness, Link Utilization, RTT, and Jitter. We evaluate those metrics as follows:

F	Fairness across all CCA pairs, defined as the mean of $\min(t_1, t_2)/\max(t_1, t_2)$, where t_1 and t_2 are the achieved throughputs of two competing CCAs.
U	Average utilization among all CCA pairs.
L(x)	Fraction of CCA pairs with latency increase $< x$ ms.
J(x)	Fraction of CCA pairs with jitter $< x$ ms.

Table III compares these metrics for FIFO, (H)CSFQ, FFS-C100, and FFS-C85. Achieving fairness and isolation is inherently challenging, as CCAs differ in how they interpret signals such as loss and delay, and in how aggressively they increase their sending rates. As expected, FIFO does not reliably ensure fairness between CCAs (F = 30.7%) and often results in high latency. In contrast, FFS-C100 and (H)CSFQ significantly improve fairness (F = 71.3% and 72.1%, respectively), latency, and jitter across diverse CCA combinations while maintaining high link utilization. FFS-C100 performs very similarly to (H)CSFQ, reflecting their shared underlying mechanisms (probabilistic dropping and the τ-finding algorithm) as well as similar configurations (per-flow fairness in (H)CSFQ and equal fairness matrix entries in FFS). However, delay-sensitive CCAs such as Vegas may still be disadvantaged when competing with more aggressive CCAs under these schemes. FFS-C85 further increases fairness

(F = 88.5%) while minimizing latency and jitter, all while maintaining utilization comparable to FIFO. Notably, an FFS node achieves low forwarding latency by deciding whether to forward or drop packets *before* they enter the egress queue, directly limiting queue growth. A more detailed and visual comparison of those results is provided in Appendix B.

Trust and Complexity. Our evaluation shows that FFS can match or even surpass (H)CSFQ in terms of fairness, link utilization, RTT, and jitter. Beyond these performance metrics, FFS addresses the issue of trust among network participants (end hosts, routers, and ASes) and comes with significantly lower implementation complexity and computation cost compared to (H)CSFQ, particularly its hierarchical variant. Regarding network trust, all fair queuing (FQ) solutions, including (H)CSFQ, are vulnerable to attackers creating multiple flows to gain a larger share of bandwidth. In contrast, FFS avoids this problem because bandwidth allocations are determined by fairness matrices rather than the number of flows. With FFS, generating additional flows does not increase bandwidth beyond what a single flow would receive, unless the original flow underutilizes its fair share. Moreover, (H)CSFQ is susceptible to label manipulation, where an attacker might claim a smaller flow rate in its packet labels to increase forwarding probability. FFS mitigates this risk by performing label normalization at ingress nodes, preventing tampering. In terms of implementation complexity, (H)CSFQ requires packet headers that grow with the number of hierarchy levels and imposes state and per-packet computation overhead (e.g., requiring tracking multiple τ values) that scales with the hierarchy depth. FFS, on the other hand, maintains a constant header size and constant per-packet computation overhead while implementing hierarchical fairness. Lastly, FFS does not require knowledge of forwarding paths in advance, and it can handle dynamic or unpredictable path changes without maintaining state for any specific hierarchy.

B. Network Utilization

We evaluate allocation sizes on the public Internet by simulating FFS's end-to-end bandwidth and comparing it to Helia's [14] per-AS allocations and GLWP's [23] allocations based on GMA [25], as those systems most closely match our desired properties (Table I).

Implementation. We implement a parallelizable simulator in Go [33] to evaluate network utilization of FFS and related systems. The simulator supports experiments on (i) realistic topologies, (ii) random power-law degree graphs, and (iii) custom topologies. The realistic topologies, used for this evaluation, are based on the top 2000 Tier-1 and Tier-2 ASes from the CAIDA AS relationships dataset [34]. Random power-law degree graphs approximate the structure of real-world IT networks [35], enabling evaluations on various topology sizes and thus faster tests. Custom topologies are available for specific needs, like debugging corner cases. In these topologies, each node represents an AS, and nodes are connected through inter-domain links. The link capacity is determined using a degree-gravity model, where capacity is proportional to the product of the degrees of adjacent nodes [36], ranging from 40 to 400 Gbps. This reflects the real-world behavior where more connected nodes typically have higher forwarding capabilities [37]. We assign every AS a local fairness matrix, with the entries corresponding to the tuple consisting of the local interface and some egress being set to 10 % of the egress capacity. Matrix entries between two non-local interfaces get initially assigned the capacity of the inter-domain link attached to the ingress interface. If the sum of matrix entries in a column is smaller than the egress capacity, the remaining bandwidth is allocated to the local interface. If the sum exceeds egress capacity, the entries are scaled down accordingly.

Unlike emulations sending actual data, our simulations focus on fairness at the stream level, avoiding packet-level computations. To compute τ_e, we solve a numerical optimization problem using Brent's method [38].[6] In case of congestion at an AS egress, we find the root of the function $g(\tau)$, where:

$$g(\tau) = \sum_{i \in \mathrm{I}^n} \sum_{s \in \mathrm{S}^n_{(i,j)}} \min\left(\mathrm{r}_s^{\mathrm{pre}(n,s)},\ \mathrm{f}_s^n \times \tau\right) \ - \ C$$

Each node attempts to send traffic to 20% of other nodes using shortest paths (this percentage is adjustable). In FFS, source ASes start at their maximum rate, and in each round, a stream's rate is reduced at congested egresses to the CFR, with the source adjusting its rate accordingly. For bandwidth reservation systems like GMA and Helia, nodes request reservations, and we evaluate the allocated bandwidth. We analyze the distribution of bandwidth across all active paths.

Results. The results in Figure 4 show that FFS achieves higher allocations than Helia and GLWP, with median values of 5.97 Gbps, 5.62 Gbps, and 0.64 Gbps, respectively. This is expected, as FFS dynamically reallocates unused bandwidth, unlike GLWP, which uses static allocations. Helia also performs well overall but does not allow reallocation of underutilized bandwidth between ASes. We expect similar behavior to FFS from other work-conserving systems like RCS-2024, though these systems do not meet all of our objectives, particularly in terms of security and scalability.

VIII. Rate Feedback

As an optional feature, orthogonal to our core design, we enable on-path nodes to convey a stream's fair rate to the destination, which authenticates and returns it to the source. This allows the source to rapidly adjust its sending rate with minimal loss, overcoming the tradeoff that is fundamental to traditional CCAs [39]. FFS' rate feedback mechanism offers higher precision than L4S, achieving accuracy similar to XCP but without requiring senders to maintain a congestion window or relying on a uniform algorithm. We stress that respecting rate feedback is not required for congestion control, as fairness is enforced via bandwidth isolation at on-path nodes.

[6]This approach is only used in the network utilization simulations. For real systems, including our Python (Section VII) and DPDK (Section XII) implementations, we use the method described in Appendix A.

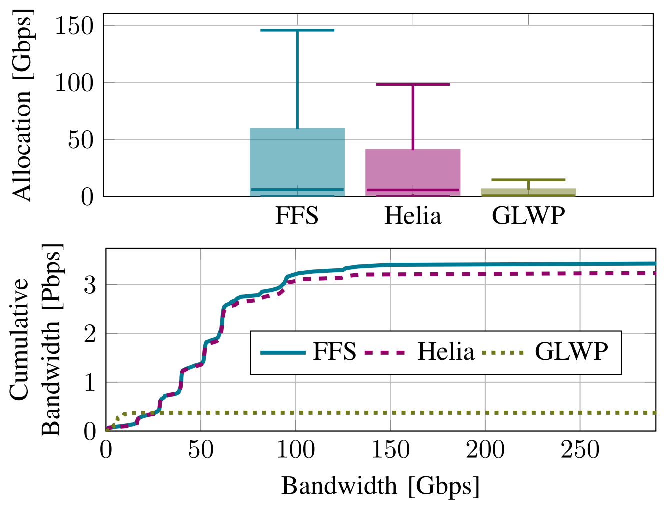


Figure 4: Network utilization of FFS, Helia, and GLWP on the CAIDA top 2000 topology.

