Undermining Delay-based QUIC Congestion Control: A Receiver-driven Attack via Crafted Host Delays

Shaorui Ren[1], Jia Zhang[2], Enhuan Dong[1], Mingwei Xu[1], Yixuan Zhang[1], Jiahao Cao[1], Jianping Wu[1]
[1]Tsinghua University, [2]Zhongguancun Laboratory
rsr23@mails.tsinghua.edu.cn, zhangj@mail.zgclab.edu.cn, {dongenhuan, xumw}@tsinghua.edu.cn, zhangyix20@mails.tsinghua.edu.cn, caojh2021@tsinghua.edu.cn, jianping@cernet.edu.cn

2025 IEEE 33rd International Conference on Network Protocols (ICNP) | 979-8-3315-0376-5/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICNP65844.2025.11192451

***Abstract*—QUIC gains significant attention due to its superior transmission performance, achieving widespread adoption in both academia and industry. To improve round-trip time (RTT) estimation, QUIC introduces the Host Delay field, enabling senders to exclude receiver-induced delays. Many delay-based congestion control algorithms (CCAs) rely on these RTT estimates to detect congestion and regulate sending rates. However, we find that malicious Host Delay values can distort RTT measurements, causing inappropriate rate adjustments by CCAs.**

In this paper, we investigate a new class of attacks leveraging maliciously crafted Host Delay values. To our knowledge, we are the first to analyze the vulnerability of Host Delay and present QUDIT, a universal receiver-driven attack targeting delay-based QUIC CCAs. Unlike prior attacks that presume full network queuing delays visibility and undetected injection capabilities, our attacker model only grants the adversary access as a standard QUIC receiver with limited knowledge of bottleneck conditions. We minimally modify the QUIC receiver to infer bottleneck queuing behavior in real time. Based on these inferences, we design dynamic Host Delay crafting strategies tailored to the specific behavior of various delay-based CCAs and accounting for random network fluctuations. Our attack prompts the sender to overshoot its rate, leading to excessive bandwidth consumption at the bottleneck and degradation of competing flows. Results demonstrate the throughput degradation of victim flow achieves up to 60% within 0.3 s and amplification gains between 200× and 600×. We propose defenses mitigating QUDIT, with vulnerabilities reported to IETF and QUIC maintainers.

***Index Terms*—QUIC, Host Delay, CCA Attack, Bandwidth Occupying**

I. Introduction

QUIC is a user-space transport protocol built on UDP, initially proposed by Google to address the limitations of TCP [1]. Its design facilitates easier customization and introduces innovations such as the Host Delay mechanism for more accurate round-trip time (RTT) measurement [2], [3]. Furthermore, by leveraging Transport Layer Security (TLS) 1.3, QUIC encrypts the entire connection, rendering packet inspection by intermediate devices infeasible [2]. Having been adopted by major technology companies including Google, Facebook, Apple, and Microsoft [4]–[10], QUIC has since gained widespread attention and achieved significant success. QUIC's user-space design makes it easier to update and customize CCAs compared to kernel-space TCP, facilitating the rapid development of a diverse set of algorithms [11].

As QUIC continues to gain widespread adoption, much research has focused on developing congestion control algorithms (CCAs) to optimize bandwidth usage while avoiding network congestion. These CCAs typically rely on metrics such as RTT, packet loss rate, and data delivery rate to assess congestion and adjust the sending rate.

Delay-based CCAs represent a significant class of CCAs [12]–[17], and are commonly used across various applications [18], [19]. When RTT increases, these CCAs infer network congestion and reduce sending rates. Conversely, stable or decreasing RTT leads to increased sending rates for better bandwidth utilization. With RTT sensitivity growing in applications like RTC [20], delay-based CCAs offer advantages by proactively detecting congestion and ensuring low end-to-end delays, leading to their widespread adoption.

Accurate RTT estimation is crucial for delay-based CCAs. In QUIC, the sender's RTT is calculated based on the ACK frames sent by the receiver [2]. To improve accuracy, QUIC introduces a *Host Delay* field in the ACK frame. This field accounts for the delays incurred at the receiver from receiving a packet to sending the corresponding ACK frame [2]. Host Delay includes factors like buffering delays due to unavailable decryption keys and waiting for additional packets to be consolidated into a single ACK frame. These receiver-side operations improve QUIC's efficiency but also introduce additional delay. To correct for this, the QUIC sender subtracts the Host Delay from its locally calculated RTT, yielding a more accurate measurement of network path conditions.

However, we find significant defects in the design of Host Delay and present a receiver-driven attack via crafting Host Delays. Host Delay is estimated and written into the ACK frame by the receiver before being transmitted to the sender. Considering QUIC operates in userspace, an attacker controlling the receiver can easily manipulate the Host Delay. In the scenario under consideration, the attacker operates as a QUIC client, requesting data from a legitimate QUIC server. When the server calculates the RTT using the Host Delay mechanism and employs a delay-based CCA, the attacker can manipulate the sender's behavior (which, in our attack, also

refers to the server's behavior). **The core idea of the attack is to dynamically set the value of Host Delay to align with the network queuing delay**. This manipulation deceives the delay-based CCA into perceiving an absence of network congestion. Consequently, the attacker can trigger excessive data transmission from the delay-based CCA on the legitimate QUIC server, intentionally causing network congestion and suppressing other victim traffic at the network bottleneck.

Although the aforementioned attack appears procedurally straightforward in principle, two significant challenges remain in its practical execution. First, the attacker needs the QUIC server to send frequent ACKs to capture RTT changes and accurately estimate bottleneck queuing delay. However, clients are typically restricted from sending large amounts of application data, limiting the number of ACKs. Second, the attack's effectiveness depends on how delay-based CCAs use RTT and on network delay fluctuations. CCAs may use absolute delay [13], [15] or delay gradients [16] for congestion assessment. Furthermore, dynamic networks cause RTT fluctuations even with stable queuing delay, introducing uncertainty into the attacker's queuing delay inference (§III-B).

In this paper, we present QUDIT[1], a universal receiver-driven attack against delay-based QUIC CCAs. To address the first challenge, we make minimal modifications to the QUIC receiver, enabling the attacker to selectively send ack-eliciting frames, as defined in [2], without accompanying application data. This allows the attacker to independently trigger ACK frames from the QUIC server, obtaining RTT measurements (§IV-A) and estimating bottleneck queuing delay more timely. To address the second challenge, we analyze various delay-based CCAs and develop Host Delay crafting strategies that adapt to both absolute delay-based and delay gradient-based CCA designs. Moreover, our crafting strategies fully consider the impact of random network fluctuations (§IV-B).

By overcoming the above challenges, an attacker carrying on QUDIT establishes a connection with a QUIC server employing a delay-based CCA, and operates as a receiver. Under normal circumstances, connections[2] would "fairly"[3] share the bottleneck link bandwidth due to the operation of CCAs. However, QUDIT allows the attacker to manipulate the QUIC sender's CCA by providing cafted Host Delay values in ACK frames, causing the scapegoat sender to oversend data and encroach on the victim's bandwidth share.

To the best of our knowledge, QUDIT is the first to exploit QUIC's new design to target CCAs, demonstrating distinct capabilities across four key dimensions. First, regarding attack effectiveness, the attacker uses only a small number of ACK frames to induce a large volume of data, achieving significant amplification. Moreover, because the exploited senders are trusted entities, traditional defenses that block source IPs are ineffective. Second, by modifying Host Delay, QUDIT manipulates RTT calculations without altering the actual reception time of ACK frames, making the scheme more concise. Third, based on QUIC's user-space operation, QUDIT is highly feasible. Finally, since QUIC calculates RTT and passes the results to the CCA through a unified interface, any CCA using delay as a congestion signal is vulnerable to this attack.

