SpinTrap: Catching Speeding QUIC Flows

Ike Kunze, Constantin Sander, Lars Tissen, Benedikt Bode, Klaus Wehrle
Chair of Communication and Distributed Systems, RWTH Aachen University, Aachen, Germany
{kunze, sander, tissen, bode, wehrle}@comsys.rwth-aachen.de

***Abstract*—The resilience of the Internet to high traffic loads fundamentally relies on hosts responding to congestion, i.e., that they back off when the network is overloaded. Despite the corresponding wide-spread deployment of congestion control, unresponsive hosts still represent a danger and can wipe out all benefits of modern congestion management approaches, such as L4S. Hence, identifying (and isolating) unresponsive flows can contribute to improving the Internet's resilience. Yet, existing approaches only provide broad or probabilistic solutions which become inapplicable with QUIC or also harm benign traffic.**

In this paper, we propose *SpinTrap*, a speed trap for Internet flows designed to identify unresponsive traffic. Leveraging the QUIC spin bit, *SpinTrap* first monitors the sending behavior of QUIC flows before assessing their congestion responsiveness by checking for reduced sending rates as reaction to congestion signals (packet loss and ECN markings). Evaluating our eBPF prototype, we show that *SpinTrap* can accurately track the sending rates and assess the responsiveness of QUIC traffic, singling out flows that do not react to congestion. As such, *SpinTrap* provides a novel building block for Internet congestion management that can help in improving the Internet's resilience.

I. Introduction

Avoiding an Internet congestion collapse relies on the cooperation of all participating entities. Applications typically deploy congestion control (CC) [1] provided by congestion-aware transport protocols or implement their own CC [2] and are expected to react to congestion once per round trip [3]. Today's congestion management further includes active queue management (AQM), i.e., services deployed in the network that give end-hosts earlier and more specific feedback on the current congestion status. Latest proposals, such as L4S [4], take this approach to the extreme and aim at providing very low queuing delays with very low congestion loss, but require a well-aligned interaction between CC and AQM.

Despite this progress, the Internet's resilience to high traffic loads still relies on the assumption that most traffic is CC-driven while unresponsive flows remain a potential cause for future congestion collapses [5]. In this context, AQM could provide incentives for deploying CC [6] and, indeed, many proposals include mechanisms to limit the advantages of CC free riders [7]–[9]. However, these approaches only probabilistically or broadly identify misbehaving traffic, e.g., by random packet drops to target large bandwidth flows, which also hurts benign actors. Hence, while AQM can prevent unresponsive flows from gaining unlimited advantages, it struggles with such traffic and benefits are often wiped out (cf. L4S [10]). Thus, congestion management needs solutions for identifying unresponsive flows, e.g., via their sending behavior.

Traditionally, flows using CC are driven by the so-called congestion window (`cwnd`) which caps the in-flight data. Hence, most related works assess CC behavior via estimating the `cwnd` [11]–[14]. However, the `cwnd` is not directly exposed to passive observers and needs to be inferred from flow behavior. For TCP, this process involves assigning the transmitted data to individual round trips, which have to be inferred as well [15]. Additional challenges are the increasing use of pacing and novel CC algorithms, such as BBR [16], that are no longer *directly* driven by the `cwnd`. Solutions assessing the flow behavior of TCP are thus prone to errors.

Instead of a detailed classification, other approaches assume TCP to be responsive and UDP to be unresponsive [17], [18] which is problematic in light of a changing Internet landscape [19]. For example, QUIC [20] represents a new class of responsive UDP applications and there is no general way to distinguish QUIC from other UDP traffic [21], which makes broad classifications by related work infeasible. Yet again, QUIC, implemented in user space, also poses new challenges as its CC can be easily modified [22], [23], such that it cannot be simply classified as responsive, either. Fortunately, QUIC also enables a new solution space as we show in this paper: its optional spin bit mechanism, while intended for round-trip time measurements, lends itself for accurately tracking the sending behavior of QUIC flows, which makes it possible to assess their congestion responsiveness.

Hence, we propose *SpinTrap*, a speed trap for Internet flows. *SpinTrap* leverages the spin bit for tracking the sending volumes of QUIC traffic and observes any visible congestion signals in the form of packet drops or Explicit Congestion Notification (ECN) markings. Based on this information, *SpinTrap* assesses the congestion responsiveness of flows by checking whether they react to the observed signals: those subject to signals should show a corresponding decrease in the sent data volume. Evaluating a prototype of *SpinTrap* with three QUIC stacks, we find that it can accurately track the sending behavior and assess the responsiveness of the studied flows. *SpinTrap* even uncovers that some stacks do not always adequately react to the congestion signals. Overall, this paper contributes the following:

- We propose *SpinTrap* which uses the QUIC spin bit to identify unresponsive QUIC flows.
- We prototype *SpinTrap* in eBPF and evaluate it using three popular QUIC stacks.
- Our results indicate that *SpinTrap* can serve as a novel building block for Internet congestion management and that it enables a broad spectrum of future work.

II. A Primer on Congestion Management

The decentralized allocation of bandwidth is a key challenge in the Internet as most flows strive for maximizing their throughput. Addressing the concomitant risk of a congestion collapse, end-hosts are encouraged to deploy congestion control (CC) [1], i.e., adjust their sending rates to the current network congestion status. For this, senders typically maintain a congestion window (`cwnd`), using network feedback, e.g., via congestion signals, to estimate how much data can be sent without causing congestion and then limiting their unacknowledged in-flight data to this estimate. We label flows that neither deploy CC nor react to congestion signals as *unresponsive*.

Congestion signals. Packet loss is *the* traditional indicator for overflowing buffers and an overloaded network, but it comes with performance penalties [16]. In contrast, Explicit Congestion Notification (ECN) [3] allows signaling impending congestion without packet loss using two bits in the IP header: end-points can use any of two available ECT codepoints to enable ECN while network hops can signal congestion by changing ECT to CE. CE markings are typically emitted by active queue management (AQM) mechanisms that give early feedback to end-hosts and actively assist in the allocation of bandwidth. *Classic CC* interprets CE signals equally to packet loss and should react to both once per window of data [3] while *scalable CC* can react more than once [24].

Advanced congestion management. Originally intended to avoid a congestion collapse, congestion countermeasures increasingly aim to additionally improve end-to-end performance. As a latest innovation, L4S [4], e.g., envisions providing ultra-low queuing delays across the Internet by sending fine-grained, more frequent ECN markings for a DCTCP-style, scalable CC [25] that relies on end-host and network support. Hence, the discussed concepts demand different forms of end-host cooperation and congestion management.

Congestion management adoption. Avoiding a congestion collapse "only" requires end-hosts to deploy some form of responsive CC. RFC 7567 [5], thus, recognizes unresponsive traffic as a threat while Floyd and Fall [6] identify AQM mechanisms as potential incentive givers for using CC, e.g., by restricting the bandwidth of unresponsive flows. Further reducing congestion-related effects, e.g., with L4S, stretches the assumed support to ECN awareness and scalable CC. Extending Floyd and Fall's thought, AQM could again provide incentives for the needed technological deployments, e.g., via L4S benefits for benign flows, but first requires suitable measures for identifying and handling unresponsive traffic.