Algorithm 2: Algorithm updating the RI label.

```
1 if $R_* > C$ then
      // Congestion at egress interface
2     $\boxed{\text{p.RI}} \leftarrow \min(\boxed{\text{p.RI}}, \boxed{\text{p.f}} \times \tau)$
3 else
      // No congestion at egress
         interface
4     $\text{RI} \leftarrow \max(\boxed{\text{p.r}} \times \left(\frac{C}{R_*}\right), \boxed{\text{p.f}})$
5     $\boxed{\text{p.RI}} \leftarrow \min(\boxed{\text{p.RI}}, C, RI)$
```

Packet Label. Each packet is extended with a Rate Indicator (RI) label $\boxed{\text{p.RI}}$, updated by routers along the path. During congestion, the RI reflects the stream's minimum CFR, and when there is no congestion, it provides a lower bound on the source's fair rate. The source initializes the RI to its maximum value (∞). If a source does not wish to learn its fair rate, it omits the RI label, and on-path nodes will not add it. A node updates the packet's RI label (Algorithm 2) along with enforcing bandwidth isolation (Algorithm 1). If congestion occurs at the egress, the RI is updated to the stream's CFR ($\boxed{\text{p.f}} \cdot \tau$) if it's lower than the current value. If there is no congestion, the node calculates the maximum of the FFS and $\boxed{\text{p.r}} \cdot (\frac{C}{R_*})$, where $\frac{C}{R_*}$ adjusts the rate for full egress utilization, and ensures it does not exceed the egress capacity C.

Rate Feedback. The source learns the RI indirectly through a notification from the destination, which can send the median of the most recent RIs instead of sending one for every packet. Typically, the destination already sends data or acknowledgments, so the RI can be included in those packets without creating new ones. Upon receiving the RI, the source adjusts its rate accordingly. Today's transport protocols often struggle to distinguish between delay and loss, often mistaking timeouts for packet loss, leading to spurious retransmissions and reduced throughput. In contrast, FFS eliminates delayed packets, as it ensures low-latency forwarding. Moreover, the rate feedback lets the source distinguish congestion drops from random losses via the RI field's continuity, making it especially useful in wireless environments.

IX. Rate Feedback Evaluation

We implement and evaluate two feedback-based algorithms and compare them to BBR and Cubic.

Implementation. We use the same implementation and evaluation setup as in the FFS-C85 evaluation, which we extend with RI-capabilities at the router. Furthermore, we implement an application running at the destination endpoint that sends the received RI back to the source. The source implements two algorithms, MinTime and MinLoss:

MinTime. Sending at the maximum possible rate, i.e., at the rate corresponding to the source's egress capacity.

MinLoss. Starting at a low rate, the source adjusts its sending rate toward the RI learned from the destination.

Both algorithms provide reliable transport by retransmitting lost packets. MinLoss uses a leaky bucket [40] to stay close to the learned rate, optimized for allowing the source's CPU to sleep until the next packet can be sent. The source stores the 21 most recent RI values in a sliding window to compute a median, smoothing feedback fluctuations. In our evaluation, 1 or 2 streams transmit 1–10 MB data, and we measure file transfer time, as well as arrival- and drop rates at the router. We compare MinTime and MinLoss with BBR and Cubic.

Transfer Time and Drop Rate. Figure 5 shows MinTime achieves the shortest transfer time but incurs the highest loss rates, followed by BBR. MinLoss has the lowest loss rates, with 0% for single-stream transfers and 0.05%–1.01% for two-stream scenarios. Cubic exhibits higher loss rates, ranging from 0.58%–3.32%. While MinLoss outperforms Cubic in both transfer time and loss rate, it has slightly higher transfer times than BBR. The two-stream measurements highlight FFS's bandwidth isolation, ensuring fair rate allocation with comparable transfer times for both streams.

Convergence and Stability. We measure sending and drop rates at the router for two streams. Both streams transfer 10 MB, with the second stream starting two seconds after the first one, as shown in Figure 6. MinLoss only experiences packet loss when the second stream starts. At this point, the link is already saturated by the first stream and the router must first detect the sending rate of the second stream and then update τ accordingly. During this adjustment, the arrival rate exceeds the link capacity, causing losses. Cubic, BBR, and MinTime experience 2.7x, 30.2x, and 232x more drops than MinLoss. Rates are highly stable for FFS in combination with MinLoss: An FFS node benefits from burst-free sending rates provided by MinRate, enabling precise and consistent rate tracking as well as reliable τ estimation, while MinRate leverages the rate feedback to rapidly converge on a stream's fair rate, characterized by low fluctuations due to the stability of τ, and to reduce or even eliminate packet loss.

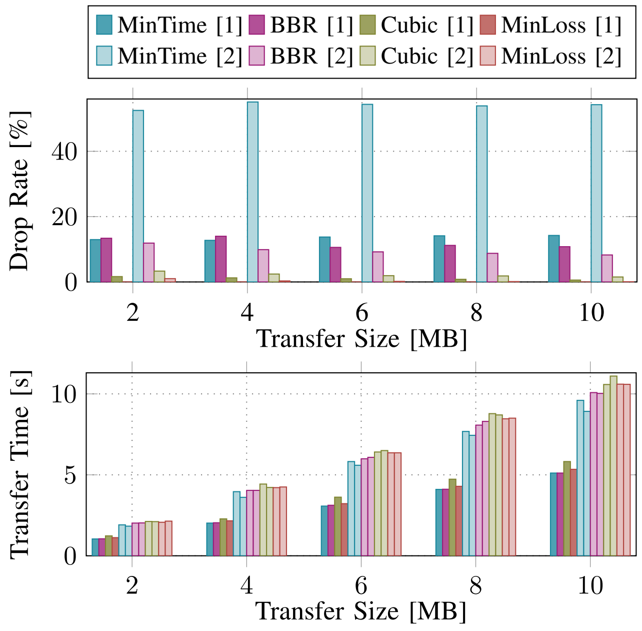


Figure 5: Rate feedback evaluation results. Numbers in square brackets denote concurrent streams.

X. Security

We now show how FFS prevents tampering and volumetric DDoS attacks and thereby achieves objective O4.

Threat Model. An on-path adversary, for instance a compromised router or destination host, can trivially undermine our security goals by modifying, delaying, or dropping packets. Therefore, the security properties of FFS hold only when a path consists entirely of honest and uncompromised nodes, a fundamental assumption shared by all secure communication systems. We place no restrictions on the capabilities or number of off-path adversaries. An off-path adversary can observe, modify, inject, drop, delay, and replay packets. Attackers can compromise entire (off-path) nodes, including infrastructure services and routers, and gain access to key material.

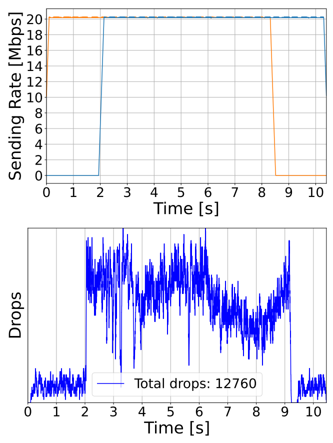
Tampering. An adversary can increase its forwarding rate only by increasing the forwarding probability of its packets. This probability is decoupled from IP addresses and thus robust to spoofing, but depends on the ratio of the packet labels $\boxed{\text{p.f}}$ and $\boxed{\text{p.r}}$ (Algorithm 1, Line 2). For instance, an attacker might inflate $\boxed{\text{p.f}}$ by a factor $X \gg 1$ or reduce $\boxed{\text{p.r}}$ by $1/X$. This behavior effectively attempts to inflate the FFS of its stream $\widehat{s}$, as indicated by the estimated sum of fair shares (Section VI): $\mathrm{f}^{\mathrm{pre}(n,\widehat{s})}_{\widehat{s}} = \mathrm{r}^{\mathrm{pre}(n,\widehat{s})}_{\widehat{s}} \times \boxed{\text{p.f}}/\boxed{\text{p.r}}$. However, this will also inflate $\mathrm{F}^{\mathrm{pre}(n,\widehat{s})}_{(i,j)}$, as $\mathrm{F}^{\mathrm{pre}(n,\widehat{s})}_{(i,j)} = \sum_{s\in(\mathrm{S}^{n}_{(i,j)}-\{\widehat{s}\})} \mathrm{f}^{\mathrm{pre}(n,\widehat{s})}_{s} + \mathrm{f}^{\mathrm{pre}(n,\widehat{s})}_{\widehat{s}}$. Normalization ensures a malicious node can only adjust the relative weights of its streams, without affecting the total FFS at the next node. In Section XI, we prove that a node maximizes egress throughput at its upstream node by treating its traffic as a single stream, setting $\boxed{\text{p.f}}$ equal to $\boxed{\text{p.r}}$. An AS could thus maximize total forwarded traffic, but this risks customer complaints since all streams, including those below their fair share, would be dropped equally.

Furthermore, off-path attackers could exploit rate feedback by spoofing the source IP, either impersonating the destination to send packets to the source, or impersonating the source to trigger reflected rate information from the destination. For connection-oriented protocols like TCP, off-path attacks are difficult due to the handshake, which defends against IP spoofing, and sequence numbers, which mitigate packet injection. A more secure approach is embedding RI feedback in a TLS connection or using systems like DRKey [41], [42], which derives symmetric keys on-the-fly to authenticate both the source and rate feedback at the network layer. Importantly, for use in FFS, any authentication protocol requires cryptographic operations solely at the endpoints, not at on-path nodes.