We implement QUDIT in msquic [7] and evaluate it in the network emulator Mininet [21] §V-B and physical testbed §V-C. Our results demonstrate that QUDIT is effective across different CCAs. RTT threshold-based Vegas experienced a throughput reduction of 40%-60%, Copa 50% maximum. And RTT gradient-based Vivace experienced a 20% throughput reduction. Notably, by exploiting Scapegoat QUIC Servers running delay-based CCAs, QUDIT achieves significant throughput degradation even against non-delay-based victims. The most widely deployed CCA CUBIC [22] suffers the most severe impact, with 50%-60% reduction. And for BBR, QUDIT can reduce its throughput by about 20%. Moreover, the attack achieves an amplification gain factor ranging from 200 to over 600, increasing with bandwidth. The impact occurs in less than 0.3 seconds, with the attacker sending only a few tens of kilobytes of data. Furthermore, the attack effect will not decrease as the RTT becomes longer.

In summary, we make the following contributions.

- To the best of our knowledge, we present QUDIT, the first attack leveraging the QUIC Host Delay mechanism, to manipulate the delay-based CCAs' decision-making.
- We develop a receiver-side method for low-cost end-to-end delay measurement and subsequent real-time bottleneck queuing delay estimation, and enable a Host Delay crafting method optimizing the best attack performance near inferred queuing delays (§IV).
- Experiments demonstrate that QUDIT can degrade the victim flow's throughput by up to 60% within 0.3 seconds across different types of CCAs, and achieves an amplification gain factor ranging from 200 to over 600 (§V).
- We propose defenses against QUDIT targeting three tiers: protocol revisions, implementation constraints, and network mechanisms (§VI-B).
- We engaged IETF and QUIC maintainers through responsible disclosure, reported this threat, proposed protocol revisions and coordinated with the maintainers (§VI-C).

II. Background and Related Work

In this section, we first introduce the background of delay-based CCAs (§II-A) and the design of Host Delay in QUIC (§II-B). Then, we conduct a review of related works on CCA attacks and their limitations (§II-C).

A. Delay-Based CCAs

Congestion control algorithms (CCAs) deployed at the transport layer regulate packet sending rates through different congestion signals. These algorithms typically implement rate control via two mechanisms: direct rate adjustment or con-

[1]We will release the QUDIT code and Mininet environment soon at: https://anonymous.4open.science/r/HostDelayCrafting25-643B/README.md

[2]In this paper, the terms "connection" and "flow" are used interchangeably.

[3]Fairness among competing CCAs is beyond the scope of this paper. We define "fair share" as the average bandwidth distribution between two undisturbed, bottleneck-sharing flows in a steady state, based on the assumption that CCA designs inherently consider fairness.

gestion window (CWND) sizing, where CWND defines the maximum unacknowledged packets allowed, and most CCAs use CWND to control sending rate. In both QUIC and TCP, the adoption is driven by ACK signals (implemented as ACK frames or in TCP packets, respectively).

Delay-based CCAs use RTT as a congestion indicator, estimating queuing delay by subtracting the minimum RTT (as propagation delay) from the current RTT. Algorithms like Vegas [13] and Copa [15] detect congestion early by comparing queuing delay against thresholds, enabling preemptive CWND reduction before packet loss occurs. However, their early response makes them less competitive against loss-based CCAs, often leading to underutilized bandwidth. To improve aggressiveness, these CCAs commonly adopt the minimum queuing delay within a sampling window (by selecting a minimum RTT in several samples for calculation) [13]–[15] to avoid overestimation. Notably, Copa is designed to be more aggressive than Vegas when competing with loss-based flows.

In contrast, delay-gradient CCAs, such as PCC Vivace [16], infer congestion based on changes in RTT rather than absolute values. PCC Vivace employs an online learning approach that evaluates the utility of different sending rates by considering the delay gradient, sending rate, and packet loss, to iteratively find and use the transmission rate that maximizes utility.

B. Host Delay in QUIC

Host Delay refers to the latency introduced by the receiver, between packet reception and the corresponding ACK transmission, including intentional delays such as buffering delays due to unavailable decryption keys, and waiting for additional packets to be consolidated into a single ACK frame (this reduces the frequency of packets that carry only ACK frames, and thus reduces packet transmission and processing cost [2], [23]). Host Delay is encoded in the ACK frame and, once decoded, is used in the calculation of RTT at the sender [3].

The introduction of Host Delay allows the RTT values to more accurately reflect changes in network path conditions, thereby mitigating the impact of receiver-induced latency. The relevant RTT metrics and calculations are summarized in Table I. An endpoint measures time between packet sending and acknowledgment as an RTT sample (*latest_rtt*), using these samples and peer-reported Host Delays to build RTT statistics for the network path. Key metrics include: (1) *min_rtt* represents the sender's estimation of the minimum RTT over a period of time; (2) *smoothed_rtt* is an exponentially weighted moving average (EWMA) of the endpoint's RTTs; (3) *rttvar* estimates the variation in RTT samples. QUIC specifies that *min_rtt* should be calculated from raw *latest_rtt* samples (unadjusted by Host Delay) to prevent underestimation.

The utilization of Host Delay is subject to two constraints. Firstly, after decoded from ACK frame, Host Delay is constrained by taking the minimum of the receiver's specified *max_ack_delay* parameter and itself. *max_ack_delay* represents the maximum amount of time by which the receiver will delay sending ACK frames (default 25 milliseconds). In order to expand the effective range of Host Delay values without being limited by this, the attacker could declare a significantly high *max_ack_delay* as it is determined by the receiver (§III-C). Secondly, when the sender adjusts *latest_rtt* using Host Delay, *min_rtt* constraint is also taken into account. The RTT sample is adjusted by subtracting Host Delay only if the result is not less than *min_rtt*. Otherwise, *adjusted_rtt* is set equal to *latest_rtt* directly. In other words, *adjusted_rtt* is bounded by *min_rtt*. After adjusting for Host Delay, the *adjusted_rtt* is utilized for the computation of *smoothed_rtt* and *rttvar*. The calculation formulas for these two parameters are fundamentally consistent with those in TCP.

After the sender processes the ACK frames, it conveys information about the network path state to the CCA through a unified interface, including the mentioned RTT values such as *adjusted_rtt* and *smoothed_rtt*. CCAs use the RTT values with the Host Delay removed, so malicious manipulation of Host Delay will affect all delay-based CCAs.

Discussions surrounding the Host Delay design in QUIC primarily focus on how its accurate RTT estimation optimizes transmission performance, particularly for delay-based CCAs. The emphasis has been on effectively integrating this new design into protocol implementation. However, the security risks associated with the potential manipulation of Host Delay by a malicious receiver have been overlooked. Host Delay is subject to the two constraints mentioned earlier, including *max_ack_delay*, which limits the receiver's ability to manipulate it within a meaningful range. Notably, this constraint's default value (25 ms) is significantly lower than typical connection RTTs. However, our research demonstrates that even when the maximum Host Delay is an order of magnitude smaller than the RTT (e.g., *max_ack_delay* of 25 ms and a minimum RTT of 250 ms), the receiver can still significantly disrupt the sender's CCA operation by adjusting the Host Delay within this limited range.