Related work on handling unresponsive traffic. Many popular AQM mechanisms, such as RED [26] and CoDel [27], and even algorithms designed with unresponsive traffic in mind, such as CHOKe [7], struggle when facing unresponsive flows [8]. CHOKe, e.g., assumes large queue and traffic shares of misbehaving flows and compares each incoming packet to a random packet in the queue, dropping both if they belong to the same flow. However, this mechanism has trouble singling out unresponsive traffic and also significantly affects benign traffic [9]. Similar problems arise for related works that also equate high queue shares or flow arrival rates with unresponsiveness [28]–[32]. Some approaches follow an orthogonal path, broadly assuming all UDP flows to be unresponsive and isolating them [17], [18].

The need for assessing responsiveness. Probabilistic solutions always carry the risk of harming benign traffic. Global considerations, such as demonizing UDP, are even more problematic, especially in light of a changing Internet landscape [19] and a growing share of responsive UDP traffic, e.g., using QUIC [33], [34] or representing videoconferencing software [2], [35]. Yet again, the user-space nature of QUIC introduces a lot of variability in CC implementations [22], [23], bringing into question whether all QUIC traffic should even be broadly classified as responsive as is generally done for TCP. Consequently, there is a growing need for assessing the responsiveness of traffic more closely.

Related work on assessing congestion responsiveness. There are many works studying the congestion response of remote hosts that aim to identify their CC algorithms (CCAs); we focus on passive approaches as the only suitable options for in-line co-deployment with AQM. While some works use end-to-end machine learning (ML) [36], [37], most rely on estimating and analyzing the `cwnd`. Jaiswal et al. [11], e.g., replicate the sender-side `cwnd` state with a finite state machine but need to explicitly model CCAs. This approach is impractical, especially considering the variety of QUIC implementations [22], [23]. Other approaches instead use the flow round-trip time (RTT) to approximate the `cwnd` as the amount of inflight bytes per RTT, e.g., via the "flight method" [12] or the TCP timestamp option [13]. However, the accuracy of intermediate RTT estimates is challenged by, e.g., the increasing use of pacing, i.e., spreading out transmissions over entire RTTs. Finally, Hagos et al. [14] estimate the inflight bytes of a connection to then approximate the `cwnd` via ML. Notably, no solution actually assesses the responsiveness of flows as they only focus on estimating the `cwnd`. Consequently, there is no suitable method for identifying unresponsive traffic for use in conjunction with AQM.

Takeaway. *Today's congestion management has evolved from its original intention of preventing a congestion collapse and increasingly strives for performance benefits. These benefits rely on an increasing coupling between network-based AQM and end-host CC, and an increasing deployment of corresponding mechanisms, such as ECN or scalable CC. Ideally, existing techniques are equipped with mechanisms to encourage CC (or even ECN/L4S) while still being capable of handling unresponsive traffic. As existing works are not up for this task, we require new approaches that can reliably assess the congestion responsiveness of flows.*

III. System Design

Fully leveraging today's congestion management approaches requires identifying (and isolating) unresponsive traffic as it can wipe out all potential benefits (cf. Sec. II). Addressing this need, we propose *SpinTrap*, an Internet speed

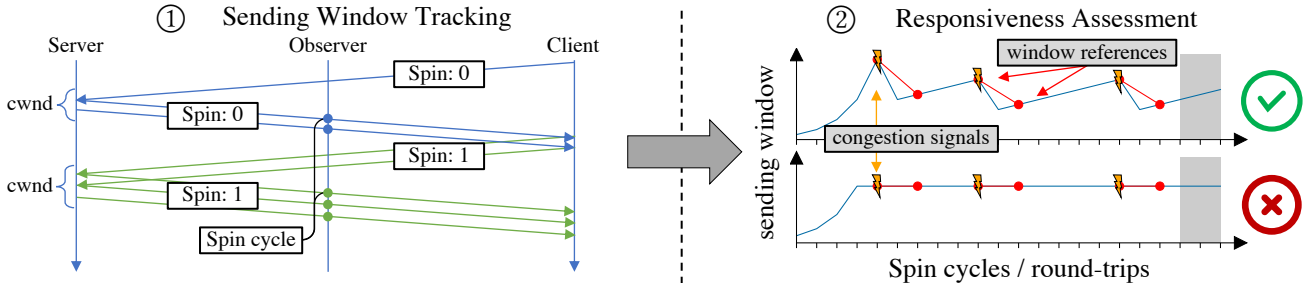

Fig. 1. *SpinTrap* uses the spin bit semantics for ① tracking the instantaneous sending window. For ② assessing responsiveness, it stores a sliding window of window estimates (blue line) and congestion signals (packet loss, CE markings; orange bolts). It then compares window estimates with congestion signals (red dots with bolts) with ones that could have reacted on them (red dots without bolts). Flows without window reduction are classified as unresponsiveness.

trap that ① tracks the sending behavior of QUIC flows and ② assesses their congestion responsiveness as visualized in Fig. 1. In particular, we monitor the *instantaneous sending window*, i.e., the amount of bytes transmitted in each round trip, and then check if flows react to congestion signals by reducing their sending volumes. Subsequent applications can then use the assessment to, e.g., provide better performance to responsive flows, thus incentivizing CC and ECN deployment. In this paper, we solely focus on *identifying* unresponsive traffic to provide a solid foundation for future work. For this, we examine the QUIC spin bit as one solution for tracking the sending behavior (①) while we leverage packet loss and ECN to assess responsiveness (②).

A. Tracking the Instantaneous Sending Window – ①

The first main component of *SpinTrap* tracks the amount of data transmitted by QUIC flows in each round trip using the explicit measurement signals of QUIC's spin bit.

Spin bit. The spin bit is a binary, explicit measurement signal designed to enable RTT measurements. As illustrated in Fig. 1 (left), the server of a connection always reflects the spin bit, i.e., it sends out the last value it has received, while the client spins the bit, sending out the inverse. Two consecutive flips of the resulting square wave constitute a *spin cycle* and indicate a round trip while the time between these flips equals the RTT.

Tracking sending windows via the spin bit. The `cwnd` defines the maximum amount of unacknowledged payload bytes a connection may have in flight. Considering that the process from sending a packet to receiving an ACK takes at least one round trip, the amount of bytes transmitted in this round trip can, in the best case, correspond to the `cwnd`, further coinciding with the spin bit semantics as visualized in Fig. 1 (left). However, there are several cases where this correlation does not apply, such that we focus on the aforementioned *instantaneous sending window*.

Why *SpinTrap* does *not* track the `cwnd`. In practice, senders, besides being application-limited, can transmit less than the `cwnd`, e.g., if packets of the previous cycle are still unacknowledged and block sending capacity. Additionally, some CCAs, such as BBR [16], do not directly follow the `cwnd`. As a consequence, estimating the `cwnd` based on the transmitted bytes is prone to error. Instead, using the *instantaneous sending window* allows us to assess the perceivable impact of the tracked flows. *SpinTrap*, thus, stores the spin bit state of each flow to distinguish spin cycles, counts the transmitted QUIC bytes for each cycle to estimate the instantaneous sending window, and logs the count at each spin bit flip. Overall, the instantaneous sending window represents the amount of data a sender sends and is still ultimately governed by CC.