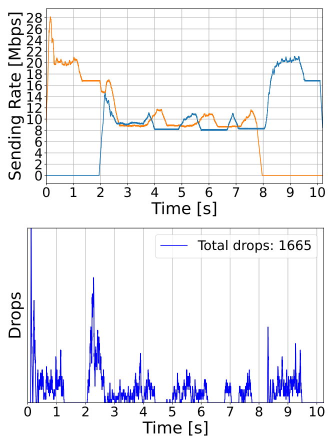
Volumetric DDoS. Under persistent DDoS attacks, continuous congestion causes legitimate flows to continually decrease their transmission rates in response to the congestion [6], [7]. FFS' bandwidth isolation mitigates this issue, as it ensures streams to retain their fair shares, even in the presence of volumetric traffic, such as link- and destination-flooding attacks. Although reallocating fairness matrix entries in response to DDoS may seem tempting, it could cause collateral damage if not all streams arriving at the same ingress interface are part of the attack. A more challenging class of volumetric attacks for FFS to mitigate are pulse-wave attacks, characterized by high-rate, short bursts typically lasting only a few seconds [12]. The key question is whether FFS' τ-tracking algorithm, adapted from (H)CSFQ, can effectively respond to such rapid on-off traffic patterns. Our high-speed evaluation (Section XII) shows that it can: τ stabilizes within tens to hundreds of milliseconds, even under aggressive conditions such as ramping from 0 to $40\,\mathrm{Gbps}$. FFS also offers some protection against volumetric attacks originating from botnets, although defending against botnets is inherently difficult for any system. The challenge stems less from the overall traffic volume and more from the large number of small, concurrent streams, which are often indistinguishable from legitimate traffic. If bots are regionally concentrated, FFS mitigates the attack through fairness matrix-based isolation. But even in the case of a globally distributed botnet (e.g., consisting of compromised IoT devices), FFS ensures that legitimate endpoints can still communicate: In Section XI, we formally prove that with FFS, each stream is guaranteed a minimum share of bandwidth, regardless of how many hosts an adversary controls within a botnet. A key insight of FFS is that it does not try to classify traffic as malicious or legitimate, a distinction that can be inherently difficult or impossible, but instead ensures that every stream receives its fair share of resources.

Others. FFS helps mitigate emerging attacks that exploit congestion probing. For instance, CrossPoint attacks [43] aim to uncover obfuscated Internet links by exploiting correlated increases in RTT and other metrics.[7] Similarly, SnailLoad [44]

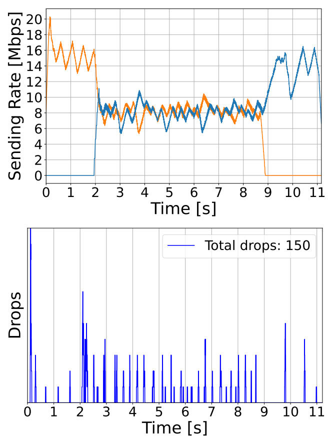
[7]Topology obfuscation defends against link-flooding attacks (LFAs). FFS inherently mitigates LFAs by enforcing fairness.



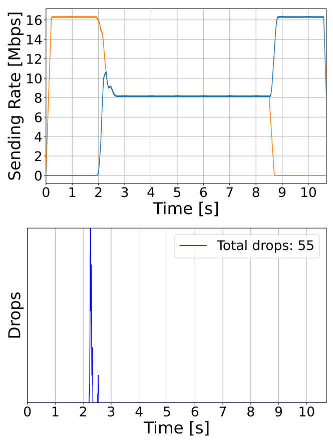
(a) MinTime



(b) BBR



(c) Cubic



(d) MinLoss

Figure 6: Source sending rate and router drop rate for MinTime, BBR, Cubic, and MinLoss (10 MB transfers).

uses latency as a side channel to monitor victim system activities, e.g., to perform video fingerprinting. By ensuring consistently minimal latency, FFS reduces the effectiveness of these latency-based probing attacks.

XI. Security proofs

This section formally proves that under FFS, every stream is guaranteed a worst-case minimum allocation, regardless of other streams' sending rates. We also show that if a greedy node attempts to maximize its throughput over a neighboring node, its best strategy is to concentrate traffic into a single stream. We begin by proving that $\tau \geq 1$, an intermediate result that will serve as a stepping stone for the main proofs.

Lemma 1. $\tau \geq 1$.

Proof. Consider egress interface j. If there is no congestion, by definition, $\tau = \infty$. In the presence of congestion, FFS ensures that $C = \sum_{i\in \mathrm{I}^n} \sum_{s\in \mathrm{S}^n_{(i,j)}} \mathrm{r}^n_s$, reformulating this gives:

$$\begin{aligned}C &= \sum_{i\in\mathrm{I}^n}\sum_{s\in\mathrm{S}^n_{(i,j)}} \mathrm{r}^n_s = \sum_{i\in\mathrm{I}^n}\sum_{s\in\mathrm{S}^n_{(i,j)}} \min(\mathrm{r}_s^{\mathrm{pre}(n,s)},\ \mathrm{f}^n_s\times\tau)\\ &\leq \sum_{i\in\mathrm{I}^n}\sum_{s\in\mathrm{S}^n_{(i,j)}} \mathrm{f}^n_s\times\tau = \tau\times\sum_{i\in\mathrm{I}^n}\sum_{s\in\mathrm{S}^n_{(i,j)}} \mathrm{f}^n_s = \tau\times\sum_{i\in\mathrm{I}^n}\mathrm{M}^n_{(i,j)}\\ &= \tau\times C\end{aligned}$$

Dividing both sides by C gives $\tau \geq 1$. This follows from applying Equation (2), the fact that min(a, b) $\leq$ b, that the sum of fractional fair shares of some interface-pair must be equal to the corresponding fairness entry, and that the sum of matrix entries is equal to the capacity. □

Lemma 1 implies that a flow's fair share at an egress cannot decrease, since it is scaled by $\tau \geq 1$. The flow's allocation can increase if other streams under-utilize their fair shares.

Lemma 2. *A stream s gets a worst-case minimum guaranteed allocation of* $\min(r_{init},\ \min_{x=0}^{\ell}(\frac{\prod_{t=0}^{x}\mathrm{M}^{n^t}_{(i^t,j^t)}}{\prod_{t=0}^{x-1}\mathrm{C}^{n^t}_{j^t}}))$*, with r_{init} referring to the source's sending rate, where this allocation holds even if every on-path interface is maximally congested.*

Proof. We first establish a lower bound for the FFS $\mathrm{f}_s^{n^x}$ of a stream's x-th on-path node:

$$\begin{aligned}\mathrm{f}_s^{n^x} &= \frac{\mathrm{f}_s^{n^{x-1}}}{\mathrm{F}^{n^{x-1}}}\cdot\mathrm{M}^{n^x} \geq \frac{\mathrm{f}_s^{n^{x-1}}}{\mathrm{C}_{j^{x-1}}}\cdot\mathrm{M}^{n^x} = \frac{\mathrm{f}_s^{n^{x-2}}}{\mathrm{F}^{n^{x-2}}}\cdot\frac{\mathrm{M}^{n^{x-1}}\cdot\mathrm{M}^{n^x}}{\mathrm{C}_{j^{x-1}}}\\ &\geq \frac{\mathrm{f}_s^{n^{x-2}}}{\mathrm{C}_{j^{x-2}}}\cdot\frac{\mathrm{M}^{n^{x-1}}\cdot\mathrm{M}^{n^x}}{\mathrm{C}_{j^{x-1}}} = \ \dots\ \geq \frac{\prod_{t=0}^{x}\mathrm{M}^{n^t}}{\prod_{t=0}^{x-1}\mathrm{C}^{n^t}}\end{aligned}$$

Here, we recursively applied Equation (1), $\mathrm{F}^{n^{x-1}} \leq \mathrm{C}_{j^{x-1}}$, and $\mathrm{f}_s^{n^0} = \mathrm{M}^{n^0}$. Next, we use this result to prove a lower bound for the stream's arrival rate at the destination $\mathrm{r}_s^{n^\ell}$.