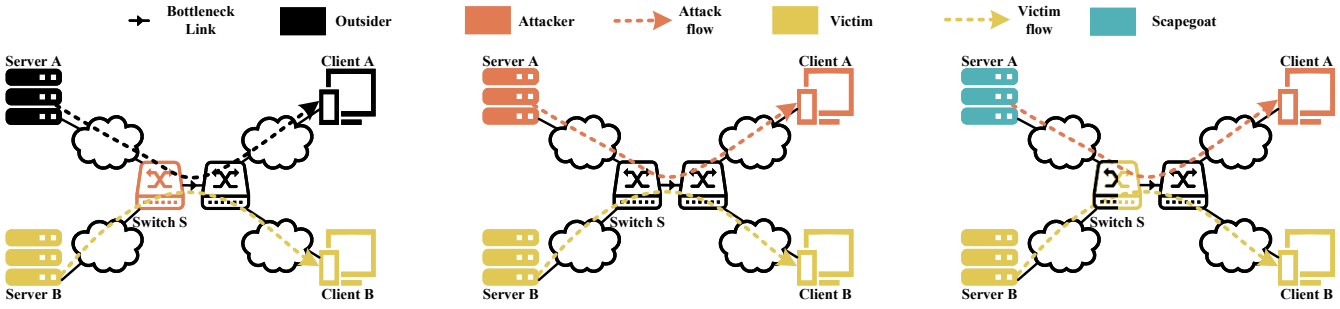
C. Limitations of Existing CCA Attacks

Existing attacks targeting TCP CCAs are primarily focused under three threat models: *on-path attack*, *off-path attack*, and *receiver-based attack*. Among these, there has been considerable discussion on *on-path* and *off-path* attacks [24]–[26] (§II-C1,§II-C2). However, both encounter limitations in QUIC, with the *on-path attack* as completely impractical. *Receiver-based attack* remains feasible, but the only known scheme, Optimistic ACK attack [27], is limited by QUIC's new design (§II-C3). QUIC senders can detect it via simple methods and defend themselves by immediately closing these malicious connections. Consequently, based on this subsection's analysis, we explore a novel attack scheme targeting QUIC CCAs following the third threat model.

To illustrate these attacks, we construct scenarios depicted in Fig. 1, where Client A and Client B establish connections with their respective servers: Server A sends data to Client A, while Server B sends data to Client B. Both connections share the same bottleneck, residing at Switch S. Note that the bottleneck link may also be located at a router. The queue or buffer impacts transport similarly on both switches and routers.

TABLE I: RTT-relevant variables maintained at QUIC endpoints

Name	Calculation	Description
latest_rtt	ACK Time - Send Time	The raw RTT samples.
min_rtt	min(latest_rtt)	Estimation of the minimum RTT
Host Delay	min(max_ack_delay, Host Delay Decoded from ACK)	Host Delay used in RTT estimation.
rtt_sub_HD	latest_rtt − Host Delay	A temporary variable
adjusted_rtt	rtt_sub_HD (if rtt_sub_HD ≥ min_rtt) latest_rtt (if rtt_sub_HD < min_rtt)	RTT adjusted for rtt_sub_HD ≥ min_rtt RTT adjusted for rtt_sub_HD < min_rtt
smoothed_rtt	$\frac{7}{8} \times$ smoothed_rtt + $\frac{1}{8} \times$ adjusted_rtt	EWMA of RTTs
rttvar	$\frac{3}{4} \times$ rttvar + $\frac{1}{4} \times$ |smoothed_rtt − adjusted_rtt|	Variation in RTT samples



(a) On-Path Attack **(b) Off-Path Attack** **(c) Receiver-Based Attack**

Fig. 1: The attack scenarios / threat models of existing CCA attacks.

For simplicity, Fig. 1 illustrates only the switch case. CCAs are deployed on Server A and Server B. Next, we analyze the existing CCA attacks under different threat models.

1) On-Path Attack: The attacker works on the switch, controls the packets of the target flow, particularly the ACK packets, affecting the performance of that flow. As shown in Fig. 1a, the attacker controls Switch S and can directly fabricate packets or modify the packets of the victim flow between Server B and Client B. In this way, the attacker causes the victim (Server B) to misinterpret the network as congested, thereby reducing the sending rate [24]–[26].

On-path attacks can be executed in TCP; however, due to QUIC's implementation of packet encryption, such attacks are no longer feasible. By integrating with TLS 1.3, QUIC encrypts both the payload of packets and parts of the packet headers, particularly the Packet Number field [28]. This encryption approach makes it nearly impossible for an attacker to modify or inject QUIC packets without detection.

2) Off-Path Attack: The attacker uses external flows to make the CCA of the target flow mistakenly believe that congestion has occurred, thereby reducing the sending rate of the victim flow. As shown in Fig. 1b, the attacker (Server A) controls the queue buildup in the bottleneck by sending data, causing Server B's CCA to believe that the network is congested and reduce its sending rate. This leads to a decrease in the transmission performance of the connection between Server B and Client B, making them victims of the attack.

Theoretically, such attacks can still be effective under QUIC. However, once malicious senders are detected [29], [30], they can be easily blocked and terminated. For instance, network devices such as switches can invalidate these attacks by deploying defenses based on ingress filtering [31]. By restricting the connection of attackers, normal users remain unaffected by the attacks and their resources are not compromised (§VI-B).

3) Receiver-Based Attack: This is the threat model investigated in our work. The attacker acts as a receiver and affects the sender's CCA behavior by forging ACK frames or packets, causing it to transmit more traffic. This leads to a decrease in the transmission performance of flows sharing the same bottleneck link, as shown in Fig. 1c. The attacker's target could be the other flows sharing the bottleneck or the link itself. Though Server A is not the attacker, it is exploited by the attacker to send a large number of packets, making it likely to be suspected as the attacker. Consequently, within this threat model, Server A acts as a scapegoat.

To the best of our knowledge, *Optimistic ACK Attack* [27] is the only proposed attack in this scenario. The attacker, acting as a data receiver, acknowledges data packets optimistically, even for those that have not actually been received. By this, the attacker circumvents the CCA from reducing CWND, which should occur following packet loss, causing it to send at rates that surpass the network's bottleneck bandwidth.

However, QUIC includes specific countermeasures against this attack: the sender can randomly skip some packet numbers to detect whether the ACKs are reasonable (§21.4 in [2]). When the sender detects that the receiver has acknowledged packet numbers that were never transmitted, it recognizes the occurrence of an attack. The sender can then terminate the connection immediately to mitigate potential damage.

Our work falls under *receiver-based attack* and uses three key concepts not considered by existing work. Compared to *on-path attack*, QUDIT manipulates the *Host Delay* field within legitimate ACK frames, without the need to forge spurious ACK frames. Unlike *off-path attack*, QUDIT utilizes

a legitimate sender to carry out the attack, making source IP blocking infeasible as it would disrupt legitimate traffic. In contrast to flooding-based attacks (e.g., Crossfire [32] and Coremelt [33]), beyond shared IP blocking resistance, QUDIT only needs to send a small number of ACK frames to achieve a significant attack, creating an amplification effect similar to that seen in amplification attacks [34] (calculated as the ratio between the volume of server-generated data and the data sent by the attacker), thereby reducing the attacker's costs.

Although this data amplification is inherent, conventional CCAs strictly limit transmission rates, ensuring flows remain within network capacity, rendering this "amplification" harmless. Our attack bypasses this critical safeguard, effectively disabling the CCA's rate-limiting mechanism. This unleashes the protocol's intrinsic amplification ratio and transforms a benign protocol feature into an exploitable vulnerability.

III. Overview of QUDIT

We first present our threat model and highlight its broad applicability (§III-A). Then, we discuss the challenges associated with this attack (§III-B). Finally, we outline the core idea and attack procedure, demonstrating how QUDIT effectively targets delay-based CCAs (§III-C).

A. Threat Model

Based on the analysis in existing CCA attacks, we adopt a *receiver-based attack* §II-C3 as our threat model. As illustrated in Fig. 1c, the attacker (Client A) establishes a connection with a QUIC server (Server A) that employs a delay-based CCA. Acting as the data receiver, the attacker manipulates the Host Delay feedback values, which interfere with the delay-based CCA's RTT measurements. This distorts Server A's perception of the current network congestion, causing it to transmit data at a rate that exceeds its fair share of bandwidth. As a result, Server A inadvertently becomes an accomplice (Scapegoat Server) in the attack, leading to performance degradation for victim flows sharing the same bottleneck, such as Flow B, and congestion at critical network links, such as Switch S.