Sources of inaccuracy. RFC 9002 [38] specifies that QUIC protocol headers and header protection mechanisms are considered in the `cwnd` and, thus, in the payload sending window. However, some QUIC packets, e.g., only containing ACK frames, are not subject to CC, but indistinguishable on the wire due to QUIC's encryption. Hence, counting *all* QUIC bytes theoretically causes an overestimation of the instantaneous sending window while misclassified spin cycles, e.g., due to reordering, could cause an underestimation. Additionally, RFC 9000 [20] mandates end-hosts to disable the spin bit for one in 16 connections to address privacy concerns [39], disabling the mechanism altogether for some connections.

B. Congestion Responsiveness Assessment – ②

In a second step, *SpinTrap* assesses the congestion responsiveness of flows using its sending window estimates and information on observed congestion signals. Traditionally, these signals equate to packet loss while ECN offers an additional, explicit signal. In any case, flows subject to these signals are expected to decrease their sending windows to free up queues and relieve congestion. Hence, *SpinTrap* observes congestion signals and checks for the described behavior; flows ignoring congestion signals are classified as unresponsive. The challenge for *SpinTrap* lies in correctly identifying whether or not a flow has seen a congestion signal.

Tracking packet loss. For packet loss, e.g., caused by overfull buffers or deliberately by AQM, the signal can only be determined unequivocally on the device causing the packet loss and on the end-hosts involved in the connection while it is inaccessible at other hops, especially with QUIC's encryption. Co-located with access to queuing and/or AQM information, *SpinTrap* can capture these packet loss signals directly.

Tracking ECN. In contrast to packet loss, ECN is an explicit signal that is set in the IP header. Hence, *SpinTrap* can track ECN markings even if these were emitted at other hops earlier on the flow path. As such, this variant can also be used in a

standalone deployment, e.g., as additional on-path device or via mirror ports. In this paper, we study both variants.

Assessing the CC reaction. Classic CC is expected to react to congestion signals once per round trip [3]. As the signal needs time to propagate back to the sender, the first reaction will, at the earliest, arrive at the observer one full RTT later (cf. Fig. 1 (left)). Translated to spin cycles, we have to track the sending window for two to three consecutive spin cycles depending on whether or not the spin bit flip and congestion signal coincide. Intuitively, as conceptualized in Fig. 1 (right), the sending window should be smaller after observing a congestion signal as reaction to congestion. Hence, for classic CC, tracking if *any* congestion signal has been emitted suffices.

Scalable CC. L4S envisions the use of DCTCP-style, scalable CC which works on fine-granular queuing statistics via frequent ECN signals, such that its reaction may also depend on the number of signals [4]. Thus, assessing its CC responsiveness potentially requires a more advanced logic, e.g., including the number of CE markings. However, we use a simple classification logic which defines those flows as responsive that have a smaller sending window after observing at least one CE marking or packet drop. Yet, accommodating corresponding extensions, *SpinTrap* also tracks the number of dropped packets and the number of CE markings.

C. Further Design Considerations

For this paper, we focus on the design of *SpinTrap* based on three considerations that we discuss in the following.

QUIC vs. TCP. Conceptually, *SpinTrap* could track TCP and QUIC alike. However, as already touched upon in Sec. II and discussed in more detail by Allman et al. [40], leveraging implicit TCP semantics for measuring flow characteristics is imprecise. Allman et al. consequently argue for adding explicit measurement semantics into end-to-end protocols. QUIC follows this call with the spin bit, an optional feature enabling RTT measurements [20]. *SpinTrap*, thus, focuses on QUIC and the spin bit to leverage the explicit information, ideally yielding a more robust assessment. Yet, *SpinTrap* is not limited to QUIC and can be extended for TCP, e.g., using concepts discussed in Sec. II, which we leave for future work.

QUIC identification. QUIC is generally indistinguishable from other UDP traffic [21]. For simplicity, *SpinTrap* identifies QUIC by checking for the fixed bit of QUIC version 1. While this approach relies on a single bit and might even become infeasible in light of QUIC bit greasing [41], we leave providing a robust QUIC identification for future work.

Flow identification. *SpinTrap* needs to keep per-flow state and distinguish flows to assess their sending behavior. We rely on the four-tuple of source and destination IP addresses and UDP ports. However, with QUIC, multiple connections could use the same four-tuple and be captured as a single flow. As an alternative, QUIC enables differentiating flows via their connection ID, yet connection migration with drastically changed network conditions without changing IDs could interfere with any subsequent sending behavior assessment. As the latter scenario seems more likely than the former, we equip *SpinTrap* with the four-tuple approach, which can be easily adapted to the five-tuple to also accommodate TCP.

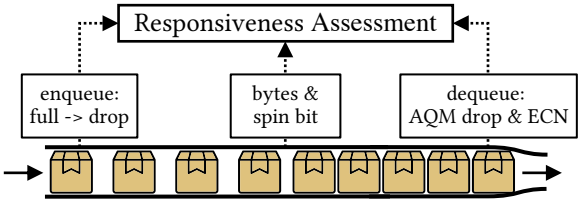

Fig. 2. *SpinTrap* tracks the sending windows of flows using a `tc filter` (middle) and observes congestion signals using Linux tracepoints (left, right). The responsiveness assessment is attached as a `tc filter` and uses the collected information to classify the responsiveness of flows.

IV. *SpinTrap* eBPF Prototype

We prototype *SpinTrap* using a combination of eBPF and Linux kernel tracepoints as visualized in Fig. 2. Via tracepoints, we collect information on packet drops caused by overflowing buffers at packet enqueue (left) and packet drops or CE markings caused by AQM at dequeue (right). We track the byte counts and spin bit state for packets that have entered the queue (middle) via a `tc filter`. *SpinTrap*'s responsiveness assessment uses the combined information and is also attached as a `tc filter`. Basing on the considerations in Sec. III-C, we next present relevant details of our implementation.

Short and long headers. Only short header packets carry the spin bit. However, at connection startup, short header packets can be *coalesced* together with long header packets in a UDP datagram. In this case, there can be at most one short header packet appearing as the last entry [20]. Hence, to correctly track *all* sent bytes, we also check for coalesced packets by disassembling the entire UDP datagram using length information in QUIC's long headers and extracting any trailing short header packet.

Per-flow state. Using the flow four-tuple as key, a hashmap stores all relevant per-flow statistics. For estimating the instantaneous sending window, we track the last spin bit state, and the observed QUIC bytes since the last flip. For assessing responsiveness, we further track the sending window estimates of the three previous spin cycles, the congestion signals (packet drops and CE markings) observed in those cycles, and a classification; new flows are initialized as *unclassified*.

Sending window estimation. For packets with a known four-tuple in the hashmap, we first check for a spin bit flip. In case of a flip, *SpinTrap* extracts (i) the current byte count, and (ii) the current congestion signal counts, updating the hashmap accordingly before proceeding with the congestion responsiveness assessment. Otherwise, we only increment the byte count by the current packet size, i.e., the length of the QUIC header and payload, leaving the classification untouched as the current round trip is not yet complete.

Congestion responsiveness assessment. In a final step, *SpinTrap* assesses the responsiveness based on the sending window estimates and the congestion signal counts. Upon detecting a spin bit flip, *SpinTrap* first checks the number of congestion signals in the oldest cycle. If this count is larger than 0, flows

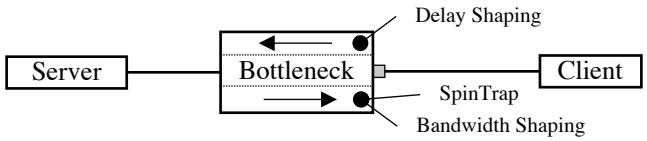

Fig. 3. Our testbed consists of three machines. The *Bottleneck* machine emulates different network scenarios and deploys *SpinTrap* while the *Server* machine transmits data to the *Client* machine using three QUIC stacks.

are expected to have reduced their window and we check for this reaction by comparing the corresponding sending windows. If the window has indeed decreased, the flow is classified as *responsive*; other traffic is classified as *unresponsive*. *SpinTrap* pessimistically assumes a noticeable reaction two RTTs after the initial congestion signal (cf. Sec. III-B).

