Exploiting Congestion Control Parameter Manipulation in QUIC for Security Implications

Y A Joarder
CIISE
Concordia University
Montreal, Canada
ya.joarder@concordia.ca

Surajit Sinha
CIISE
Concordia University
Montreal, Canada
surajit.sinha@mail.concordia.ca

Guillaume Doyen
IRISA
IMT Atlantique
Rennes, France
guillaume.doyen@imt-atlantique.fr

Carol Fung
CIISE
Concordia University
Montreal, Canada
carol.fung@concordia.ca

***Abstract*—QUIC has emerged as a fundamental transport protocol for modern Internet infrastructure, serving as the foundation for HTTP/3. Although QUIC implements congestion control algorithms (*CCA*) to ensure fair network resource allocation, its user-space implementation architecture creates significant security vulnerabilities through accessible parameter manipulation. As transport layers become increasingly programmable, these vulnerabilities represent a broader security challenge for future network infrastructures where applications may deploy custom transport implementations. This paper presents a systematic analysis of selfish behaviors in QUIC through deliberate congestion control parameter (*CCPM*). Using the *aioquic* implementation, we experimentally demonstrate how strategic parameter manipulation in both NewReno and CUBIC algorithms provides substantial unfair bandwidth *(BW)* advantages. NewReno exhibits a major vulnerability with *Loss Reduction Factor (LRF)* and *Congestion Avoidance Growth Rate (CAGR)* manipulation, while CUBIC demonstrates better resilience, but remains exploitable, with combined *LRF* (β_{cubic}) and *Maximum Idle Time (MIT)* manipulations.**

***Index Terms*—QUIC, Congestion Control, Selfish Behavior, NewReno; CUBIC, Network Fairness, Transport Protocols, QoS, CCPM Attack, Programmable Transport Layer, QUIC Security**

I. Introduction

QUIC is a transformative transport protocol that addresses TCP limitations through enhanced performance and security mechanisms [1]. Evolving from Google's experimental implementation in 2012 to IETF's standardized RFC 9000 in 2021 [2], QUIC represents a fundamental paradigm shift in transport architecture [3]. Built on the top of UDP, QUIC implements connection-oriented semantics with reduced latency, enhanced reliability, and embedded TLS 1.3 security to overcome TCP's limitations in high-latency, loss-prone networks [4], [5]. Since Chrome's initial deployment of QUIC in 2014 [6], QUIC has achieved substantial Internet-scale adoption, with approximately 35% of websites supporting HTTP/3 and an additional 8.6% supporting QUIC for other applications [7]. This widespread deployment is particularly evident in multimedia applications such as real-time streaming and interactive services, with major technology companies including Google, Meta, Microsoft, Cloudflare, and Akamai integrating QUIC into their production infrastructure [4].

A critical aspect of QUIC's performance is its flexible CCA framework, which allows different algorithms to be implemented and configured [8]. Although RFC 9002 specifies a CCA similar to TCP NewReno [8], most implementations default to CUBIC, with BBR available as an optional deployment [9]. QUIC's performance and fairness critically depend on its configurable parameters that vary across CCA [3], [9]. However, QUIC's user-space implementation [3], [10] enables rapid deployment [6] and exemplifies how programmable transport layers allow applications to implement custom transport mechanisms independent of kernel constraints [11]. This design approach introduces new security concerns, as demonstrated in this investigation, where transport layer protocol (QUIC) implementation (e.g. *aioquic* [12]) becomes vulnerable to our proposed *CCPM* (Congestion Control Parameter Manipulation) attack. Unlike kernel-based TCP implementations, where parameter modification requires administrative privileges [13], QUIC's user-space nature may enable selfish clients to manipulate congestion control parameters more easily [10]. This accessibility may give selfish clients an opportunity to gain unfair bandwidth advantages through aggressive parameter tuning. Such behavior can potentially cause bandwidth starvation and, in extreme cases, denial-of-service (*DoS*) conditions for normal clients on shared links.