Broadness of Victim CCAs: The attack can create bandwidth occupation effects no matter what type of CCA is deployed by Server B. Specifically, the victim Server B may use different CCAs than Server A, including non-delay-based CCAs such as CUBIC [35] or BBR [36]. Even when competing with loss-based CCAs, QUDIT can manipulate RTT measurement so that when congestion occurs (packet loss has already happened), Server A incorrectly perceives a non-congested state and continues to send packets at a high rate, thereby increasing aggressiveness in competition with other flows. Our experiments confirm that QUDIT can effectively occupy bandwidth when competing with various types of CCA (§V-C). For Server A, the attacker can connect to any QUIC server using a delay-based CCA, such as Vegas, Copa, and PCC Vivace. Unlike the attacks in [24]–[26], QUDIT is not limited to a specific CCA deployed at Server A.

Broadness of Victim Targets: The attack can target specific network victim flows (such as Flow B) or congest key links carrying multiple traffic streams (such as Switch S). For the first type of attack on a specific victim flow, we need to establish a flow sharing a bottleneck with the victim. Since we do not restrict Server A to one specific CCA, it is more likely to establish an eligible connection by attempting to connect with different servers. The attacker can then use bots to detect bottleneck links and infer whether the attack flow shares the same bottleneck link with the victim flow, a method widely used in [25], [32]. Additionally, when the attacker and the victim are in close proximity, they may share the last hop of the access network, thus sharing a bottleneck, such as connecting to the internet through the same public WiFi.

B. Design Challenge

To achieve QUDIT, we need to address two challenges.

Firstly, existing link state estimation information is insufficient for the attacker to accurately estimate current network bottleneck queuing delay and sender state, making it difficult to set an effective Host Delay for the intended attack. Therefore, the attacker needs the QUIC server to frequently send ACK frames to timely capture changes in RTT and accurately estimate queuing delay. However, as a data receiver, small data frames are inadequate to trigger the sender to provide sufficient ACK feedback, thus obtaining RTT measurements.

Secondly, directly using the inferred bottleneck queuing delay as the Host Delay hinders sustained attack effectiveness due to two inherent limitations: (1) randomness of the network means that the measured queuing delay estimates are likely to be inaccurate, and (2) varying ways that different CCAs utilize RTT complicate the design of an effective and universal attack scheme applicable to various delay-based CCAs.

C. Overview

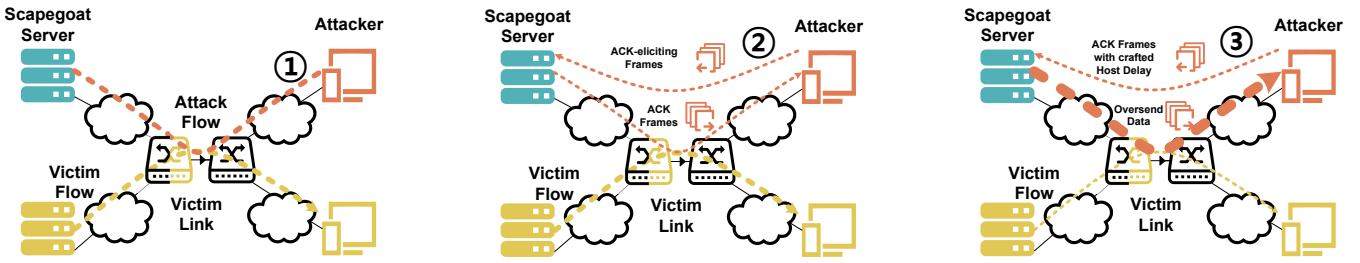
QUDIT includes two key design mechanisms to address the challenges above and achieve a long-term effective attack. The attack flow is shown in Fig. 2.

First, QUDIT establishes a flow that shares a bottleneck with the victim flow or passes through a victim link, using a QUIC server that employs a delay-based CCA. Considering that the utilization of Host Delay is limited by the *max_ack_delay* parameter [2], QUDIT will first declare a large *max_ack_delay* value to maximize the variable range of Host Delay[4].

Next, two key design mechanisms will be used to construct effective Host Delay values:

Delay Probing (*DP*) mechanism helps QUDIT obtain sufficient RTT measurements and enhance attack efficiency by varying the frequency of ACKs. *DP*'s dual-function mechanism operates through two key strategies: (1) When upper layer applications lack data to trigger QUIC server ACKs, the attacker sends minimal data packets via *DP*, inducing frequent ACK generation. This provides real-time RTT measurements for bottleneck queuing delay estimation. (2) Unlike normal receivers that delay ACKs (up to *max_ack_delay*) to aggregate

[4]*max_ack_delay* is set to 25 ms by default, with an upper limit of less than 2^{14} ms (approximately 16 seconds, which is ample to encompass the maximum queuing delay allowed in most transmission scenarios).



(a) Establish a flow sharing a bottleneck with the victim or via victim link with a QUIC server using delay-based CCA.

(b) Send ack-eliciting frames to elicit an ACK frame from QUIC, and obtain sufficient RTT measurements by this.

(c) Craft Host Delay to induce delay-based CCA overtransmission, congesting networks and throttling victim flows.

Fig. 2: The attack flow of QUDIT.

them, reducing feedback frequency, *DP* mechanism overrides ACK aggregation and timing of ACK, ensuring high-frequency and timely Host Delay feedback (§IV-A).

Host Delay Crafting (*HDC*) mechanism analyzes CCA behavior, and implements a refined Host Delay setting algorithm to overcome queuing delay estimation errors caused by network fluctuations. Due to estimation inaccuracies caused by network randomness and the diverse delay utilization strategies of different delay-based CCAs, directly using queuing delays measured by the *DP* mechanism is insufficient for sustained and effective attacks. *HDC* mechanism analyzes CCA behavior and selects Host Delay values near the estimated queuing delay across consecutive ACK frames. This approximates the effect of an ideal, error-free queuing delay estimation attack and enhances attack effectiveness over time (§IV-B).

IV. Attack Designs

In this section, we introduce the detailed design of the *Delay Probing* (§IV-A) and *Host Delay Crafting* (§IV-B).

A. Delay Probing

Delay Probing mechanism dictates when the attacker, acting as the data receiver, should send ACK frames and ack-eliciting frames, which can elicit the recipient to send ACK frames. By sending ack-eliciting frames at a certain frequency, QUDIT can obtain sufficient RTT measurements and estimate the queuing delay. By sending ACK frames, QUDIT can influence the server's CCA with the crafted Host Delay.

Ack-eliciting frames refer to the collective term for various types of frames in the QUIC protocol that can trigger ACK frames. For example, STREAM frames are ack-eliciting frames which carry application data. Considering the attacker, acting as a data receiver, may not have sufficient application data to transmit, resulting in a lower frequency of received ACK frames, QUDIT additionally sends other types of ack-eliciting frames to prompt the QUIC server to frequently send ACK frames, thereby obtaining RTT measurements.

QUDIT chooses to use MAX_DATA frames to trigger ACKs when there is not enough application data. In the QUIC protocol, MAX_DATA frames are used to inform the peer of the maximum amount of data that can be sent [2]. These frames are frequently used to communicate the new maximum amount of sendable data from the sender, so increasing the sending frequency does not raise any alarms. Complementing this stealth, the minimalist structure of the frame, containing only one field named Maximum Data, allows attackers to inject the minimal amount of data to trigger an ACK frame.

ACK frames contain the Host Delay field. Since an ACK frame can only contain one Host Delay value, to gain more chances to feedback Host Delays, and to ensure it is used promptly by the QUIC server, we need to select the time to send ACK frames in order to achieve an effective attack.

Contrary to aggregating feedback of several packets into one ACK frame, the attacker determines whether to send an ACK frame directly or wait for other packets based on the time interval since the last ACK frame was sent. Although the attacker sets a significantly large *max_ack_delay*, which allows it to wait a long period and acknowledge packets received within a single ACK frame, this results in an untimely response to Host Delay (due to additional latency introduced by waiting for more packets) and a limited number of Host Delays sent (due to aggregation of ACK frames). Therefore, we proactively decide when to send ACK frames to achieve effective attack.