V. Evaluation

We extensively study *SpinTrap* in a controlled testbed, evaluating the window tracking and congestion responsiveness assessment for different QUIC stacks and CCAs in different network scenarios. In the following, we first present our evaluation methodology before discussing our main findings.

A. Methodology

We conduct our evaluation in the testbed illustrated in Fig. 3. It consists of three machines running Ubuntu 20.04 that are interconnected with Gigabit Ethernet. QUIC servers on the *Server* machine (left) transmit data to QUIC clients on the *Client* machine (right), both using a Linux 5.15 kernel. The *Bottleneck* machine runs an out-of-tree 5.10. kernel [42]; it shapes network characteristics and deploys our prototype.

Network configuration. We add delay on the ingress of the interface from *Client* to *Bottleneck* using Intermediate Functional Block (IFB) interfaces and `tc netem`. We emulate bandwidths on the egress of the same interface using a Hierarchical Token Bucket (HTB) to which we can attach child `tc qdiscs` for modelling different queues and AQM mechanisms. In all settings shown in this paper, we configure a bottleneck queue size equal to the bandwidth delay product (BDP) and use a bottleneck bandwidth of 50 Mbps.

Studied queues. We study the behavior of *SpinTrap* subject to a standard drop-tail queue and CoDel [27]. The latter is one prominent AQM mechanism that can emit the CE markings needed for *SpinTrap*'s standalone mode.

Studied QUIC stacks. We test *SpinTrap* with three QUIC stacks. Again accounting for *SpinTrap*'s standalone mode, we focus on stacks that do support ECN while most do not implement it [43]. In particular, we use Microsoft's *MsQuic* [44], Amazon's *s2n-quic* [45], and *picoquic* [46], a well-maintained independent QUIC implementation. We add the spin bit to MsQuic and s2n-quic as they do not support it by default.

Traffic generation. Throughout all scenarios and connections, the server transmits one 100 MB file. To investigate the optimal achievable performance, we enable the spin bit on *all* connections, ignoring the recommendations of RFC 9000 [20].

Studied CCAs. We select a common subset of CCAs from the three QUIC stacks. In particular, all stacks support *Cubic*, a classic loss-based CCA that relies on the `cwnd` and serves as the TCP default on Linux. We further select BBR as a modern model-based CCA which does not use the `cwnd` to track congestion, but instead estimates the available capacity via measurements of the BDP. The stacks notably differ in their BBR implementations: MsQuic uses a pure BBRv1 which does not react to packet loss or ECN while s2n-quic uses BBRv2, a newer BBR version that also considers these signals. Picoquic uses an adapted version of BBRv1 described to also react to packet loss and ECN. Hence, the BBR implementations might not all provide the expected response to congestion signals and represent a challenge for *SpinTrap*.

Measurements. We capture two main forms of information: for all QUIC end-hosts (server and client), we collect logs provided by the different QUIC stacks. Picoquic uses the `qlog` format [47] while the other stacks use custom formats. We further capture the detailed output of *SpinTrap*, including the timestamp of each spin bit change, the observed sending window, and the responsiveness classification.

Experiments. We perform twenty independent measurement runs for each configured scenario.

B. Tracking the Sending Window.

In a first set of experiments, we assess the accuracy of *SpinTrap*'s sending window tracker, comparing its estimates with the groundtruth provided by the QUIC stacks. Here, we specifically mind the semantics of different frames in QUIC w.r.t. CC. In particular, packets only carrying ACK frames are not governed by CC and should not be counted, yet our *Client* will mainly only acknowledge the payload of the *Server*. *SpinTrap* might, thus, overestimate the client's sending window. To characterize this difference, we subtract the bytes of packets only carrying ACK frames from the overall number of bytes sent by the QUIC hosts to correctly represent the sending window without ungoverned packets and compare it with *SpinTrap*'s prediction. Further, to rule out a temporal bias at the end of the experiment, we filter out the last spin cycle.

Experiment Setup. We start a single flow using each stack and each CCA, four different RTTs from 5 ms to 100 ms, and three different queues. Fig. 4 shows the mean absolute difference in bytes normalized per spin cycle between *SpinTrap*'s estimate and the bytes that should be counted at a drop-tail queue (a), CoDel with packet drop ((b), top), and CoDel with ECN ((b), bottom). Each point represents one measurement run; darker circles correspond to tracking the server, lighter crosses to the client which mainly sends ACKs.

1) Drop-Tail Queue: We first study the accuracy of *SpinTrap*'s sending window tracking on a regular drop-tail queue.

Results. Starting with the servers (darker points in Fig. 4 (a)), we observe that *SpinTrap* achieves (near) perfect accuracy for picoquic and s2n-quic. For MsQuic, there are only slight overestimations which are caused by MsQuic sending pure ACK frames that *SpinTrap* mistakenly includes in its estimation. As the difference is negligibly small (less than 50 bytes), we conclude that *SpinTrap* can accurately track the sending behavior of QUIC stacks that primarily send data.

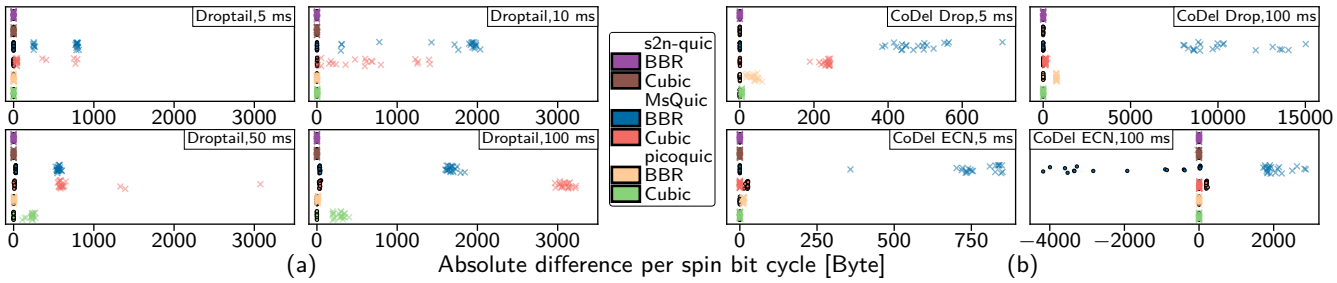

Fig. 4. Mean absolute difference between the sending window estimated by *SpinTrap* and the bytes that should be counted for single flows at a drop-tail queue (left), a CoDel queue with packet drop (right, top), and a CoDel queue with ECN (right, bottom) with different RTTs.

To study the behavior of QUIC end-points that mainly acknowledge data, we also inspect the inverse sending direction (lighter crosses in Fig. 4 (a)). In this setting, we can observe a similar behavior for picoquic and s2n-quic as both mostly transmit ACK frames with PADDING, meaning that (almost) all QUIC packets count toward the inflight. In contrast, MsQuic again sends many ACK frames without PADDING: these do not count toward the sending window, causing a large overestimation by *SpinTrap*. To protect such pure ACKing flows from a subsequent misclassification, we could equip *SpinTrap* with thresholds on a minimum sending volume that needs to be reached before flows are classified.