$$\begin{aligned}\mathrm{r}_s^{n^\ell} &= \min(\mathrm{r}_s^{n^{\ell-1}},\ \mathrm{f}_s^{n^\ell}\cdot\tau_\ell)\\ &= \min(\min(\mathrm{r}_s^{n^{\ell-2}},\ \mathrm{f}_s^{n^{\ell-1}}\cdot\tau_{\ell-1}),\ \mathrm{f}_s^{n^\ell}\cdot\tau_\ell)\\ &= \min(\mathrm{r}_s^{n^{\ell-2}},\ \mathrm{f}_s^{n^{\ell-1}}\cdot\tau_{\ell-1},\ \mathrm{f}_s^{n^\ell}\cdot\tau_\ell)\\ &= \ \dots = \min(\mathrm{r}_{\mathrm{init}},\ \mathrm{f}_s^{n^0}\cdot\tau_0, \dots,\ \mathrm{f}_s^{n^\ell}\cdot\tau_\ell)\\ &\geq \min(\mathrm{r}_{\mathrm{init}},\ \mathrm{f}_s^{n^0}, \dots,\ \mathrm{f}_s^{n^\ell})\\ &\geq \min(\mathrm{r}_{\mathrm{init}},\ \min_{x=0}^{\ell}(\frac{\prod_{t=0}^{x}\mathrm{M}^{n^t}}{\prod_{t=0}^{x-1}\mathrm{C}^{n^t}}))\end{aligned}$$

Here, we additionally applied Equation (2) and Lemma 1. □

Corollary 1. *If traffic from ingress i to egress j consists of a single stream, then at least up to a rate of $M_{(i,j)}$ is guaranteed to be forwarded, irrespective of the rate and FFS distributions of streams originating from other ingresses.*

Proof. The guaranteed rate for traffic from interface x is:

$$\begin{aligned}\sum_{s\in\mathrm{S}^n_{(i,j)}}\mathrm{r}^n_s &= \mathrm{r}^n_s = \min(\mathrm{r}_s^{\mathrm{pre}(n,s)},\ \mathrm{f}^n_s\times\tau) \geq \min(\mathrm{r}_s^{\mathrm{pre}(n,s)},\ \mathrm{f}^n_s)\\ &= \min(\mathrm{r}_s^{\mathrm{pre}(n,s)},\ \mathrm{M}^n_{(i,j)})\end{aligned}$$

We first used that only a single stream exists from ingress i to egress j, then applied Equation (2) and Lemma 1, and finally

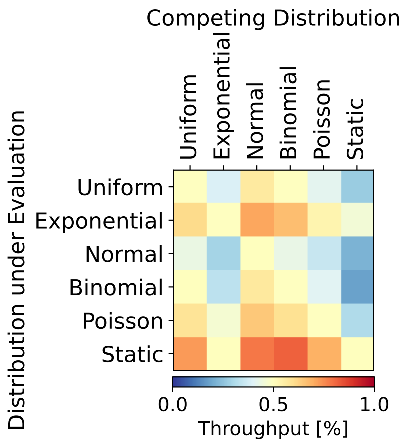


Figure 7: Throughput of two competing interfaces, each originating traffic with a certain distribution of $\boxed{\text{p.r}}$/$\boxed{\text{p.f}}$. The total traffic rate is the same for both ingress interfaces. If the interface under evaluation transmits using a static distribution, it consistently achieves at least 50 % of the overall throughput.

noted that the sum of fractional fair shares over an interface pair equals the corresponding fairness matrix entry. □

The proof can be trivially generalized to multiple streams, i.e., if each stream $s \in \mathrm{S}^{n}_{(i,j)}$ sends at least at a rate $\mathrm{r}^{\mathrm{pre}(n,s)}_{s} \geq \mathrm{f}^{n}_{s}$, then the total sum of bandwidth is at least $\mathrm{M}^{n}_{(i,j)}$, irrespective of the rate and FFS distributions of streams originating from other ingresses. In Figure 7, we experimentally verify that this strategy is optimal by evaluating two competing interfaces originating traffic with a certain FFS distribution.

XII. High-Speed Implementation

Implementation. We implement high-speed versions of both an FFS source endpoint and router using Intel's Data Plane Development Kit (DPDK) [45]. While the source endpoint is straightforward to implement at high speeds (primarily involving the creation of packets with FFS headers), the router demands more careful design considerations. Next, we therefore describe our implementation of Algorithm 1, which includes on-the-fly updates of τ (Appendix A).

To distribute incoming packets across available router cores, we leverage Receive Side Scaling (RSS). Each core is exclusively dedicated to a single port, and memory is allocated in a NUMA-aware manner to minimize access times. To prevent false sharing, we cache-align shared data structures. For random value generation, we use DPDK's `rte_rand()` function. Our implementation is entirely lock-free, which is achieved by ensuring that no variables are updated by multiple cores concurrently. For rate estimation, we take inspiration from exponential averaging [13], which updates rate R upon receiving a packet p of length len(p) as follows:

$$x \leftarrow \frac{t}{K}; \quad f \leftarrow \text{exp(-x)}; \quad R \leftarrow (1-f) \cdot \frac{\text{len(p)}}{t} + f \cdot R$$

Table IV: Average packet processing overhead at router.

Task	Time [ns]
Parse headers from received FFS packet	13
Update estimation of $\mathrm{R}^{n}_{(*,j)}$, $\mathrm{F}^{\mathrm{pre}(n,s)}_{(i,j)}$ and A_j	16
Update $\boxed{\text{p.f}}$, $\boxed{\text{p.r}}$, and $\boxed{\text{p.ri}}$	28
Compute forwarding probability γ	9
Update header fields	15
Others (sync. cores, timestamp, update τ)	21
Total	**102**

Here, t is the inter-arrival time of the last two packets, and K is a constant (we use K = 10^7). We optimize exponential averaging further for high-speed processing by approximating exp(-x) using 1/(1+x) for small values of x, which reduces computational overhead at high forwarding rates. For inter-arrival times below 1 ms, this approximation introduces only a 0.47% error, which is negligible in practice. For larger values of x (corresponding to low traffic rates) we revert to computing exp(-x) directly. This approach is feasible because computational overhead is less critical at low traffic rates.

Testbed. We evaluate the performance of both the source endpoint and the router by running them sequentially—never simultaneously—on a commodity machine equipped with an Intel Xeon processor running at 2.1 GHz. This machine is connected via four 40 Gbps bidirectional Ethernet links to a Spirent SPT-N4U device, which generates streams that are not CCA-controlled for the router evaluation. During the evaluation, the router reads incoming packets from its designated ports, processes them, and either drops or sends the processed packets back to the Spirent device. The Spirent device also provides functionality to randomize specific bytes within the packets, enabling the generation of random $\boxed{\text{p.f}}$ and $\boxed{\text{p.r}}$ values.

Results. Figure 8 shows source packet generation and router forwarding performance. For the source, a single CPU core can generate traffic at 40 Gbps with 600 B payloads, and the router achieves a 160 Gbps line rate even for packets with a 250 B payload, using only 10 cores. As expected, fewer cores are required for higher payloads. The high forwarding speed is thanks to the low per-packet processing overhead of 102 ns. This overhead is approximately 2.4x lower than Helia's [14] and 3x lower than Hummingbird's [26]. This gain stems from FFS' ability to provide secure forwarding without relying on computationally expensive operations such as key derivation, authentication, or stateful policing. Table IV provides a detailed breakdown of the FFS processing latency.

During our measurements, we observed that the tracking of τ performed well, rapidly converging to the optimal value. As a result, A_j fluctuated around C_j with a variance of approximately 100 Mbps. We further stress-tested the algorithm for estimating τ by generating random values for $\boxed{\text{p.f}}$ and $\boxed{\text{p.r}}$. Even under these conditions, the algorithm consistently derived τ such that $A_j \approx C_j$.

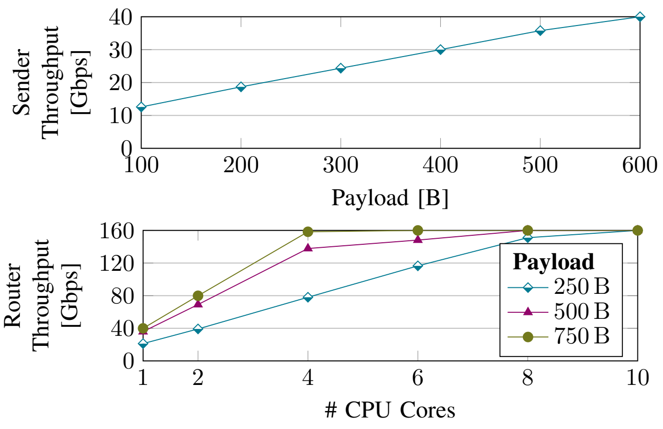


Figure 8: Throughput of sender (single core) and router.

XIII. Discussion

Future Work. First, to discourage overuse, an overusing stream could be assigned a lower forwarding probability, resulting in a rate lower than its fair share. For instance, the enforced rate could decrease more sharply as the stream's overuse intensifies. Adapting FFS to incorporate this feature is straightforward, as the τ computation naturally accommodates such adjustments. Future research should evaluate the practicality and effectiveness of such an incentive mechanism.