Sending interval: Although sending ACK frames and ack-eliciting frames more frequently inherently benefit the attack scheme, it also incurs costs. First, the QUIC server needs to allocate more resources to handle these frames, which may impact normal data transmission and reduce its sending rate. Second, the attacker's bandwidth overhead increases, further diminishing the effectiveness of the attack, as excessively short time intervals could lower the amplification gain factor.

Fig. 3 illustrates the amplification gain factor and the median RTT measurement interval for the attacker at different interval thresholds. A "0ms" interval represents the scenario in which the attacker sends ACK frames and ack-eliciting frames as frequently as possible. "∞" represents normal receiver behavior without increased frequency. Using the same interval for both frame types enables them to be contained within a single QUIC packet. Otherwise, separate packets would require more resources and introduce complexity. We selected a 5 ms interval, which, under the experimental conditions detailed in §V-A, reduces the median RTT measurement interval for the

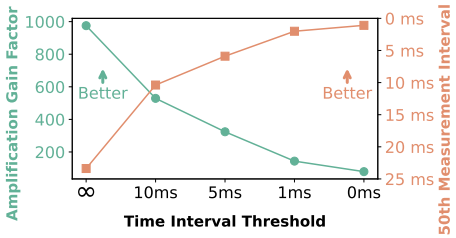


Fig. 3: Trade-off between RTT measurement frequency and amplification gain factor across attacker-selected sending intervals of ACK frames and ack-eliciting frames.

attacker by about 75% compared to a normal QUIC receiver, while achieving an amplification gain factor greater than 300.

B. Host Delay Crafting

The core idea of QUDIT is to dynamically set the value of Host Delay to align with the queuing delay. QUDIT estimates the current network queuing delay utilizing the RTT measurements obtained through *DP* mechanism, as Eq. (1). This estimation method is conceptually similar to the delay-based CCA's approach for estimating network queuing delays.

$$\text{queuing delay} = \text{adjusted\_rtt} - \text{min\_rtt} \tag{1}$$

However, simply setting the Host Delay to the estimated queuing delay is ineffective. This is due to three reasons. (1) Network fluctuations lead to inaccurate estimates of the queuing delay. (2) Different CCAs use delay in various ways and setting the estimated delay may yield poor results. (3) Constraints on Host Delay in the QUIC protocol can also affect its effectiveness (§II-B). To analyze these and present the corresponding designs, we first consider the inaccuracies in queuing delay estimation caused by network fluctuations, and distinguish three scenarios (assuming the sender's calculated *min_rtt* represents the RTT without queuing delay, which does not affect the validity of the analysis):

1) **Scenario 1 (Ideal Condition)**: If the estimated queuing delay matches the true value, the attacker sets the optimal Host Delay, and the sender calculates *rtt_sub_HD* equal to *min_rtt*. This maximizes the deception, as the *adjusted_rtt* contains no queuing delay, leading the delay-based CCA to believe there is no congestion, thus achieving the ideal attack result.
2) **Scenario 2 (Overestimated Queuing Delay)**: If the estimated queuing delay is too large, the attacker sets an overly high Host Delay, resulting in *rtt_sub_HD* being smaller than *min_rtt*. This has no deceptive effect, as the sender uses the unadjusted *latest_rtt* as the *adjusted_rtt* (As introduced in §II-B), allowing the CCA to detect the queuing delay and neutralizing the attack.
3) **Scenario 3 (Underestimated Queuing Delay)**: If the estimated queuing delay is too small, the attacker sets the Host Delay too low, making *rtt_sub_HD* larger than *min_rtt*. Although this influences the sender's *adjusted_rtt*, it does not completely eliminate the network queuing delay, so the CCA partially detects the queuing delay, leading to a suboptimal attack outcome.

Due to network fluctuations, the attacker can hardly consistently maintain the ideal condition (Scenario 1). The other scenarios reduce or completely eliminate the effectiveness of the attack. Next, we combine the above analysis to present dedicated attack methods for two different CCAs that use delay in distinct ways, and then unify them.

1) Attacking CCA Based on the Absolute Value of Delay: CCAs using absolute delay infer queuing delay by taking the minimum RTT from several samples (to improve its aggressiveness) (§II-A), and the *DP* mechanism ensures that the attacker has sufficient opportunities to feedback Host Delay. Thus, the occasional occurrence of Scenario 2 does not impact the attack, as long as there is an effective Host Delay within the delay-based CCA's sampling window. In other words, for CCAs that use the absolute value of delay, Scenario 2 can be tolerated, with an emphasis on minimizing the impact of Scenario 3. The attacker does not need every Host Delay to fall under Scenario 1, but can instead set the Host Delay using the following formula to achieve the most effective attack:

$$\text{Host Delay}_i = \text{queuing delay} + \text{AbsCorr} \tag{2}$$

$$\text{AbsCorr} = i - \frac{\text{scanning\_range} + 1}{2} \tag{3}$$

where i ranges from 1 to the *scanning_range*, and *scanning_range* defines the number of Host Delay samples (ACK frames) in one scanning cycle. During each cycle, the Host Delay is linearly scanned near the estimated queuing delay, with a 1 ms interval. For instance, with *scanning_range* $= 5$, five consecutive ACK frames are used in a group, and the corresponding Host Delays are calculated for $i = 1, 2, \ldots, 5$. A smaller *scanning_range* requires fewer ACK frames, which results in a shorter scanning duration and better stability of queuing delay changes. Conversely, a larger *scanning_range* allows exploration of a broader range for the optimal Host Delay. This approach ensures that the difference between the feedback Host Delay and the optimal value does not exceed 0.5 ms, and produces an effect similar to an estimated queuing delay with an error of less than 0.5 ms.

2) Attacking CCA Based on the Delay Gradient: Attacking CCAs based on delay gradient is more intricate, as each RTT sample influences the calculation of the RTT gradient. Host Delay deviations (especially Scenario 2) severely degrade attack efficacy by distorting gradient trends. However, gradient-based congestion estimation offers two attacker advantages: First, the RTT gradient is independent of the absolute RTT values, which means that the attacker only needs to ensure a low RTT gradient and can freely adjust the absolute RTT value. Second, the fitting algorithm is insensitive to small fluctuations. In other words, for CCAs based on delay gradients, Scenario 3 is more tolerable, while Scenario 2 severely undermines attack performance. So, the attacker can achieve an effective attack by setting the Host Delay as:

$$\text{Host Delay} = \text{queuing delay} + \text{GradCorr} \tag{4}$$

$$\text{GradCorr} = -\text{ threshold} \tag{5}$$

where *threshold* represents the tolerance for errors in queuing delay estimation. To achieve effective attacks, QUDIT manipulates Host Delay to create near-zero delay gradients. The attacker sets the gradient correction (*GradCorr*) as the negative *threshold*, ensuring the maximum detectable RTT gradient at the sender remains below $threshold/T_g$, where T_g is CCA's gradient detection period length. This triggers the utility-maximizing behavior of PCC, which aggressively escalates rates under low-perceived congestion until packet loss dominates the congestion response. A larger *threshold* enhances tolerance to network jitter and estimation errors, preventing Scenario 2. However, it also relaxes the constraints on detectable RTT gradients at the sender, increasing sensitivity to subtle delay variations. As long as the queuing delay estimation error remains below the *threshold*, the Host Delay will not trigger Scenario 2.

3) Unified Framework and Parameter Selection: While unifying delay-based CCA attacks is challenging due to incompatible optimization objectives, attackers can establish dual QUIC connections to the same delay-based CCA server while forging the Host Delay in different ways. By monitoring the average data arrival rate differential, the attacker dynamically terminates the underperforming connection. The effectiveness of attacks obviously differs, for example, against CCAs using delay gradient, the attack in §IV-B1 fails completely (enabling accurate detection of the RTT gradient). Transmission rate differences primarily stem from the applied Host Delay manipulations since both connections share the same conditions. Normally, attacks co-exist only for a few initial RTTs §V-B. Under exceptional path divergence (e.g., ECMP), maintaining both connections still achieves more than 100 amplification factor with acceptable overhead (estimated from Fig. 5).