Although extensive research has examined selfish behavior and parameter manipulation in TCP environments [13]–[15], and other security issues in QUIC [16]–[19], comprehensive investigations of congestion control parameter manipulation in QUIC implementations are still lacking. Although TCP-focused congestion control studies [13]–[15] are limited to theoretical analysis due to kernel space implementation constraints [20], QUIC's user space architecture with pluggable congestion control [2], [8] enables direct parameter manipulation. This creates new security vulnerabilities, such as the *CCPM* attack, which is not present in traditional TCP implementations. To address this critical knowledge gap, this research systematically investigates the manipulation of congestion control parameters in QUIC and its implications for network fairness. We conducted comprehensive experiments with NewReno and CUBIC algorithms using the *aioquic* implementation [12] in controlled network environments. Our experimental results demonstrate that NewReno exhibits extreme vulnerability with *Loss Reduction Factor (LRF)* and *Congestion Avoidance Growth Rate (CAGR)* manipulation, allowing monopolization of up to around 85% of available bandwidth at network bottlenecks. CUBIC demonstrates better resilience due to its cubic window growth function. However, it

978-3-903176-75-1 ©2025 IFIP

remains exploitable with combined *LRF* (β_{cubic}) and *Maximum Idle Time (MIT)* manipulation, which allows a selfish client to gain up to 59% of bandwidth resources while degrading the *Quality of Service* (*QoS*) of the normal (legitimate) client.

The contribution of this paper can be summarized as follows:

- We present the first empirical study examining the exploitability of QUIC CCA (NewReno and CUBIC) from the security perspective;
- We developed a testbed based on *aioquic* to evaluate the gains of selfish QUIC clients through manipulating congestion control parameters;
- Our empirical study reveals a major vulnerability that selfish clients can achieve significant bandwidth *(BW)* advantages through strategic parameter manipulation, reveals multiplicative vulnerability effects.

The remainder of this paper is organized as follows. Section II reviews the background and related work. Section III details our empirical methodology and evaluation. Section IV presents the overall discussion. Section V concludes the paper.

II. Background and Related Work

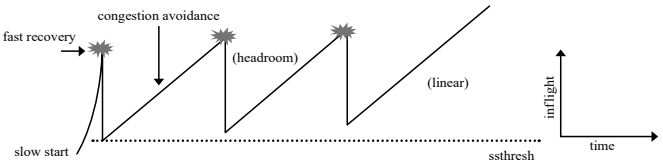
QUIC's congestion control architecture provides enhanced bandwidth management through three key design principles. First, enhanced ACK frames deliver richer feedback than conventional TCP acknowledgments, allowing more precise transmission rate adjustments [8]. Second, seamless connection migration preserves the congestion control state across network transitions, maintaining optimization when connections move between interfaces [2]. Third, flexible algorithm selection allows endpoints to deploy specialized congestion control beyond baseline specifications [8]. QUIC supports many CCA. Among them, NewReno and CUBIC [8] are widely adopted by QUIC implementations.

A. NewReno Congestion Control Algorithm

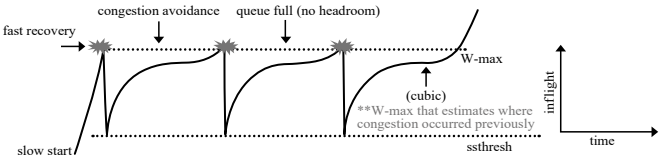
NewReno serves as QUIC's reference CCA [8], operating through three synchronized states: Slow Start, Congestion Avoidance, and Fast Recovery. As shown in Figure 1a, the algorithm begins with exponential window growth in Slow Start until reaching the slow start threshold (*ssthresh*) or detecting packet loss. It then transitions to linear growth in Congestion Avoidance, increasing by one maximum datagram size per round trip time (*RTT*). Upon packet loss, Fast Recovery immediately halves the congestion window and updates *ssthresh*, creating a feedback loop where the algorithm returns to Congestion Avoidance with adjusted parameters. This tri-state coordination enables adaptive transmission rate control while maintaining network stability [21].