2) CoDel: For standalone deployments, *SpinTrap*'s responsiveness assessment requires CE markings, e.g., emitted by AQM. Hence, in a second scenario, we study the behavior of *SpinTrap*'s window tracking subject to CoDel with packet drop and with ECN. We only configure CoDel's queue size, use default or recommended values otherwise, and keep the remaining configuration from Sec. V-B1 without changes.

Results. Fig. 4 (b) illustrates the mean absolute differences for CoDel with packet drops (top) and ECN (bottom) for RTTs of 5 ms (left) and 100 ms (right). The results for CoDel with packet drops indicate that deploying AQM in general does not change the fundamental observations from before: *SpinTrap* can still track the sending window of all stacks and CCAs accurately. However, for CoDel with ECN, there are two noteworthy observations. First, *SpinTrap* again slightly overestimates the sending window for MsQuic Cubic which we attribute to ACK frames without PADDING as before. Second, there are significant underestimations for MsQuic BBR in about half of our experiments which we pinpoint to burst loss on the ingress interface of our bottleneck machine. In particular, each experiment shows exactly *one* spin cycle with heavy packet loss *before* the packets reach *SpinTrap*. Hence, the inaccuracy is caused by packets not reaching *SpinTrap* and not by flaws of our prototype. Yet, this observation shows that the path from the sender to *SpinTrap* has a non-negligible impact on the measured sending windows.

Takeaway. SpinTrap *can accurately track the sending window of all stacks and CCAs in most cases. The main challenges are overestimations caused by ACK frames sent without PADDING and underestimations caused by packet loss on the path from the sender to* SpinTrap*. However, for picoquic and s2n-quic, we achieve very high accuracy independent of the used CCA and traffic pattern. Hence, overall, the sending window estimation of* SpinTrap *fulfills its intended task.*

C. Congestion Responsiveness Assessment

Given that *SpinTrap* can accurately track the sending behavior of QUIC flows, we now evaluate how well this information can be leveraged for assessing the flows' responsiveness. In this paper, we use a simple classification logic which checks for a decreased instantaneous sending window two round trips after a congestion signal has been noticed. In case there is no congestion signal for a new classification, we use the previous one. We further experiment with additional logic in the form of *majority voting*: we collect the individual classifications, but reclassify the flow based on the maximum number of classifications overall. Thus, a flow that has already been classified several times as unresponsive will also be classified as unresponsive even if one individual classification attests responsiveness. For simplicity, we emulate an unresponsive sender by fixing the `cwnd` of a picoquic flow to $4 \times BDP$ (*No_CC*). Note that this flow still shows *some* reaction if packet loss caps its in-flight, preventing *No_CC* from transmitting at full rate (cf. Sec. III-A). In the following, we first study the assessment quality for single flows before evaluating a scenario with multiple interacting flows.

1) Single Flow: For the single flow scenario, we mostly reuse the setup of Sec. V-B: we start a single flow using each stack and each CCA as well as *No_CC* and setup the testbed with three different queues and four RTTs. We then compute the *responsiveness score* as the fraction of spin cycles in which a flow has been classified as *responsive*. Hence, *SpinTrap* would ideally give a low score to *No_CC* while all other flows should get a high score. Fig. 5 shows the responsiveness score for each flow and iteration at a drop-tail queue (a), CoDel with packet drop ((b), top), and CoDel with ECN ((b), bottom); each data point represents the value for one iteration. The values for the raw classifications are plotted in the unshaded areas while the shaded regions represent the majority voting.

Drop-Tail. As can be seen in Fig. 5 (a), *SpinTrap* achieves very accurate assessments for many stack and algorithm combinations, such as s2n-quic, MsQuic Cubic, and picoquic BBR. Where the plain classification struggles, the additional majority voting can often significantly improve the results,

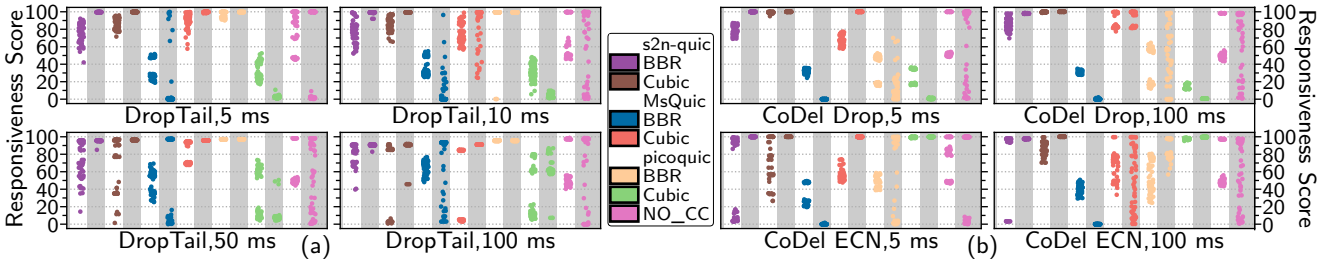

Fig. 5. Responsiveness scores for single flows at a drop-tail queue (left), a CoDel queue with packet drop (right, top), and a CoDel queue with ECN (right, bottom) with different RTTs. Shaded columns indicate the majority voting results.

which is especially noticeable for MsQuic and s2n-quic Cubic: both see a large range of responsiveness scores in the plain classification across most settings, but have near perfect responsiveness scores when applying majority voting.

SpinTrap expectedly struggles when classifying MsQuic BBR, an implementation of BBRv1, which does not react to packet loss and ECN. Surprisingly, *SpinTrap* also fails to correctly classify picoquic Cubic which it mostly classifies as unresponsive. Studying our results in more detail, we find that picoquic Cubic tends to only react to larger numbers of packet loss, e.g., as part of burst loss, while it seldomly reacts on few lost packets. As this behavior arguably constitutes unresponsiveness, we consider the decision of *SpinTrap* correct.

Finally evaluating the results for *No_CC*, we observe that the plain classification is undecided in a lot of cases, indicated by raw responsiveness scores of around 50 % in most scenarios, meaning that *SpinTrap* mistakenly deems *No_CC* responsive in half of the cases. In contrast to these rather steady results, the majority voting fails to provide a clear classification, instead yielding a large range of responsiveness scores which corresponds to *SpinTrap* constantly flipping its decision. The main reason for these results is that our emulated unresponsive sender *No_CC* shows *some* responsiveness as it reduces its sending rate when packets are already lost on the wire but not yet declared lost by the sender which blocks part of the sending window. Overall, however, *SpinTrap* generally fulfills its task despite only using a very simple logic.

CoDel. Next, we study the assessment quality of *SpinTrap* as reported in Fig. 5 (b) when using CoDel with packet drops (top) or ECN markings (bottom) for RTTs of 5 ms (left) and 100 ms (right). Starting with CoDel with packet drops, we observe that most settings are similar to the drop-tail queue. Notable exceptions are picoquic's and MsQuic's BBR, which see lower responsiveness scores. The reason is that CoDel drops packets earlier than a drop-tail queue, but gives a less aggressive signal. Hence, the BBRv1 implementation of MsQuic and the adapted BBRv1 version of picoquic do not seem to react as strongly. The same holds for picoquic Cubic due to the reasons discussed in the previous paragraph. Hence, *SpinTrap* again arguably identifies unresponsiveness correctly.