Regarding the selection of parameters, *scanning_range* is set to the ACK frame feedback number within $1/4$ RTT, and *threshold* is set to $\frac{\text{RTT}}{10} + 5\,\text{ms}$. As most CCAs' RTT sampling window is more than $\frac{\text{RTT}}{2}$ [13], [15], the $1/4$ factor ensures that regardless of the starting time of the sampling window, at least one scan occurs within it. The *threshold* is set in this way to achieve different objectives: the first term compensates for random network fluctuations, and the second term accounts for the time intervals in the *DP* mechanism, ensuring that even the maximum time interval will not overestimate queuing delay.

V. Evaluation

In this section, we present the validation of QUDIT. We begin by detailing the experimental setup (§V-A). Next, we evaluate the effectiveness of various delay-based CCA attacks in reducing the bandwidth share of the victim flow and the amplification. We then examine the required attack duration and data overhead for the attacker under different conditions (§V-B). In addition to emulation experiments, we conduct physical testbed experiments with 10 Gbps bottleneck links to quantify the efficacy of the attack in various scenarios (§V-C).

A. Experiment Setup

We implement the proposed attack and evaluate its impact on Vegas [13], Copa [15], and PCC Vivace [16] using Mininet network emulator [21] and a physical testbed. Experiments are based on msquic [7], an implementation of the IETF QUIC protocol. As msquic includes only CUBIC [37] and BBR algorithms, we incorporate Copa, PCC, and Vegas refering to mvfast [5] QUIC implementation, Chrome QUIC [38], [39] and Linux kernel [40] separately. We port all three algorithms to msquic and conduct the experiments within this unified framework. During the experiments, the attacker and victim flows used the same adjustable parameters for each algorithm. QUIC flows used msquic's default configuration (1500B MTU) and app-unlimited traffic, focusing on CCA behavior without application bottlenecks.

Fig. 2c illustrates our experimental network topology. For emulation, the bandwidth of the bottleneck link is varied between 20 Mbps and 60 Mbps, and RTT is fixed at 30 ms, with bottleneck buffer size configured to 1 Bandwidth-Delay Product. Results represent averages of multiple runs excluding abnormal connection failures. Each emulation runs for 100 seconds to ensure steady-state convergence. For physical testbed validation, we use 10 Gbps NIC ports (end points) and switch using 10 G SFP+ (bottleneck), in which the buffer can tolerate a queuing delay of approximately 30 ms. However, individual QUIC flow is inherently incapable of saturating 10 Gbps links [41]. We therefore deploy an auxiliary Iperf flow to maintain persistent bottleneck saturation, and scaled competition with six attacker/victim QUIC flows to replicate realistic congestion dynamics (12 QUIC flows, 1 Iperf flow).

B. Attack Effectiveness in Emulated Network Environments

Bandwidth Utilization. We first evaluate QUDIT's impact on throughput degradation of the victim flow in different bandwidths when subjected to attack (Fig. 4). In this scenario, both scapegoat (Server A) and victim (Server B) use identical CCAs, with QUDIT reducing victim B's average bandwidth share compared to baselines without attack. For Vegas, Client A can reduce the victim's bandwidth share ranges from 40% to 60% by using QUDIT, while the attack on Copa leads to a maximum reduction in the victim's average bandwidth of approximately 50%. For Vivace, the victim's average bandwidth decreases by around 20%. Due to random network fluctuations and the use of delay gradients in Vivace, the attacker must set a threshold. As analyzed in §IV-B2, the gradient can only be minimized to $threshold/T_g$ (non-zero). The weaker efficacy also comes in part from stochasticity within Vivace's complex online-learning mechanism. In contrast, for the other two algorithms, which rely on absolute delay values, the attacker can achieve results that are very close to the ideal attack effect using the mechanism described in §IV-B1, leading to relatively better performance.

The attack on the Copa algorithm does not maintain the same effectiveness as the Vegas attack, and its effectiveness decreases with increasing bandwidth. This is due to the dynamic threshold adjustment mechanism in Copa (§II-A),

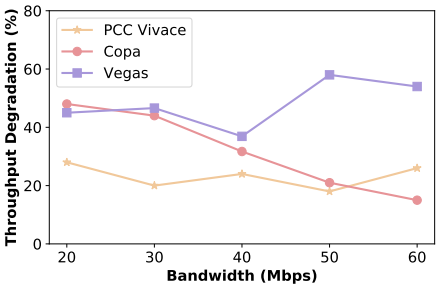


Fig. 4: Throughput degradation with different bottleneck bandwidth

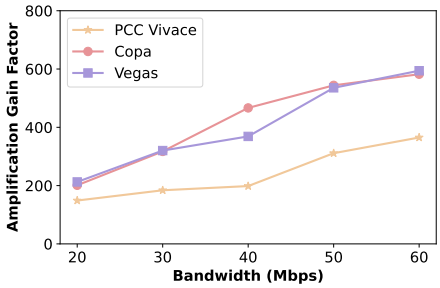


Fig. 5: Amplification Gain with different bottleneck bandwidth.

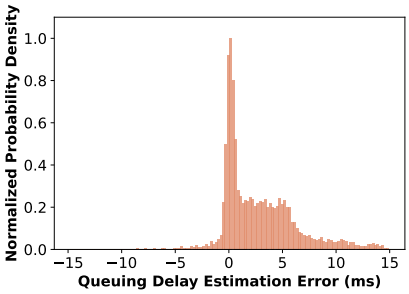
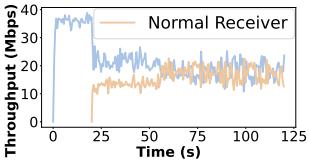
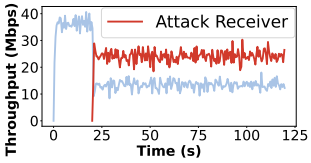


Fig. 6: Normalized Probability density of queuing delay estimation error.

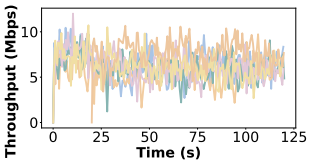


(a) Bandwidth Allocation without Attack

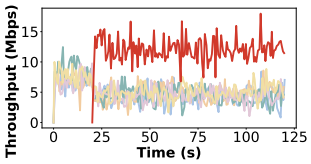


(b) Bandwidth Allocation with Attack

Fig. 7: Attacking a specific victim flow (one victime flow)



(a) Bandwidth Allocation without Attack



(b) Bandwidth Allocation with Attack

Fig. 8: Attacking a key network link (5 victim flows)

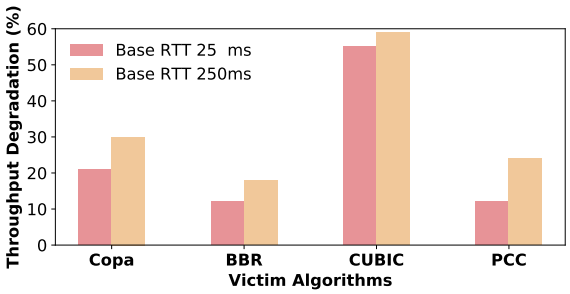


Fig. 9: Throughput Degradation in Physical Testbed Across CCAs: Impact of Base RTT and Victim Congestion Control

which means the attack does not necessarily weaken, but rather Copa in the victim flow becomes more tolerant to delays as the bottleneck bandwidth increases. The attacker can address this issue by extending the RTT. For example, when the RTT is 250ms, the attacker can reduce the victim's average throughput by more than 25% with the Copa algorithm as Fig. 9 .