B. CUBIC Congestion Control Algorithm

CUBIC employs a time-based cubic function, $W(t) = C(t-K)^3 + W_{max}$, where C is the scaling constant, K represents the time to reach W_{max}, and W_{max} is the window size at the previous congestion event [22]. Unlike NewReno's *RTT*-dependent approach, CUBIC synchronizes with elapsed



(a) NewReno Algorithm Behavior



(b) CUBIC Algorithm Behavior

Fig. 1: Congestion Window Behavior Comparison Over Time

time since congestion, creating the concave-convex pattern shown in Figure 1b. The algorithm operates through three phases: conservative growth approaching W_{max} (concave), stabilization at the inflection point, and aggressive bandwidth probing beyond W_{max} (convex). This time-based coordination enables consistent behavior regardless of *RTT* variations, making CUBIC suitable for high-speed networks while balancing caution near known congestion points with aggressive exploration of new bandwidth opportunities [22].

C. Selfish Behavior studies

Studies on selfish behavior in transport protocols has primarily focused on TCP environments. Akella et al. [13] pioneered game-theoretic analysis of TCP congestion control, demonstrating that users can achieve significant gains through strategic manipulation of additive increase and multiplicative decrease (*AIMD*) parameters. Their work established that protocol design decisions fundamentally influence network stability under selfish user behavior. In addition, Zhang et al. [14] extended this analysis through the *TCP Connection Game*, quantifying how opening multiple concurrent connections could lead to potentially unbounded efficiency losses. Chen et al. [15] provided contemporary frameworks for analyzing competing TCP implementations, demonstrating how selfish behavior leads to suboptimal network utilization while identifying potential mechanisms to maintain efficiency.

However, these TCP-focused congestion control related studies [13]–[15] face a fundamental limitation in their applicability to QUIC due to architectural differences between the protocols. While previous research has been primarily constrained to theoretical analysis of TCP behavior, examining generalized *AIMD* parameters through analytical models and simulation studies [23], [24] rather than actual parameter manipulation, due to TCP's kernel-space implementation where the CCA requires kernel recompilation for updates [20], QUIC's user-space architecture fundamentally transforms this landscape [2]. Its congestion control implementation enables easy upgrades and supports configuration of different CCA for applications [8], allowing future changes to be made without

kernel modifications. This architectural shift from theoretical modeling of fixed kernel implementations to direct algorithm customization expands the scope for potential manipulation far beyond the simulation-based parameter studies conducted for TCP environments.

Although multiple studies have focused on QUIC performance optimization through legitimate parameter tuning, no research work has systematically investigated the security implications of deliberate parameter manipulation for unfair bandwidth advantage. Letourneau et al. [25] demonstrated early exploitation of QUIC's CCA in *Low Latency, Low Loss and Scalable Throughput* (*L4S*) environments through *ECN*-based attacks targeting low-latency flows, though their work focused on specific *L4S* vulnerabilities rather than systematic manipulation strategies. Kharat and Kulkarni [26] achieved 35% throughput increases through congestion window optimizations, while Han et al. [9] demonstrated 10.87% higher throughput with modified parameters for performance enhancement purposes. These studies, however, operate under the assumption of benevolent usage and focus solely on legitimate performance improvements rather than analyzing exploitation potential.

The complexity of detecting implementation-level parameter differences is underscored by implementation diversity across QUIC stacks [27]. Mishra et al. [27] found significant implementation divergence across QUIC implementations, making it challenging to distinguish between legitimate implementation variations and potentially malicious parameter modifications. Their follow-up work [10] revealed that major providers deploy custom congestion control variants that deviate from standard implementations. This implementation flexibility, combined with user-space accessibility, creates an unprecedented attack surface that has not been systematically analyzed yet. Moreover, the trend toward programmable transport layers [28] suggests that applications may soon deploy custom transport implementations, potentially introducing untrusted mechanisms into network environments. Such developments extend the security concerns identified in this work beyond QUIC to any custom application-defined protocol.