Second, while adding FFS to fixed-function hardware routers is impractical, programmable architectures like DPDK and P4 allow existing devices to be reprogrammed to support FFS. Therefore, future work could include implementing FFS in P4, also to demonstrate Tbps throughput, as even more complex systems such as HCSFQ have already been successfully implemented in P4. Additionally, FFS could be extended to multipath protocols or integrated into existing CCAs like Cubic or BBR, to facilitate incremental deployment.

Limitations. Discovering the rate through feedback inherently involves at least one round-trip time (RTT). While this delay is relatively small, a node located closer to the source could, in theory, provide the source with rate information more quickly. Notifying the source at the first congested node would require on-path nodes to authenticate the feedback, which introduces additional deployment challenges. Therefore, FFS avoids this approach and relies solely on the destination to authenticate rate feedback. Nonetheless, FFS' low-latency design ensures that rate feedback delay stays close to one RTT and does not increase substantially even under congestion.

Also, FFS relies on several approximations that may introduce some inaccuracies. For instance, FFS nodes estimate rates between interfaces and calculate τ reactively based on observed traffic, assuming that future traffic patterns will follow past behavior. As a result, the rate feedback provided to endpoints may fluctuate. These approximations are important to consider in implementations, where techniques such as filtering or smoothing of rate feedback at the source help mitigate such fluctuations. Our evaluation in Section IX demonstrates that these challenges can be effectively managed and do not compromise the effectiveness of FFS' rate feedback mechanism. We do not claim to achieve the fine-grained fairness of many-queue systems that guarantee perfect fairness. Instead, we willingly accept the trade-off of providing only approximate fairness in exchange for greater scalability, enhanced security, and easier deployment.

Configuration and Deployment. FFS can be incrementally deployed at endpoints and nodes, as discussed in Appendix C. Label range and precision influence the accuracy of the provided fair rates. For Internet deployment, allocating 2 B per field should be sufficient (e.g., using binary16 [46]). Appendix D shows how FFS' size overhead can be substantially reduced. We cover deployment incentives in Appendix E and further header considerations in Appendix F.

XIV. Conclusion

FFS provides secure and scalable bandwidth allocation and isolation on the Internet, while minimizing barriers to deployment by making minimal assumptions, imposing few requirements, and avoiding reliance on external systems. Its network-enforced notion of fairness mitigates volumetric DDoS attacks and addresses key issues in existing CCAs, such as RTT-dependent unfairness and differences in CCA aggressiveness. As such, FFS marks a significant step toward enabling meaningful communication guarantees on the global Internet by offering a practical solution where previous systems have remained mostly academic. Its significantly easier implementation and deployment increase its potential to bridge the gap between academic research and real-world adoption.

Our results highlight several surprising aspects of FFS:

- FFS contributes to fairer bandwidth allocations across all tested CCAs while keeping latency and jitter low.
- FFS ensures constant overheads in packet processing time, memory, number of queues, and packet label sizes, making it suitable for large-scale networks.
- FFS does not require trust between nodes because the normalization step mitigates tampering with FFS fields.
- FFS is inherently robust against source address spoofing attacks without needing cryptographic authentication between on-path nodes because the forwarding probability is independent of IP addresses.
- The flexible granularity and independence of streams make FFS suitable for multi-path protocols and inherently support traffic protection in both directions.

This work establishes the foundations of FFS and opens up exciting opportunities for future research, including further optimizations and integration with real-world systems.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback. We appreciate the financial support received from ETH Zurich, the Zurich Information Security and Privacy Center (ZISC), and armasuisse Science and Technology.

References

[1] S. Scherrer, M. Legner, A. Perrig, and S. Schmid, "Model-based insights on the performance, fairness, and stability of bbr," in *IMC*, 2022.

[2] A. A. Philip, R. Ware, R. Athapathu, J. Sherry, and V. Sekar, "Revisiting tcp congestion control throughput models & fairness properties at scale," in *Proceedings of the 21st ACM Internet Measurement Conference*, ser. IMC '21. Association for Computing Machinery, 2021.

[3] T. Kozu, Y. Akiyama, and S. Yamaguchi, "Improving rtt fairness on cubic tcp," in *CANDAR*, 2013.

[4] N. Dukkipati, "Rate control protocol (rcp): congestion control to make flows complete quickly," Ph.D. dissertation, 2008.

[5] K. Sasaki, M. Hanai, K. Miyazawa, A. Kobayashi, N. Oda, and S. Yamaguchi, "Tcp fairness among modern tcp congestion control algorithms including tcp bbr," in *CloudNet*, 2018.

[6] M. Mathis *et al.*, "The macroscopic behavior of the tcp congestion avoidance algorithm," *CCR*, 1997.

[7] Neal Cardwell, "BBR v2: A model-based congestion control," http://tinyurl.com/25khc6et, 2024.

[8] K. Sanjta, "Path Quality Part 1: The Surprising Impact of 1% Packet Loss," https://www.thousandeyes.com/blog/path-quality-surprising-impact-one-percent-packet-loss, 2024.

[9] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson, "Tcp congestion control with a misbehaving receiver," *SIGCOMM CCR*, 1999.

[10] L. Brown *et al.*, "On the future of congestion control for the public Internet," ser. HotNets, 2020.

[11] B. Briscoe, K. D. Schepper, M. Bagnulo, and G. White, "Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture," RFC 9330, 2023.

[12] A. G. Alcoz, M. Strohmeier, V. Lenders, and L. Vanbever, "Aggregate-based congestion control for pulse-wave ddos defense," in *SIGCOMM*, 2022.

[13] I. Stoica, S. Shenker, and H. Zhang, "Core-stateless fair queueing: achieving approximately fair bandwidth allocations in high speed networks," *SIGCOMM CCR*, 1998.

[14] M. Wyss *et al.*, "Protecting critical inter-domain communication through flyover reservations," in *CCS*, 2022.

[15] M. Wyss and A. Perrig, "Zero-setup intermediate-rate communication guarantees in a global internet," in *USENIX Security*, 2024.

[16] L. Brown, A. G. Alcoz, F. Cangialosi, A. Narayan, M. Alizadeh, H. Balakrishnan, E. Friedman, E. Katz-Bassett, A. Krishnamurthy, M. Schapira, and S. Shenker, "Principles for internet congestion management," in *SIGCOMM*, 2024.

[17] Z. Yu, J. Wu, V. Braverman, I. Stoica, and X. Jin, "Twenty years after: Hierarchical Core-Stateless fair queueing," in *NSDI*, 2021.

[18] K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," IETF, RFC 3168, Sep. 2001. [Online]. Available: http://tools.ietf.org/rfc/rfc3168.txt

[19] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *SIGCOMM*, 2002.

[20] J. Chou *et al.*, "Proactive surge protection: a defense mechanism for bandwidth-based attacks," ser. USENIX Security, 2008.

[21] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker, "Approximate fairness through differential dropping," *SIGCOMM CCR*, 2003.

[22] G. Giuliari *et al.*, "Colibri: A cooperative lightweight inter-domain bandwidth-reservation infrastructure," ser. CoNEXT, 2021.

[23] M. Wyss *et al.*, "Secure and scalable QoS for critical applications," ser. IWQoS, 2021.

[24] T. Weghorn, S. Liu, C. Sprenger, A. Perrig, , and D. Basin, "N-Tube: Formally verified secure bandwidth reservation in path-aware internet architectures," in *CSF*, 2022.

[25] G. Giuliari, M. Wyss, M. Legner, and A. Perrig, "Gma: A pareto optimal distributed resource-allocation algorithm," in *SIROCCO*, 2021.

[26] K. Wüst *et al.*, "Hummingbird: Fast, flexible, and fair inter-domain bandwidth reservations," 2024. [Online]. Available: https://arxiv.org/abs/2308.09959

[27] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time," *Queue*, 2016.

[28] A. A. Philip, R. Ware, R. Athapathu, J. Sherry, and V. Sekar, "Revisiting tcp congestion control throughput models & fairness properties at scale," in *IMC*, 2021.

[29] Netfilter, "The netfilter.org libnetfilter_queue project," https://netfilter.org/projects/libnetfilter_queue/, 2024.

[30] S. community, "Scapy," https://scapy.net/, 2024.

[31] M. P. Contributors, "Mininet: An Instant Virtual Network on Your Laptop (or Other PC)," http://mininet.org/, 2024.

[32] ESnet, "iperf3 documentation," http://software.es.net/iperf/, 2024.

[33] G. LLC, "The Go programming language," https://go.dev/, 2023.

[34] C. for Applied Internet Data Analysis, "AS Relationships," https://www.caida.org/catalog/datasets/as-relationships-geo/, 2023.

[35] A. D. Broido and A. Clauset, "Scale-free networks are rare," *Nature communications*, 2019.

[36] L. Saino, C. Cocora, and G. Pavlou, "A toolchain for simplifying network simulation setup," in *ICST*, 2013.

[37] D. Alderson, H. Chang, M. Roughan, S. Uhlig, and W. Willinger, "The many facets of internet topology and traffic," 2006.