Inspecting the ECN variant, we see similar behavior in most cases. The largest difference is notable for picoquic's Cubic implementation which does not react to packet loss, but does react to ECN and now sees near perfect scores.

Takeaway. *Overall,* SpinTrap *can provide a correct assessment for most CCAs in most cases for single flows. Yet, it expectedly struggles with BBR and our emulated unresponsive sender as both do not show a clear behavior. Additionally,* SpinTrap *uncovers unexpected unresponsiveness of picoquic's Cubic implementation which we verified with additional analyses. Hence, even in cases where* SpinTrap *mistakenly classifies flows as unresponsive, the flows also do react in an unresponsive way, showing that* SpinTrap *works as intended.*

2) Multiple Flows: Lastly, we evaluate *SpinTrap* in a more challenging scenario with multiple flows. For this, we let four flows with CC compete against four flows with No_CC. We judge the responsiveness assessment quality using F1 scores: unresponsive flows classified as such are true positives, while responsive flows that are correctly classified as responsive represent true negatives. Fig. 6 shows the F1 scores for each stack and CCA with CoDel with packet drop (top) and CoDel with ECN (bottom). Due to heavy congestion, we further slightly modify *SpinTrap* to require a reduction to 90 % or less of the original window[1] and only assess responsiveness if the original window is larger than four full-size packets to not classify flows at their extreme lows. Fig. 6 (left) shows the original results, Fig. 6 (right) the modified results; each data point again represents the value for one iteration and the shaded regions again represent the majority voting.

Unmodified *SpinTrap*. Inspecting Fig. 6 (a), we only observe few good results, mainly for CoDel with packet loss and a small RTT. The main reason for this bad performance is that the bandwidth is severely limited and each NO_CC flow constantly pressures the queue, hence causing the responsive flows to back off. However, these already operate at their lower limits and hence cannot back off further causing many false positives. Similarly, the heavy congestion causes significant packet loss for the NO_CC flows which hence also implicitly react to the congestion, thus causing false negatives. Accounting for these observations, we next study the behavior of *SpinTrap* with the modifications described above.

Modified *SpinTrap*. Looking at Fig. 6 (b), we can see that the modifications significantly improve the performance of *SpinTrap*. In particular, all stacks and CCAs that we have previously identified as being indeed responsive now see F1 scores of up to 100. However, some algorithms, such as

[1]For reference, TCP Cubic reduces its cwnd to 70 % upon congestion [48].

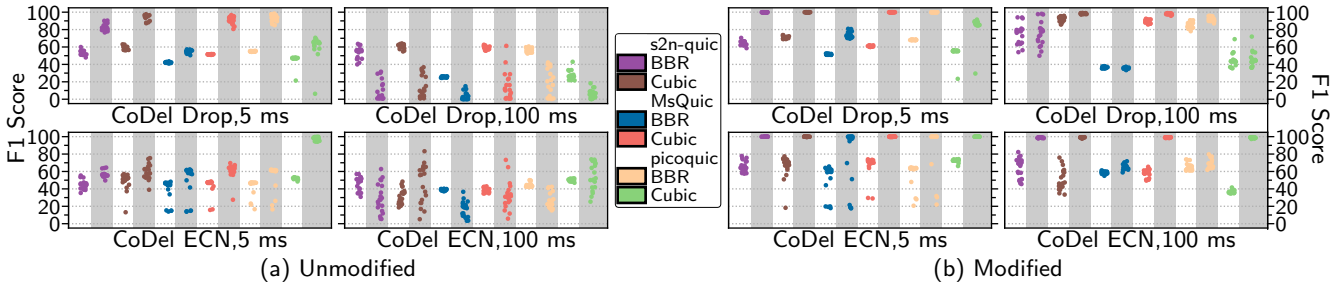

Fig. 6. F1 scores of the responsiveness assessment with (right) and without modifications (left) for multiple flows and CoDel with packet drop (top) and CoDel with ECN (bottom) with different RTTs. Shaded columns indicate the majority voting results.

picoquic Cubic and MsQuic BBR, still challenge *SpinTrap*'s classification as they do not show a truly responsive behavior.

Takeaway. SpinTrap *is challenged by our emulated unresponsive sender and falsely classifies flows as unresponsive if they cannot back off further. However, equipping* SpinTrap *with simple additional filters significantly improves the classification performance, both for packet loss and ECN. We conclude that* SpinTrap *can provide a solid basis for classifying the responsiveness of real Internet flows and that there is additional potential in focusing its feature set and studying its performance across even more diverse settings.*

VI. Discussion

Our evaluation shows the general usefulness of *SpinTrap*. In the following, we discuss a selection of further considerations, especially regarding its operational use.

Deployment location. *SpinTrap* relies on packet loss information or CE markings. For using packet loss, *SpinTrap* needs to be co-located with access to queuing or AQM information. As such, it could be integrated into existing monitoring systems, e.g., by leveraging the growing programmability with SDN or P4 [49]. With ECN, *SpinTrap* can also be deployed standalone as an additional on-path device or via mirror ports. Yet, observed CE markings can be lost on the way to the receiver after passing *SpinTrap* as misbehaving devices are known to interfere [43]. Affected flows would not have a fair chance to react and might be classified as unresponsive. Hence, for more reliable assessments, *SpinTrap* should be deployed close to the destinations, e.g., at ISP access links.

Which flows to monitor? Tracking *all* flows with per-flow state at Internet-scale is infeasible [50]. However, related work has already shown that random sampling can be sufficient in many applications [50]–[52] and Internet traffic is generally imbalanced as relatively few flows carry large shares of the overall volume [53]. Further considering that these flows pose a bigger threat if they are unresponsive, using *SpinTrap* in combination with a heavy-hitter detection, such as PRECISION [54], could be a sensible deployment scenario.

Using the spin bit. *SpinTrap* relies on the spin bit, an optional mechanism of QUIC [20] not contained in other protocols. However, more than 50 % of QUIC hosts reachable via IP already support the mechanism [39]. As our approach underlines the usefulness of explicit measurement information, we argue that adding the spin bit on a permanent basis to QUIC, and maybe as an option to TCP, can help in providing more measurable information to the network. One open challenge is checking the spin bit's correctness as malicious actors could fake spin cycles, e.g., to hide their unresponsiveness.

Using ECN. While ECN sees broad support for TCP, QUIC stacks are significantly lacking behind [43] and the use of ECN with QUIC is currently severely limited as the needed mirroring of ECN signals is only usable on a small fraction of connections [43]. However, this will likely change in the future as ECN mirroring is a mandatory feature of QUIC.

Responsiveness assessment logic. *SpinTrap* achieves good results with a simple assessment logic. We further show that extending our concept with a majority voting and additional filtering rules can improve the results. Hence, we believe that *SpinTrap* provides a solid foundation for future experimentation to fine-tune the responsiveness assessment. For example, vanilla *SpinTrap* considers flows as responsive if they show *any* response, i.e., a reduction of 1 byte suffices while larger reductions might be desirable for larger flows as demonstrated by our modifications. Finally, incorporating L4S and scalable CC might require a new assessment logic. We leave studying these considerations in more detail to future work.