III. Empirical Methodology and Evaluation

We adopted an empirical experimental approach because QUIC's congestion control exhibits complex nonlinear dynamics that cannot be predicted theoretically, particularly with simultaneous parameter interactions, and its user-space implementation creates behaviors requiring empirical validation [29]. Unlike TCP/kernel, parameter manipulation is realistic in QUIC since user-space implementations give applications direct control over congestion control parameters, enabling selfish clients to modify parameters for unfair bandwidth advantages. To study the exploitability of QUIC congestion control, we built our experiments on *aioquic*, a *Python*-based QUIC implementation incorporating NewReno [30] and CUBIC [22] algorithms [12]. We focus on these algorithms as they represent different paradigms: NewReno is a traditional loss-based algorithm with linear growth [30], and CUBIC is an advanced loss-based algorithm with polynomial window growth [22]. In contrast, BBR [9] is a model-based algorithm that uses bandwidth and round-trip time measurements rather than packet loss for congestion detection [31]. While BBR represents an important alternative approach, we focus on NewReno and CUBIC as they share loss-based principles that allow systematic parameter manipulation analysis. We systematically evaluate how parameter modifications can be exploited for unfair bandwidth advantages, using unified terminology with *LRF* denoting loss reduction factor for both NewReno's Fast Recovery and CUBIC's multiplicative decrease (β_{cubic}) as implemented in *aioquic* [12]. In this section, we have evaluated QUIC congestion control manipulation through experimental setup, parameter selection and setting, NewReno & CUBIC performance analysis and cross-algorithm comparison.

TABLE I: Network Configuration Parameters' Details

Virtual Machine	Network Interface	IP Address
Client-1 VM (Normal)	enp0s8	192.168.56.102
Client-2 VM (Selfish)	enp0s8	192.168.56.103
Server VM	enp0s8	192.168.57.101
Router VM	enp0s8 (Client-facing)	192.168.56.50
Router VM	enp0s9 (Server-facing)	192.168.57.1

A. Experimental Setup

Our experimental setup consists of a unitary testbed comprising four *Linux Ubuntu 24.04.2* Virtual Machines (VMs) deployed in *Oracle VirtualBox version 7.1.6*, a hosted hypervisor, on an *ASUS VivoBook* laptop (*AMD Ryzen 7 5800HS, 16GB RAM*). The testbed includes three experimental VMs and one router VM configured with *Linux Traffic Control (tc)* and *netem* to control network conditions, as illustrated in Figure 2. Experiments were conducted across multiple physical locations using identical virtual topologies. The specific network configuration parameters are detailed in Table I. The implementation utilized the *Hierarchical Token Bucket* (*HTB*) [32] queuing discipline configured with a 15 Mbps bandwidth limit, creating a bottleneck scenario that compels client *VM*s to compete for network resources and enables direct comparison of CCA under resource contention. The *HTB* configuration incorporates burst parameters that permit temporary token-bucket borrowing beyond the configured rate, thereby accommodating the inherently bursty characteristics of network traffic. Complementing the bandwidth limitation, the *netem* discipline introduces a 15 ms propagation delay and 0.3% packet loss rate to emulate authentic network impairments [33]. It should be noted that instantaneous bandwidth measurements may occasionally exceed the 15 Mbps threshold due to burst allowances and the measurement granularity inherent in the *Linux tc with netem* framework. This carefully selected combination of parameters establishes an optimal experimental environment for evaluating the performance of *CCA* under realistic network constraints.

B. Parameter Selection and Setting

After reviewing CCAs, we have identified specific parameters in the NewReno and CUBIC algorithms that could impact

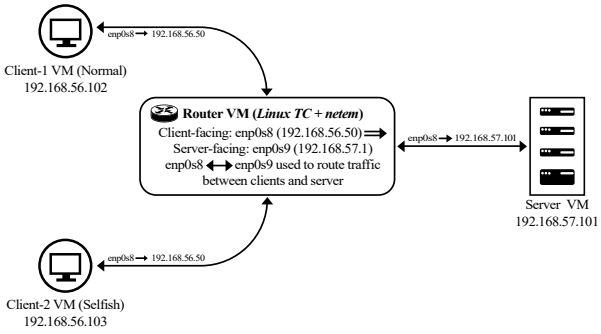


Fig. 2: Network Topology Overview

in the aggressiveness of the congestion control, allowing the selfish client to gain advantage in bandwidth allocation.