Amplification Gain. We further investigate the effect of amplification. As shown in Fig. 5, the amplification gain factor refers to the ratio of the amount of data sent by the Scapegoat Server (Server A in Fig. 2c) due to the attack to the amount of data sent by the attacker itself. For attacks targeting different algorithms, we observe amplification gain factors range from about 200 to a maximum of more than 600 across various bandwidth conditions, with this value trending upward as bandwidth increases. This is because, as described in §IV-A, the attacker, acting as the data receiver, needs to send an ACK frame at least once every 5 ms. As bandwidth increases, the amount of data received by the attacker within each 5 ms period also increases, allowing each ACK frame to confirm more data. Consequently, the amplification factor increases. This is particularly advantageous for attacks targeting key network links, as the bandwidth of critical network links may far exceed the bottleneck bandwidth range explored in this experiment. As a result, the amplification factor of the attack will increase further, reducing the attacker's overhead.

Queuing Delay Estimation. Fig. 6 shows deviations of estimated queuing delay by receiver and sender side via §IV-A under 30Mbps-30ms conditions. Sender-side queuing delays derived from current RTT values are optimal estimates for attack. Although network fluctuations cause discrepancies between this estimate and actual delays, by aligning Host Delay with optimal estimates, the attacker can mask buffer queuing measured by the sender completely, maximizing attack efficacy. Approximately 50% of the estimates deviate by < 5 ms from optimal, and only about 5% deviate by > 15 ms despite network jitter. This shows that the estimation scheme in §IV-A provides sufficiently accurate delay estimates, further validating the choice of threshold.

Multi-Victim Attack Efficacy & Efficiency. Figs. 7 and 8 demonstrate QUDIT's impact on single/multiple victim flows under 40Mbps-30ms conditions using Copa. Initial 1/5 victim flows observe 20% bandwidth reduction post-attack initiation (20 s), maintained despite victim scale-up. Our unified framework (§IV-B3) enables transient dual-connection setup without compromising long-term efficacy. QUDIT converges rapidly to over 90% of its steady-state attack effect within approximately 10 RTTs (avg. < 0.3 s), requiring only 0.5 MB scapegoat-side data and tens of kilobytes attacker-side overhead. Results here reflect the maintained attack flow.

C. Attack Effectiveness in Physical Testbed

In this section, we validate QUDIT's effectiveness under high-bandwidth conditions (10 Gbps) using a physical testbed. Fig. 9 illustrates throughput degradation for victims employ-

ing diverse CCAs under different base RTTs, with attackers targeting Scapegoat Servers running Copa.

Cross-CCA Robustness. QUDIT demonstrates consistent effectiveness across victims employing diverse CCAs, including delay-based Copa, congestion-based BBR, loss-based CUBIC and learning-based PCC. As shown in Fig. 9, victims experience about 15%-55% bandwidth degradation under attack, regardless of their CCA.

For delay-based Copa, victims (aligned with emulation results Fig. 4), throughput reduces by about 20%-30%. Loss-based CUBIC suffers the most severe impact (50%-60% degradation): manipulated Host Delays prevent Copa from detecting actual queuing delays, forcing aggressive data transmission that triggers bufferbloat and packet loss. CUBIC drastically reduces its CWND upon detecting losses, being attacked the most severely. In contrast, BBR periodically probes the bandwidth, while PCC leverages multiple congestion signals calculating utility to adaptively balance throughput and delay. As a result, they both maintain more robustness under attack. Unfortunately, the dominance of the most vulnerable CUBIC in real-world deployments (about 40% of Alexa top 20k sites using CUBIC [22]) amplifies the practical threat of QUDIT.

BaseRTT-Dependent Efficacy. Fig. 9 also shows attack effectiveness escalates with increasing base RTT: the reduction under 250 ms base RTT is about 10% higher compared to 25 ms scenarios, with consistent efficacy trends observed in all types of victim CCA. This occurs because larger base RTTs reduce the relative magnitude of queuing delay estimation errors, thereby enhancing Copa's tolerance to minor residual delays that persist due to incomplete Host Delay masking. Notablly, even when *max_ack_delay* is constrained to $1/10$ of base RTT (25 ms for 250 ms RTT), QUDIT maintains obvious effects, proving such restrictions insufficient. The root vulnerability lies in QUIC's current *max_ack_delay* design: (1) Default 25 ms values roughly match the max queuing delays, giving attackers excessive manipulation headroom. (2) Attackers can arbitrarily declare a large *max_ack_delay* to maximize the variable range of Host Delay.

VI. Discussion

A. Delay Measurement in other Protocols

To accurately measure delay, various protocols incorporate mechanisms in which the data receiver assists in latency measurement. In addition to the discussed QUIC's Host Delay design, the Google Congestion Control Algorithm (GCC) [42], [43] used in WebRTC [44] also relies on the receiver to assist in measuring latency. GCC determines the network's congestion state from one-way delay gradients, necessitating the receiver to calculate the time interval between the arrival of two consecutive groups of data packets.

B. Attack Defenses

Several defense mechanisms can be implemented to mitigate the potential threats posed by our attack.

From the QUIC protocol perspective, the *max_ack_delay* parameter, which is declared by the data receiver, cannot be directly controlled or validated by the QUIC server, creating a large attack vector for malicious manipulation of Host Delay to disrupt the RTT measurements. We propose protocol layer modifications via RFC revision: transferring control to senders and enforcing sender-imposed constraints on *max_ack_delay* to limit the range of adjustments that an attacker can make. However, the configuration of *max_ack_delay* presents a trade-off: stricter bounds mitigate QUDIT effectiveness, but may limit QUIC's applicability to resource-constrained devices.

Consequently, QUIC maintainers can prioritize tighter constraints in security-sensitive deployments while permitting relaxed values in controlled environments. To avoid imposing bounds of *max_ack_delay* that risk excluding resource-constrained endpoints, maintainers may instead deploy localized countermeasures: disable Host Delay locally (though may reduce RTT accuracy) or validate Host Delay and dynamically weight its usage as a measured alternative. This preserves deployment pragmatism without mandating universal bounds.

Moreover, from the network traffic manager, using fine-grained fair queuing mechanisms [45] can significantly limit the effect of these attacks. Weighted fair queuing (WFQ) could be deployed at bottleneck links (e.g., ingress/egress of QUIC-enabled data centers) to isolate malicious flows, alleviate congestion and mitigate attack impact. By allocating separate buffers and bandwidth resources for each flow, fair queuing mechanisms ensure a fair distribution of network resources, preventing any single flow from monopolizing resources.

C. Ethical considerations

In accordance with ethical principles, we carefully conducted experiments within controlled environments to ensure no public internet impact. Regarding the potential vulnerability in QUIC's Host Delay mechanism and its defenses, we have: (1) reported this threat and proposed protocol revisions to the IETF, and (2) coordinated with QUIC implementation maintainers through responsible disclosure.

VII. Conclusion

QUIC has become a prominent transport protocol, offering low-latency, encrypted communication and strong performance. QUIC's Host Delay mechanism improves RTT estimation for delay-based congestion control but introduces security vulnerabilities. In this paper, we present QUDIT, a novel receiver-driven attack that manipulates Host Delay values to deceive delay-based CCAs, causing congestion underestimation and excessive traffic. Our evaluation shows that QUDIT can reduce victim throughput by up to 60% within 0.3 seconds, with an amplification gain of 200 to over 600, and we propose mitigation strategies such as stricter bounds on QUIC's *max_ack_delay* parameter.

ACKNOWLEDGMENT

We sincerely thank our anonymous shepherd and the reviewers for their valuable feedback. The research was supported by the National Natural Science Foundation of China under Grant 62221003. Mingwei Xu, Enhuan Dong, and Jia Zhang are the corresponding authors.

REFERENCES

[1] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi, "The quic transport protocol: Design and internet-scale deployment," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, (New York, NY, USA), p. 183–196, Association for Computing Machinery, 2017.

[2] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport." RFC 9000, May 2021.

[3] J. Iyengar and I. Swett, "QUIC Loss Detection and Congestion Control." RFC 9002, May 2021.

[4] Google Inc., "Quiche: Google's production-ready implementation of quic." https://github.com/google/quiche, 2024. Accessed: December 26, 2024.

[5] META Inc., "mvfst: an implementation of the ietf quic protocol by facebook." https://github.com/facebook/mvfst, 2024. Accessed: December 26, 2024.