[38] mathsfromnothing, "Brent's method," https://mathsfromnothing.au/brents-method/, 2025.

[39] A. Agarwal, V. Arun, D. Ray, R. Martins, and S. Seshan, "Towards provably performant congestion control," in *NSDI*, 2024.

[40] A. Tanenbaum, *Computer Networks*, 4th ed. Prentice Hall Professional Technical Reference, 2002.

[41] T. H.-J. Kim, C. Basescu, L. Jia, S. B. Lee, Y.-C. Hu, and A. Perrig, "Lightweight source authentication and path validation," in *SIGCOMM*, 2014.

[42] B. Rothenberger, D. Roos, M. Legner, and A. Perrig, "PISKES: Pragmatic Internet-scale key-establishment system," in *ASIACCS*, 2020.

[43] X. Huang, K. Xue, L. Chen, M. Ai, H. Zhou, B. Luo, G. Gu, and Q. Sun, "You can obfuscate, but you cannot hide: CrossPoint attacks against network topology obfuscation," 2024.

[44] S. Gast, R. Czerny, J. Juffinger, F. Rauscher, S. Franza, and D. Gruss, "SnailLoad: Exploiting remote network latency measurements without JavaScript," 2024.

[45] DPDK Project, "Data Plane Development Kit," https://dpdk.org, 2025.

[46] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, 2019.

[47] J. Kumar, S. Anubolu, J. Lemon, R. Manur, H. Holbrook, A. Ghanwani, D. Cai, H. Ou, Y. Li, and X. Wang, "Inband Flow Analyzer," Internet Engineering Task Force, Internet-Draft draft-kumar-ippm-ifa-08, 2024, work in Progress.

[48] T. P. A. W. Group, "In-band network telemetry (int) dataplane specification," https://p4.org/p4-spec/docs/INT_v2_1.pdf, 2020.

[49] F. Gont, "Security Assessment of the Internet Protocol Version 4," IETF, RFC 6274, Jul. 2011. [Online]. Available: http://tools.ietf.org/rfc/rfc6274.txt

[50] K. Nichols, S. Blake, F. Baker, and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," IETF, RFC 2474, Dec. 1998. [Online]. Available: http://tools.ietf.org/rfc/rfc2474.txt

[51] T. I. C. C. R. Group, "Internet congestion control research group iccrg," https://www.irtf.org/iccrg.html, 2025.

[52] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, and M. Yu, "Hpcc: high precision congestion control," in *SIGCOMM*, 2019.

[53] W. Wang, M. Moshref, Y. Li, G. Kumar, T. S. E. Ng, N. Cardwell, and N. Dukkipati, "Poseidon: Efficient, robust, and practical datacenter CC via deployable INT," in *NSDI*, 2023.

[54] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, "Homa: a receiver-driven low-latency transport protocol using network priorities," in *SIGCOMM*, 2018.

[55] S. Agarwal, Q. Cai, R. Agarwal, D. Shmoys, and A. Vahdat, "Harmony: A congestion-free datacenter architecture," in *NSDI*, 2024.

[56] S. Arslan, Y. Li, G. Kumar, and N. Dukkipati, "Bolt: Sub-RTT congestion control for Ultra-Low latency," in *NSDI*, 2023.

[57] P. Goyal, P. Shah, K. Zhao, G. Nikolaidis, M. Alizadeh, and T. E. Anderson, "Backpressure flow control," in *NSDI*, 2022.

[58] N. K. Sharma, M. Liu, K. Atreya, and A. Krishnamurthy, "Approximating fair queueing on reconfigurable switches," in *NSDI*, 2018.

[59] P. Gao, A. Dalleggio, Y. Xu, and H. J. Chao, "Gearbox: A hierarchical packet scheduler for approximate weighted fair queuing," in *NSDI*, 2022.

APPENDIX

A. Computing τ

We update τ_j at egress interface j on the fly using the following expression, adapted from the CSFQ paper [13] (symbols mapped to match our notation):

$$\tau_j \leftarrow \tau_j \times \frac{C_j + \delta}{A_j + \delta}$$

Here, C_j denotes the capacity of egress j, and A_j is the rate of outgoing (i.e., admitted) traffic (see Table I). The intuition is as follows. When $A_j > C_j$, meaning that FFS admits more traffic than the link can support, τ_j is decreased, which in turn lowers the streams' fair rates (see Equation (2)). Conversely, when $A_j < C_j$, indicating that the link is underutilized, τ_j increases, raising the streams' current fair rates. We introduce damping parameter δ to smooth τ_j updates and enhance stability.

Performance. The computational effort to derive τ is constant, and in particular does not increase with the number of streams or interfaces. In practice, τ is not recomputed on every packet arrival; instead, it is updated periodically, either after a given number of packets or after a minimum time interval. We adopt this strategy in our multi-core DPDK implementation (Section XII), where a single core computes τ once every $1\,\mathrm{ms}$. This yields an amortized per-packet cost well below $1\,\mathrm{ns}$, which is insignificant compared to the total per-packet processing time of $102\,\mathrm{ns}$ (Table IV). This policy works well in high-rate environments, where traffic aggregates are sufficiently stable at the millisecond scale. In contrast, our Python prototype operates at lower data rates, where traffic can exhibit considerable burstiness: a small number of large packets may disproportionately influence observed throughput. To handle these conditions, we refine the update procedure so that τ can react swiftly to major changes in sending rates. Concretely, we maintain auxiliary candidate values near τ (above and below) together with their associated hypothetical admission rates. When a sudden change occurs, this structure allows us to move quickly to an appropriate new value of τ. This additional tracking incurs overhead, resulting in a noticeably higher (both absolute and relative) computational cost for deriving τ in the Python prototype. Finally, we stress that only in the network utilization experiments of Section VII-B do we determine τ via numerical optimization (Brent's method [38]). This is feasible in the simulator because it has global knowledge of all streams. That setting is intended to analyze Internet-scale allocation outcomes rather than packet-level dynamics, making numerical search–based computation of τ appropriate in that context.

B. Detailed Bandwidth Isolation Results

Figure 9 provides a visual overview of the isolation results for each pair of competing CCAs. The evaluated CCAs are bbr, bic, cdg, cubic, highspeed, htcp, hybla, illinois, lp, nv, scalable, vegas, veno, westwood, and yeah. The rows and columns of the matrices in the figure follow this order, starting from the top left. *While the figure shows results for each CCA pair, the key insights emerge from the overall trends.* It is observable, for example, that throughput is unevenly distributed under FIFO, whereas FFS-C100 and (H)CSFQ generally move allocations toward a more equal distribution, though not for all CCA combinations. With FFS-C85, two competing CCAs consistently get roughly equal bandwidth.

C. Incremental Deployment

FFS can be deployed incrementally, with support for partial adoption among network nodes and end hosts.

Network Deployment. Fairness and isolation improve as FFS-enabled nodes increase. If all congested nodes support FFS and non-congested nodes preserve FFS fields, a stream's fair share is fully protected. In early deployment, with a signle on-path FFS node (or when labels are removed between FFS nodes), FFS operates like RCS-2020 [10] and PSP [20], enforcing fairness only locally. As a result, even the very first adopter benefits from improved fairness and enhanced DDoS resilience. As deployment grows, labels are trustlessly propagated hop-by-hop, gradually achieving better CCA isolation.

We evaluate the security effectiveness of incremental in-network deployment of FFS by simulating the top 2000 Tier-1 and Tier-2 ASes from the CAIDA AS relationships dataset (as described in Section VII-B). We define the security effectiveness of a path as the fraction of on-path nodes that implement fairness and isolation through FFS, which reflects the path's resilience against an attacker attempting to congest any of its nodes. We study the impact of incremental deployment by varying the fraction of nodes (randomly sampled among the 2000 ASes) that support FFS. Our focus is on the distribution of path-level security effectiveness under these deployment scenarios. Figure 10 presents the CDF of this distribution. As expected, increasing FFS deployment improves path protection. For example, when 40% of nodes implement FFS, 30% of paths achieve 33% protection, 10% achieve 50% protection, 28% reach 66%, and 3% even attain 75% protection.