1) NewReno Manipulation Parameters: Two critical parameters in the NewReno algorithm were selected for analysis: *LRF* controls the congestion window reduction following packet loss events, while *CAGR* determines window expansion during stable transmission periods. Normal clients employ standard values (*LRF*=0.5, *CAGR*=1 *MSS* (Maximum Segment Size) per *RTT*, while selfish clients utilize modified ranges (*LRF*=0.6-0.9, *CAGR*=2-6 *MSS* per *RTT*).

2) CUBIC Manipulation Parameters: For CUBIC algorithm analysis, we have also focused on two parameters, namely, *LRF* and *MIT*. *LRF* controls window reduction during congestion events, while *MIT* governs CUBIC's characteristic cubic growth function. Higher *MIT* values accelerate CUBIC's window growth phase, allowing selfish clients to capture bandwidth more aggressively, while elevated *LRF* values result in less aggressive window reduction during packet loss events, enabling faster recovery and sustained higher bandwidth. Normal clients operate with default parameters (*LRF*=0.7, *MIT*=2.0), while selfish clients employ elevated values (*LRF*=0.75-0.99, *MIT*=2.5-3.5).

C. NewReno Exploitation Results and Analysis

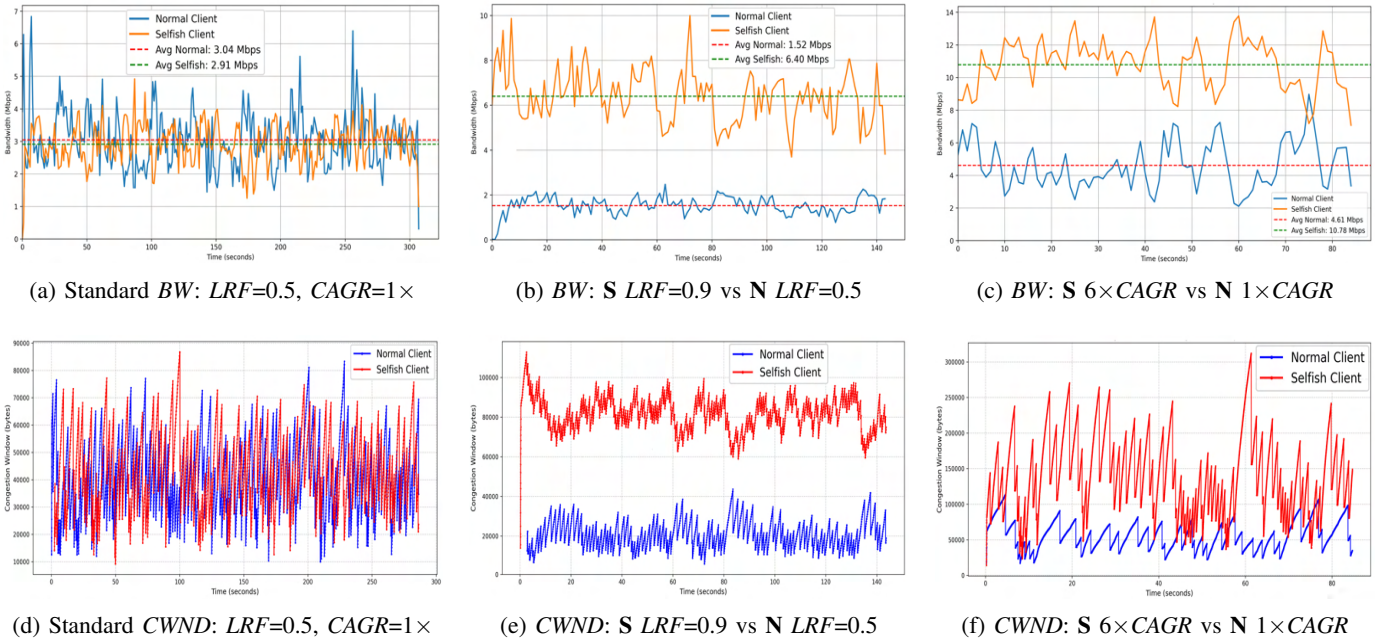
1) Loss Reduction Factor (LRF) Manipulation: We conducted controlled experiments using the experimental (*VM*) setup described in Section III-A with simultaneous 100 MB file uploads from both clients (normal and selfish). We tested one baseline configuration (both clients at *LRF*=0.5) and four manipulation scenarios where the selfish client used *LRF* values of 0.6, 0.7, 0.8, and 0.9 while the normal client remained at *LRF*=0.5. Each configuration was repeated 20 times, capturing separate PCAP and JSON files for each client during each trial. Post-processing extracted *BW* measurements from PCAP files and congestion window *(CWND)* data from JSON files, generating a CSV dataset (file) containing 20 samples with *BW* measurements, *BW* ratios, average and maximum *CWND* values, and *CWND* ratios for each *LRF* configuration. In addition, we have performed two types of statistical analysis on the CSV dataset using 95% confidence intervals (CI) with t-distribution: *BW* allocation analysis comparing normal versus selfish client performance across various *LRF* values, and *BW* advantage analysis, quantifying the relative unfairness *BW* gained through manipulation.