VII. Conclusion

Unresponsive flows still strain the resilience of the Internet and many approaches aiming to limit their advantages cause unnecessary harm to benign flows. Additionally, unresponsive traffic can wipe out all performance benefits provided by modern congestion management approaches, such as L4S, thus negatively affecting others with their behavior. Hence, there is a need for identifying unresponsive Internet flows.

In this paper, we propose *SpinTrap*, a novel building block for Internet congestion management. *SpinTrap* leverages the spin bit to track the sending windows of QUIC flows and combines this information with observations on packet loss and ECN markings to assess the flows' responsiveness. We prototype *SpinTrap* in eBPF and evaluate it with popular QUIC stacks, finding that simple logic suffices for assessing responsiveness. The resulting flow classification can be used in multiple ways and lays the foundation for future efforts protecting the Internet from unresponsive traffic.

Acknowledgements

This work has been funded by the German Research Foundation DFG under Grant No. WE 2935/20-1 (LEGATO). We thank the anonymous reviewers for their valuable feedback.

References

[1] V. Jacobson, "Congestion Avoidance and Control," *ACM SIGCOMM Computer Communication Review*, vol. 18, no. 4, pp. 314–329, 1988. [Online]. Available: https://doi.org/10.1145/52325.52356

[2] K. MacMillan, T. Mangla, J. Saxon, and N. Feamster, "Measuring the Performance and Network Utilization of Popular Video Conferencing Applications," in *Proceedings of the 2021 ACM Internet Measurement Conference (IMC)*, 2021. [Online]. Available: https://doi.org/10.1145/3487552.3487842

[3] K. K. Ramakrishnan, S. Floyd, and D. L. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," IETF, RFC 3168, 2001. [Online]. Available: https://doi.org/10.17487/RFC3168

[4] B. Briscoe, K. De Schepper, M. Bagnulo, and G. White, "Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture," IETF, RFC 9330, 2023. [Online]. Available: https://doi.org/10.17487/RFC9330

[5] F. Baker and G. Fairhurst, "IETF Recommendations Regarding Active Queue Management," IETF, RFC 7567, 2015. [Online]. Available: https://doi.org/10.17487/RFC7567

[6] S. Floyd and K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet," *IEEE/ACM Transactions on Networking*, vol. 7, no. 4, pp. 458–472, 1999. [Online]. Available: https://doi.org/10.1109/90.793002

[7] R. Pan, B. Prabhakar, and K. Psounis, "CHOKe - a stateless active queue management scheme for approximating fair bandwidth allocation," in *Proceedings of the 2000 IEEE Conference on Computer Communications (INFOCOM)*, 2000. [Online]. Available: https://doi.org/10.1109/INFCOM.2000.832269

[8] G. Abbas, Z. Halim, and Z. H. Abbas, "Fairness-Driven Queue Management: A Survey and Taxonomy," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 324–367, 2016. [Online]. Available: https://doi.org/10.1109/COMST.2015.2463121

[9] G. Abbas, U. Raza, Z. Halim, and K. Kifayat, "ARCH: A dual-mode fairness-driven AQM for promoting cooperative behaviour in best effort Internet," *IET Networks*, vol. 8, no. 6, pp. 372–380, 2019. [Online]. Available: https://doi.org/10.1049/iet-net.2018.5089

[10] K. De Schepper, O. Albisser, O. Tilmans, and B. Briscoe, "Dual Queue Coupled AQM: Deployable Very Low Queuing Delay for All," *arXiv.2209.01078*, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2209.01078

[11] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, "Inferring TCP Connection Characteristics Through Passive Measurements," in *Proceedings of the 2004 IEEE International Conference on Computer Communications (INFOCOM)*, 2004. [Online]. Available: https://doi.org/10.1109/INFCOM.2004.1354571

[12] J. Oshio, S. Ata, and I. Oka, "Identification of Different TCP Versions Based on Cluster Analysis," in *Proceedings of the 2009 International Conference on Computer Communications and Networks (ICCCN)*, 2009. [Online]. Available: https://doi.org/10.1109/ICCCN.2009.5235248

[13] G. Casagrande, F. Granelli, and D. Miorandi, "TCPMoon: Monitoring the Diffusion of TCP Congestion Control Variants in the Internet," in *Proceedings of the 2011 IEEE International Conference on Communications (ICC)*, 2011. [Online]. Available: https://doi.org/10.1109/icc.2011.5963408

[14] D. H. Hagos, P. E. Engelstad, A. Yazidi, and Ø. Kure, "General TCP State Inference Model From Passive Measurements Using Machine Learning Techniques," *IEEE Access*, vol. 6, pp. 28 372–28 387, 2018. [Online]. Available: https://doi.org/10.1109/ACCESS.2018.2833107

[15] S. Shakkottai, R. Srikant, N. Brownlee, A. Broido, and K. Claffy, "The RTT distribution of TCP flows on the Internet and its impact on TCP based flow control," CAIDA, Technical Report, 2004. [Online]. Available: https://catalog.caida.org/paper/2004_tr_2004_02

[16] 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," *ACM Queue*, vol. 14, no. 5, pp. 20–53, 2016. [Online]. Available: https://doi.org/10.1145/3012426.3022184

[17] S. Yilmaz and I. Matta, "On Class-based Isolation of UDP, Short-livedand Long-lived TCP Flows," in *Proceedings of the 2001 International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2001. [Online]. Available: https://doi.org/10.1109/MASCOT.2001.948894

[18] T. Yamaguchi and Y. Takahashi, "A queue management algorithm for fair bandwidth allocation," *Computer Communications*, vol. 30, no. 9, pp. 2048–2059, 2007. [Online]. Available: https://doi.org/10.1016/j.comcom.2007.04.002

[19] A. Feldmann, O. Gasser, F. Lichtblau, E. Pujol, I. Poese, C. Dietzel, D. Wagner, M. Wichtlhuber, J. Tapiador, N. Vallina-Rodriguez, O. Hohlfeld, and G. Smaragdakis, "The Lockdown Effect: Implications of the COVID-19 Pandemic on Internet Traffic," in *Proceedings of the 2020 ACM Internet Measurement Conference (IMC)*, 2020. [Online]. Available: https://doi.org/10.1145/3419394.3423658

[20] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," IETF, RFC 9000, 2021. [Online]. Available: https://doi.org/10.17487/RFC9000

[21] M. Kühlewind and B. Trammell, "Manageability of the QUIC Transport Protocol," IETF, RFC 9312, 2022. [Online]. Available: https://doi.org/10.17487/RFC9312

[22] A. Mishra, S. Lim, and B. Leong, "Understanding Speciation in QUIC Congestion Control," in *Proceedings of the 2022 ACM Internet Measurement Conference (IMC)*, 2022. [Online]. Available: https://doi.org/10.1145/3517745.3561459

[23] A. Mishra and B. Leong, "Containing the Cambrian Explosion in QUIC Congestion Control," in *Proceedings of the 2023 ACM Internet Measurement Conference (IMC)*, 2023. [Online]. Available: https://doi.org/10.1145/3618257.3624811

[24] K. De Schepper and B. Briscoe, "The Explicit Congestion Notification (ECN) Protocol for Low Latency, Low Loss, and Scalable Throughput (L4S)," IETF, RFC 9331, 2023. [Online]. Available: https://doi.org/10.17487/RFC9331