End Host Deployment. The endpoint components of FFS (measuring sending rates and initiating packet labels, or optionally applying the MinTime or MinLoss algorithms) can be incrementally deployed at individual hosts. Crucially, FFS enforces allocation and isolation guarantees within the network itself, so security does not rely on correct behavior from end hosts. Stream initiation rates do not necessarily need to be measured by the application; they may instead be inferred by the operating system or by an edge router. FFS headers can also be added deeper within the network at the first FFS-enabled node, where packets without FFS headers can be assigned default labels (r = 1.0, f = 1.0), after which normalization ensures consistency with packets that do include FFS headers. The feasibility depends on the concrete implementation: for example, if FFS uses the TOS field, no additional header space is needed, avoiding MTU issues. Alternatively, a router can treat packets without FFS headers as having implicit values of r = 1.0 and f = 1.0, without adding a header. The limitation of this approach is that fairness cannot propagate across FFS nodes, since each subsequent FFS router will reset these values to 1.0. When an

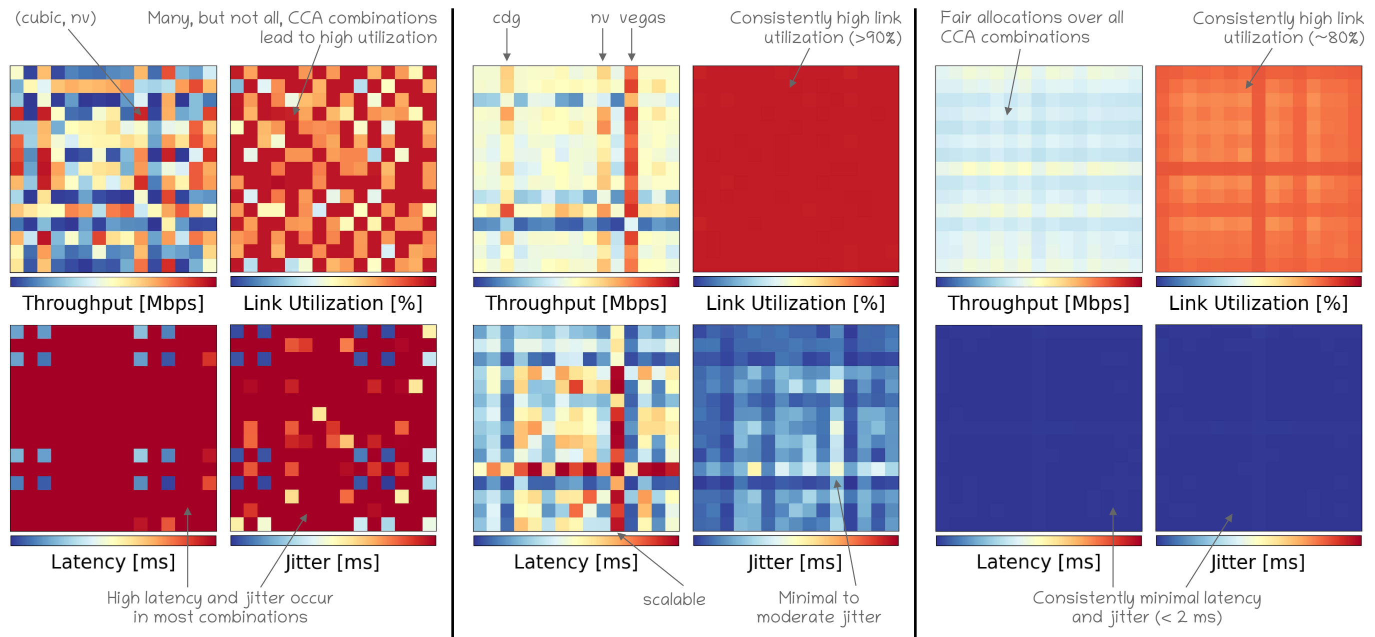


Figure 9: CCA isolation results. Each colored square in a grid corresponds to a CCA competing against another CCA. For all three subfigures, value ranges are always [0, 20] Mbps, [20, 200] ms, and [0, 200] ms, for throughput, RTT, and jitter.

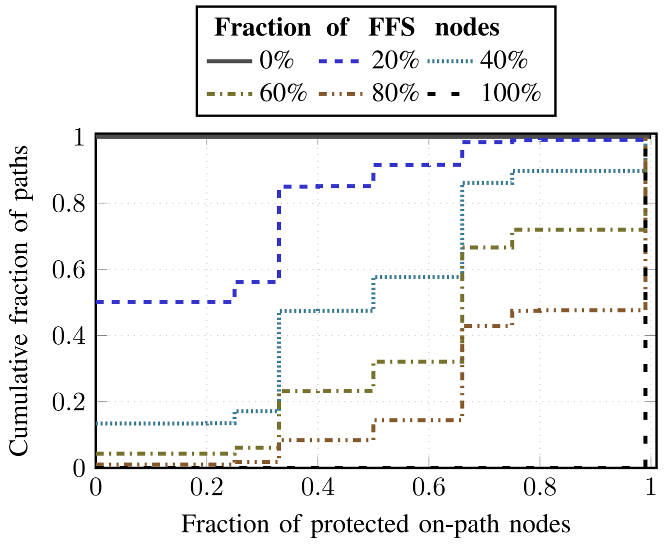


Figure 10: CDF of path protection for different levels of FFS deployment. Greater FFS participation shifts the curves rightward, reflecting better path protection.

FFS node receives packets without FFS headers, by assigning default labels it conceptually groups them into a single stream aggregate per interface pair. This aggregate receives its fair bandwidth share, but if demand exceeds that share, drops occur across all underlying streams. In this case, the node cannot distinguish among individual streams, so losses are distributed uniformly across the aggregate.

Impact on RI. Incremental network deployment affects how $\boxed{\text{p.RI}}$ should be interpreted. If a sender transmits at the rate indicated by $\boxed{\text{p.RI}}$ but observes packet loss, the loss could stem from either (i) congestion at an FFS-unaware on-path node, or (ii) random loss. Since the sender cannot distinguish between these cases, it must take a conservative approach by reducing its rate to prevent congestion at FFS-unaware nodes. During incremental deployment, algorithms such as MinLoss and MinTime (Section IX) therefore cannot be used. Instead, traditional congestion control algorithms (CCAs) like BBR and Cubic remain necessary, as they can respond to increased loss or RTT. However, RI-based feedback can still provide guidance. Specifically, RI-based feedback can be used to decrease the rate in response to overuse but not to increase the rate when underutilization is detected.

D. *Relative Rates*

To optimize the space used for packet labels, an alternative approach is to encode shorter labels instead of including $\boxed{\text{p.r}}$, $\boxed{\text{p.f}}$, and potentially $\boxed{\text{p.RI}}$. Specifically, this involves encoding the following compact labels: $\boxed{\text{p.rf}}$, which represents the ratio between $\boxed{\text{p.r}}$ and $\boxed{\text{p.f}}$, and $\boxed{\text{p.RRI}}$, which stands for the *Relative Rate Indicator (RRI)* and indicates by which factor the source's rate should be adapted (increased or decreased).

While this method substantially reduces the space required for packet labels, it comes at the cost of increased state and computational overhead, as well as potentially reduced precision in the learned rate at the source.

Label Size. By encoding a single relative rate instead two absolute rates, the label size can be drastically reduced.

With, e.g., *logarithmic encoding*, it is possible to encode the values with a certain relative spacing between them. For a given range $RF_{min} \leq RF \leq RF_{max}$ we can use a

Algorithm 3: Adapted version of Algorithm 1 working on a single label p.rf instead of labels p.r and p.f.

```
1 $\boxed{\text{p.rf}} \leftarrow \boxed{\text{p.rf}} \cdot \frac{\mathrm{F}^{\mathrm{pre}(n,s)}_{(i,j)}}{\mathrm{M}^{n}_{(i,j)}}$
2 $\gamma \leftarrow \min(1,\ \tau\ /\ \boxed{\text{p.rf}})$
3 $\boxed{\text{p.rf}} \leftarrow \boxed{\text{p.rf}} \times \gamma$
4 Call Algorithm 4 (for rate feedback)
5 Forward packet with probability $\gamma$, otherwise drop it.
```

Algorithm 4: Adapted version of Algorithm 2 working on p.RRI instead of p.RI.

```
1 if $\boxed{\text{p.RRI}}$ ¡ 1 then
      // Congestion previously on the path
2     $\boxed{\text{p.RRI}} \leftarrow \boxed{\text{p.RRI}} \cdot \gamma$
3 else
      // No congestion at previous nodes
4     if $R_* > C$ then
          // Congestion at egress interface
5         $\boxed{\text{p.RRI}} \leftarrow \min(\boxed{\text{p.RRI}},\ (\tau\ /\ \boxed{\text{p.rf}}))$
6     else
          // No congestion at egress interface
7         RRI = $\max(\frac{C}{R_*},\ 1\ /\ \boxed{\text{p.rf}})$
8         $\boxed{\text{p.RRI}} \leftarrow \min(\boxed{\text{p.RRI}},\ \text{RRI})$
```

logarithmic transformation ($\widehat{RF} = \log_b(RF)$ with base b) to get the corresponding range $\widehat{RF}_{min} \leq \widehat{RF} \leq \widehat{RF}_{max}$. After normalizing the transformed RF using $N = \frac{\widehat{RF} - \widehat{RF}_{min}}{\widehat{RF}_{max} - \widehat{RF}_{min}}$, the normalized value can be encoded to binary (with n-bit precision) using $B = \lfloor N \cdot (2^n - 1) \rceil$ ($B \in [0, 2^n - 1]$).