[6] Apple Inc., "Network.framework." https://developer.apple.com/documentation/network, 2024. Accessed: December 26, 2024.

[7] Microsoft Inc., "Msquic: Microsoft's implementation of the ietf quic protocol." https://github.com/microsoft/msquic, 2024. Accessed: December 26, 2024.

[8] M. B. Ian Swett, "Introducing quic support for https load balancing." https://cloud.google.com/blog/products/gcp/introducing-quic-support-https-load-balancing, 2018. Accessed: January 3, 2025.

[9] Y. C. Matt Joras, "How facebook is bringing quic to billions." https://engineering.fb.com/2020/10/21/networking-traffic/how-facebook-is-bringing-quic-to-billions/, 2020. Accessed: January 3, 2025.

[10] Microsoft Inc., "Quic support in .net." https://learn.microsoft.com/en-us/dotnet/fundamentals/networking/quic/quic-overview#quic-in-net, 2024. Accessed: January 3, 2025.

[11] A. Mishra and B. Leong, "Containing the cambrian explosion in quic congestion control," in *Proceedings of the 2023 ACM on Internet Measurement Conference*, IMC '23, (New York, NY, USA), p. 526–539, Association for Computing Machinery, 2023.

[12] T. Meng, N. R. Schiff, P. B. Godfrey, and M. Schapira, "Pcc proteus: Scavenger transport and beyond," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, (New York, NY, USA), p. 615–631, Association for Computing Machinery, 2020.

[13] L. S. Brakmo, S. W. O'malley, and L. L. Peterson, "Tcp vegas: New techniques for congestion detection and avoidance," in *Proceedings of the conference on Communications architectures, protocols and applications*, pp. 24–35, 1994.

[14] S. Shalunov, G. Hazel, J. Iyengar, and M. Kühlewind, "Low Extra Delay Background Transport (LEDBAT)." RFC 6817, Dec. 2012.

[15] V. Arun and H. Balakrishnan, "Copa: Practical Delay-Based congestion control for the internet," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, (Renton, WA), pp. 329–342, USENIX Association, Apr. 2018.

[16] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, "PCC vivace: Online-Learning congestion control," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, (Renton, WA), pp. 343–356, USENIX Association, Apr. 2018.

[17] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "Timely: Rtt-based congestion control for the datacenter," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, (New York, NY, USA), p. 537–550, Association for Computing Machinery, 2015.

[18] webrtc.org (Google Inc.), "Congestion Controller in WebRTC." https://source.chromium.org/chromium/chromium/src/+/main:third_party/webrtc/modules/congestion_controller/?q=congestion_controller, 2025. Accessed: January 3, 2025.

[19] N. Garg, "Evaluating COPA congestion control for improved video performance." https://code-dev.fb.com/2019/11/17/video-engineering/copa/, 2019. Accessed: January 3, 2025.

[20] C. Holmberg, S. Hakansson, and G. Eriksson, "Web Real-Time Communication Use Cases and Requirements." RFC 7478, Mar. 2015.

[21] B. Lantz, B. Heller, and N. McKeown, "Mininet." [Online]. Available: http://mininet.org/, 2022. Accessed: January 3, 2025.

[22] A. Mishra, L. Rastogi, R. Joshi, and B. Leong, "Keeping an eye on congestion control in the wild with nebby," in *Proceedings of the ACM SIGCOMM 2024 Conference*, ACM SIGCOMM '24, (New York, NY, USA), p. 136–150, Association for Computing Machinery, 2024.

[23] M. Sooriyabandara, G. Fairhurst, V. Padmanabhan, and H. Balakrishnan, "TCP Performance Implications of Network Path Asymmetry." RFC 3449, Dec. 2002.

[24] S. Jero, M. E. Hoque, D. R. Choffnes, A. Mislove, and C. Nita-Rotaru, "Automated attack discovery in tcp congestion control using a model-guided approach.," in *NDSS*, 2018.

[25] Z. Yang, J. Cao, Z. Liu, X. Zhang, K. Sun, and Q. Li, "Good learning, bad performance: A novel attack against rl-based congestion control systems," *IEEE Transactions on Information Forensics and Security*, vol. 17, pp. 1069–1082, 2022.

[26] A. Peterson, S. Jero, E. Hoque, D. Choffnes, and C. Nita-Rotaru, "aBBRate: Automating BBR attack exploration using a Model-Based approach," in *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, (San Sebastian), pp. 225–240, USENIX Association, Oct. 2020.

[27] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson, "Tcp congestion control with a misbehaving receiver," *SIGCOMM Comput. Commun. Rev.*, vol. 29, p. 71–78, Oct. 1999.

[28] M. Thomson and S. Turner, "Using TLS to Secure QUIC." RFC 9001, May 2021.

[29] C. Fu, Q. Li, and K. Xu, "Detecting unknown encrypted malicious traffic in real time via flow interaction graph analysis," *arXiv preprint arXiv:2301.13686*, 2023.

[30] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. V. Ramos, and A. Madeira, "Flowlens: Enabling efficient flow classification for ml-based network security applications," *Proceedings 2021 Network and Distributed System Security Symposium*, 2021.

[31] D. Senie and P. Ferguson, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing." RFC 2827, May 2000.

[32] M. S. Kang, S. B. Lee, and V. D. Gligor, "The crossfire attack," in *2013 IEEE Symposium on Security and Privacy*, pp. 127–141, 2013.

[33] A. Studer and A. Perrig, "The coremelt attack," in *Proceedings of the 14th European Conference on Research in Computer Security*, ESORICS'09, (Berlin, Heidelberg), p. 37–52, Springer-Verlag, 2009.

[34] D. Wagner, D. Kopp, M. Wichtlhuber, C. Dietzel, O. Hohlfeld, G. Smaragdakis, and A. Feldmann, "United we stand: Collaborative detection and mitigation of amplification ddos attacks at scale," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, (New York, NY, USA), p. 970–987, Association for Computing Machinery, 2021.

[35] S. Ha, I. Rhee, and L. Xu, "Cubic: a new tcp-friendly high-speed tcp variant," *SIGOPS Oper. Syst. Rev.*, vol. 42, p. 64–74, July 2008.

[36] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: congestion-based congestion control," *Commun. ACM*, vol. 60, p. 58–66, Jan. 2017.

[37] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks." RFC 8312, Feb. 2018.

[38] M. Dong and X. Zhu, "PCC for QUIC." https://github.com/modong/chrome_quic, 2015. Accessed: January 3, 2025.

[39] T. Meng, "The userspace implementations of PCC.." https://github.com/PCCproject/PCC-Uspace, 2021. Accessed: January 3, 2025.

[40] L. Torvalds, "Linux Kernel.." https://github.com/torvalds/linux, 2025. Accessed: January 3, 2025.

[41] N. Tyunyayev, M. Piraux, O. Bonaventure, and T. Barbette, "A high-speed quic implementation," in *Proceedings of the 3rd International CoNEXT Student Workshop*, CoNEXT-SW '22, (New York, NY, USA), p. 20–22, Association for Computing Machinery, 2022.

[42] S. Holmer, H. Lundin, G. Carlucci, L. D. Cicco, and S. Mascolo, "A Google Congestion Control Algorithm for Real-Time Communication," Internet-Draft draft-ietf-rmcat-gcc-02, Internet Engineering Task Force, July 2016. Work in Progress.

[43] G. Carlucci, L. De Cicco, S. Holmer, and S. Mascolo, "Analysis and design of the google congestion control for web real-time communication (webrtc)," in *Proceedings of the 7th International Conference on*

Multimedia Systems, MMSys '16, (New York, NY, USA), Association for Computing Machinery, 2016.
[44] H. T. Alvestrand, "Transports for WebRTC." RFC 8835, Jan. 2021.
[45] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *Symposium Proceedings on Communications Architectures & Protocols*, SIGCOMM '89, (New York, NY, USA), p. 1–12, Association for Computing Machinery, 1989.