[25] K. De Schepper, O. Tilmans, B. Briscoe, and V. Goel, "Prague Congestion Control," IETF, Internet-Draft, 2023, work in progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-briscoe-iccrg-prague-congestion-control/

[26] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993. [Online]. Available: https://doi.org/10.1109/90.251892

[27] K. Nichols and V. Jacobson, "Controlling Queue Delay," *ACM Queue*, vol. 10, no. 5, pp. 20–34, 2012. [Online]. Available: https://doi.org/10.1145/2208917.2209336

[28] G. Chatranon, M. A. Labrador, and S. Banerjee, "BLACK: Detection and Preferential Dropping of High Bandwidth Unresponsive Flows," in *Proceedings of the 2003 IEEE International Conference on Communications (ICC)*, 2003. [Online]. Available: https://doi.org/10.1109/ICC.2003.1204258

[29] I. Yeom, "A rate-based drop policy for punishing unresponsive flows," *Computer Communications*, vol. 29, no. 10, pp. 1868–1878, 2006. [Online]. Available: https://doi.org/10.1016/j.comcom.2005.05.012

[30] J. Zheng, L. Zhao, and T. Zhang, "Improving Unresponsive Flow Control by Active Queue Management Algorithm," in *Proceedings of the 2007 IEEE Wireless Communications and Networking Conference (WCNC)*, 2007. [Online]. Available: https://doi.org/10.1109/WCNC.2007.795

[31] S. Yi, X. Deng, G. Kesidis, and C. R. Das, "A dynamic quarantine scheme for controlling unresponsive TCP sessions," *Springer Telecommunication Systems*, vol. 37, no. 4, pp. 169–189, 2008. [Online]. Available: https://doi.org/10.1007/s11235-008-9104-2

[32] G. Aldabbagh, M. Rio, and I. Darwazeh, "Fair Early Drop: An Active Queue Management Scheme for the Control of Unresponsive Flows," in *Proceedings of the 2010 IEEE International Conference on Computer and Information Technology (CIT)*, 2010. [Online]. Available: https://doi.org/10.1109/CIT.2010.449

[33] J. Rüth, I. Poese, C. Dietzel, and O. Hohlfeld, "A First Look at QUIC in the Wild," in *Proceedings of the 2018 International Conference on Passive and Active Network Measurement (PAM)*, 2018. [Online]. Available: https://doi.org/10.1007/978-3-319-76481-8_19

[34] J. Zirngibl, P. Buschmann, P. Sattler, B. Jaeger, J. Aulbach, and G. Carle, "It's Over 9000: Analyzing Early QUIC Deployments with the Standardization on the Horizon," in *Proceedings of the 2021 ACM Internet Measurement Conference (IMC)*, 2021. [Online]. Available: https://doi.org/10.1145/3487552.3487826

[35] C. Sander, I. Kunze, K. Wehrle, and J. Rüth, "Video Conferencing and Flow-Rate Fairness: A First Look at Zoom and the Impact of Flow-Queuing AQM," in *Proceedings of the 2021 International Conference on Passive and Active Network Measurement (PAM)*, 2021. [Online]. Available: https://doi.org/10.1007/978-3-030-72582-2_1

[36] C. Sander, J. Rüth, O. Hohlfeld, and K. Wehrle, "DeePCCI: Deep Learning-based Passive Congestion Control Identification," in *Proceedings of the 2019 ACM SIGCOMM Workshop on Network Meets AI & ML (SIGCOMM NetAI)*, 2019. [Online]. Available: https://doi.org/10.1145/3341216.3342211

[37] X. Chen, S. Xu, X. Chen, S. Cao, S. Zhang, and Y. Sun, "Passive TCP Identification for Wired and Wireless Networks: A Long-Short Term Memory Approach," in *Proceedings of the 2019 IEEE International Wireless Communications & Mobile Computing Conference (IWCMC)*, 2019. [Online]. Available: https://doi.org/10.1109/IWCMC.2019.8766577

[38] J. Iyengar and I. Swett, "QUIC Loss Detection and Congestion Control," IETF, RFC 9002, 2021. [Online]. Available: https://doi.org/10.17487/RFC9002

[39] I. Kunze, C. Sander, and K. Wehrle, "Does It Spin? On the Adoption and Use of QUIC's Spin Bit," in *Proceedings of the 2023 ACM Internet Measurement Conference (IMC)*, 2023. [Online]. Available: https://doi.org/10.1145/3618257.3624844

[40] M. Allman, R. Beverly, and B. Trammell, "Principles for Measurability in Protocol Design," *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 2, pp. 2–12, 2017. [Online]. Available: https://doi.org/10.1145/3089262.3089264

[41] M. Thomson, "Greasing the QUIC Bit," IETF, RFC 9287, 2022. [Online]. Available: https://doi.org/10.17487/RFC9287

[42] "Linux kernel tree with L4S patches," 2023. [Online]. Available: https://github.com/L4STeam/linux

[43] C. Sander, I. Kunze, L. Blöcher, M. Kosek, and K. Wehrle, "ECN with QUIC: Challenges in the Wild," in *Proceedings of the 2023 ACM Internet Measurement Conference (IMC)*, 2023. [Online]. Available: https://doi.org/10.1145/3618257.3624821

[44] "MsQuic," 2023. [Online]. Available: https://github.com/microsoft/msquic

[45] "S2n-quic," 2023. [Online]. Available: https://github.com/aws/s2n-quic

[46] "Picoquic," 2023. [Online]. Available: https://github.com/private-octopus/picoquic

[47] R. Marx, L. Niccolini, M. Seemann, and L. Pardue, "Main logging schema for qlog," IETF, Internet-Draft, 2023, work in progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-quic-qlog-main-schema

[48] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks," IETF, RFC 8312, 2018. [Online]. Available: https://doi.org/10.17487/RFC8312

[49] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-Independent Packet Processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014. [Online]. Available: https://doi.org/10.1145/2656877.2656890

[50] T. Holterbach, E. C. Molero, M. Apostolaki, A. Dainotti, S. Vissicchio, and L. Vanbever, "Blink: Fast Connectivity Recovery Entirely in the Data Plane," in *Proceedings of the 2019 USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2019. [Online]. Available: https://www.usenix.org/system/files/nsdi19-holterbach.pdf

[51] M. Apostolaki, A. Singla, and L. Vanbever, "Performance-Driven Internet Path Selection," in *Proceedings of the 2021 ACM SIGCOMM Symposium on SDN Research (SOSR)*, 2021. [Online]. Available: https://doi.org/10.1145/3482898.3483366

[52] S. Sengupta, H. Kim, and J. Rexford, "Continuous In-Network Round-Trip Time Monitoring," in *Proceedings of the 2022 ACM SIGCOMM Conference*, 2022. [Online]. Available: https://doi.org/10.1145/3544216.3544222

[53] S. Bauer, B. Jaeger, F. Helfert, P. Barias, and G. Carle, "On the Evolution of Internet Flow Characteristics," in *Proceedings of the 2021 ACM/IRTF Applied Networking Research Workshop (ANRW)*, 2021. [Online]. Available: https://doi.org/10.1145/3472305.3472321

[54] R. Ben Basat, X. Chen, G. Einziger, and O. Rottenstreich, "Designing Heavy-Hitter Detection Algorithms for Programmable Switches," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1172–1185, 2020. [Online]. Available: https://doi.org/10.1109/TNET.2020.2982739