In summary, the function for encoding RF in n bits with relative spacing is as follows:

$$B = \lfloor \frac{\widehat{RF} - \widehat{RF}_{min}}{\widehat{RF}_{max} - \widehat{RF}_{min}} \cdot (2^n - 1) \rceil \tag{3}$$

Getting back RF from B can be achieved as follows:

$$RF = b^{\frac{B \cdot (\widehat{RF}_{max} - \widehat{RF}_{min})}{2^n - 1} + \widehat{RF}_{min}} \tag{4}$$

This can be further adapted to *piecewise* logarithmic encoding, e.g., to allocate more bits to represent values that are more important or require higher precision, in our case around 1.0, while using fewer bits for values at the extremes, i.e., for very small or very large values.

Overhead. Performing all computations based on relative rates requires the source to keep a history of its past sending rates. When the source receives the relative feedback (p.RRI) via notification from the destination, it has to look up the rate 1 RTT ago, and multiply it with the RRI to compute its end-to-end fair rate. For this, the source has to keep additional state in form of a history of its sending rates for a duration of at least 1 RTT. If our system is deployed at every possible point of congestion, all packets are forwarded with minimal latency and jitter, thus the RTT remains constant and looking up past rates becomes predictable and precise. But in an incremental deployment scenario, latency and jitter may be unpredictably high, thus making it challenging for the source to fetch the correct rate corresponding to the received RRI, which might lead to the source learning an incorrect rate.

Adaptations. We define rf^n_s as the ratio $\mathrm{r}^n_s / \mathrm{f}^n_s$ and adapt the operations in Algorithm 1 to process packets labeled with $\boxed{\text{p.rf}}$, replacing the separate labels $\boxed{\text{p.r}}$ and $\boxed{\text{p.f}}$. These changes are reflected in Algorithm 3. The update of $\boxed{\text{p.f}}$ in Line 1 of Algorithm 1 is applied to the denominator of $\boxed{\text{p.rf}}$. Similarly, for the computation of γ in Line 2, the division of $\boxed{\text{p.r}}$ by $\boxed{\text{p.f}}$ is replaced by taking the inverse of $\boxed{\text{p.rf}}$. Finally, the update of $\boxed{\text{p.r}}$ in Line 3 is applied to the numerator of $\boxed{\text{p.rf}}$.

The process for updating $\boxed{\text{p.RRI}}$ is detailed in Algorithm 4. If $\boxed{\text{p.RRI}} < 1$, this indicates that the packet's stream has encountered congestion at a previous node along its path, and the source must adjust its sending rate by the factor $\boxed{\text{p.RRI}}$. Consequently, any subsequent node on the path should only reduce $\boxed{\text{p.RRI}}$ further, reflecting the extent to which the stream exceeds its fair rate, as shown in Line 2. On the other hand, if $\boxed{\text{p.RRI}} \geq 1$, it implies that previous nodes have indicated the source can increase its sending rate. The current node then updates $\boxed{\text{p.RRI}}$ based on whether congestion exists at its egress interface. In the event of congestion, the node calculates in Line 5 the extent to which the stream's rate could be increased to achieve its fair rate. If there is no congestion, the node determines in Line 8 the potential increase in the stream's rate, accounting for the fact that this feedback will guide the source of every stream.

Lastly, the estimation of $\mathrm{F}^{\mathrm{pre}(n,s)}_{(i,j)}$ can be modified by replacing the computation $\text{p.len}\cdot(\boxed{\text{p.f}}\ /\ \boxed{\text{p.r}})$ with $\text{p.len}\cdot(1\ /\ \boxed{\text{p.rf}})$.

We implemented Algorithm 3 and Algorithm 4 and verified their correct operation in the Mininet setup from Section VII.

E. Deployment Incentives

FFS provides strong incentives for deployment by benefiting both endpoints (applications, end users) and Internet Service Providers (ISPs).

For ISPs, FFS offers the following advantages:

- Maintaining communication availability for customers even during volumetric DDoS attacks; if a customer's communication gets disrupted, the disruption can be attributed to factors outside the ISP's domain.
- Facilitating end-to-end traffic isolation without requiring trust in other network providers.
- Enabling customers to use the MinTime or MinLoss algorithms that minimize transfer time or loss, respectively.
- Creating new monetization opportunities, such as allowing customers to bid for higher allocations by adjusting fairness matrix entries.

- Improving latency characteristics and easing buffer pressure, which can allow routers to operate with shallower buffers and potentially reduce hardware requirements.
- Scaling across a range of deployment environments.

For endpoints, FFS offers the following advantages:

- Benefiting from fairness and traffic isolation without requiring changes to existing systems, as current CCAs can continue to operate.
- Maintaining communication availability even in the presence of DDoS (or any other) traffic.
- Experiencing low-latency, low-jitter forwarding that reduces the bandwidth-delay product and thus the buffering required at senders.
- Using MinTime and MinLoss to achieve high throughput or low loss when FFS is fully deployed along the forwarding path; these algorithms operate with negligible overhead compared to reservation-based isolation approaches.
- Obtaining higher allocations through provider agreements that map to larger entries in the fairness matrix.

F. FFS Header

Determining which header fields to utilize, which headers to extend, or whether to introduce new headers is not a challenge specific to FFS but a general issue faced by all secure isolation mechanisms. FFS simplifies this design space, since it only requires a low, constant number of bytes to encode $\boxed{\text{p.r}}$ and $\boxed{\text{p.f}}$ (and optionally $\boxed{\text{p.RI}}$). The encoding should satisfy three key requirements: (i) efficiency, (ii) ease of parsing, and (iii) optionally also compatibility with FFS-unaware nodes, ensuring they forward traffic without removing the FFS header. These requirements align with those addressed in prior work [4], [13]–[15], [17], [19], [22], [23], [26], [47], [48]. Accordingly, FFS can leverage existing approaches such as dedicated protocol headers for IPv4 and IPv6, hop-by-hop extension headers (e.g., in IPv6 or SCION), packet trailers, MPLS or TCP headers, or encapsulation protocols like GRE or VXLAN. Although IPv4 options are straightforward to parse using tools like DPDK or P4, their use is discouraged. In today's networks, IPv4 options often require processing by router CPUs, which is slow, or the packets are directly dropped due to security concerns [49]. Supporting FFS headers for only a subset of packets (e.g., IPv6 but not IPv4) necessitates isolating FFS-encoded packets from others, such as by using a dedicated non-FFS queue. Furthermore, if an endpoint does not append an IP extension, but the first on-path node does, the node must ensure a lower MTU was previously announced to accommodate the extension. We also considered using the TOS field in our design, but ultimately decided against it, as we could not guarantee that other protocols would not also utilize this field (e.g., RFC 2474 [50]).

G. Further Related Work

The most closely related work is discussed in Section III, and the Internet Congestion Control Research Group (ICCRG) [51] serves as a valuable resource for further exploration of Internet congestion control research.

This section focuses on work related to FFS's bandwidth isolation, but particularly its rate feedback mechanism. The systems described here are primarily designed for closed, fully controlled environments, such as data centers, rather than deployments on the public Internet. These approaches rely on assumptions that are impractical at Internet scale, such as benign behavior or full support from all switches and clients. As they can assume complete control over clients and can rely on their compliance, many of these protocols do not implement in-network isolation.

DiffServ [50] classifies and manages traffic by utilizing a field in the IP header to prioritize specific types of data flows, but its effectiveness is limited to benign and controlled settings, and it works only when latency-critical communication constitutes a minority of the traffic. HPCC [52] leverages fine-grained Inband Network Telemetry (INT) information from switches, such as queue sizes, to enable clients to compute precise flow rates, aiming for max-min fairness. It achieves high utilization and near-zero latency by controlling the number of in-flight bytes and applying packet pacing. Also Poseidon [53] utilizes INT. It achieves low queuing delays, high throughput, and fast convergence while ensuring max-min fairness across traffic patterns. Homa [54] implements receiver-driven congestion control, while Agarwal [55] eliminates congestion-related packet drops entirely and ensures bounded network delays. Bolt [56] reacts to congestion faster than one round-trip time (RTT) by predicting flow completions and quickly occupying released bandwidth. BFC [57] approximates per-hop, per-flow flow control with limited switch state and a modest number of switch queues, requiring state and dedicated queues only for active flows with queued packets. However, its performance degrades when the number of active flows exceeds the available queues. Similarly, AFQ [58] approximates FQ by using a few queues to emulate many, storing per-flow counters in a count-min sketch, but lacks support for HFQ. Gearbox [59] approximates weighted max-min fairness.