Under std. baseline conditions, NewReno demonstrates fair allocation with equitable *BW* distribution between clients (Figure 3a) and balanced *CWND* behavior (Figure 3d). However, manipulation scenarios reveal severe unfairness escalation. As selfish client *LRF* increases (*LRF*=0.9), dramatic inequity emerges with the selfish client capturing significantly more *BW* while constraining the normal client (Figure 3b). The selfish client maintains substantially larger *CWND* values by reducing windows less aggressively during loss events compared to the normal client (Figure 3e). The *BW* allocation analysis reveals a clear inverse relationship where selfish client *BW* increases progressively while normal client *BW* diminishes correspondingly across *LRF* values (n=20, 95% CI) (Figure 4a). The *BW* advantage analysis demonstrates that the selfish client achieved dramatically escalating and nonlinear *BW* advantages over the normal client through *LRF* manipulation, fundamentally undermining network equity (n=20, 95% CI) (Figure 4d).

2) Congestion Avoidance Growth Rate (CAGR) Manipulation: We conducted controlled experiments using the same experimental (VM) setup and methodology. We tested one standard baseline configuration (both clients at *CAGR*=1×) and manipulation scenarios where the selfish client used *CAGR* values ranging from 1× to 6× while the normal client remained at standard *CAGR*=1×. Each configuration was repeated 20 times, following identical data collection and statistical analysis procedures with *BW* measurements extracted from PCAP files and *CWND* data from JSON files, generating a CSV dataset containing 20 samples with *BW* measurements, *BW* ratios, average and maximum *CWND* values, and *CWND* ratios for each *CAGR* configuration.

Under baseline conditions, both clients demonstrate equitable *BW* distribution (Figure 3a). However, *CAGR* manipulation (6×) reveals severe resource monopolization where the selfish client captures substantially more *BW* compared to the normal client (Figure 3c). The underlying mechanism shows the selfish client maintaining considerably larger *CWND* values, enabling aggressive *BW* acquisition between congestion events (Figure 3f). The *BW* allocation analysis demonstrates distribution shifting from near-equality at baseline to marked disparity at higher *CAGR* values, where selfish client *BW* increases progressively while normal client *BW* diminishes correspondingly (n=20, 95% CI) (Figure 4b). The *BW* advantage analysis reveals that the selfish client achieve progressive exploitation growth from baseline equality to substantial *BW* advantages over the normal client at maximum growth rates (n=20, 95% CI) (Figure 4e). This manipulation is most effective when there are few packet losses, because the selfish client can keep growing its window size for longer periods without interruption, leading to more unfair *BW* distribution.

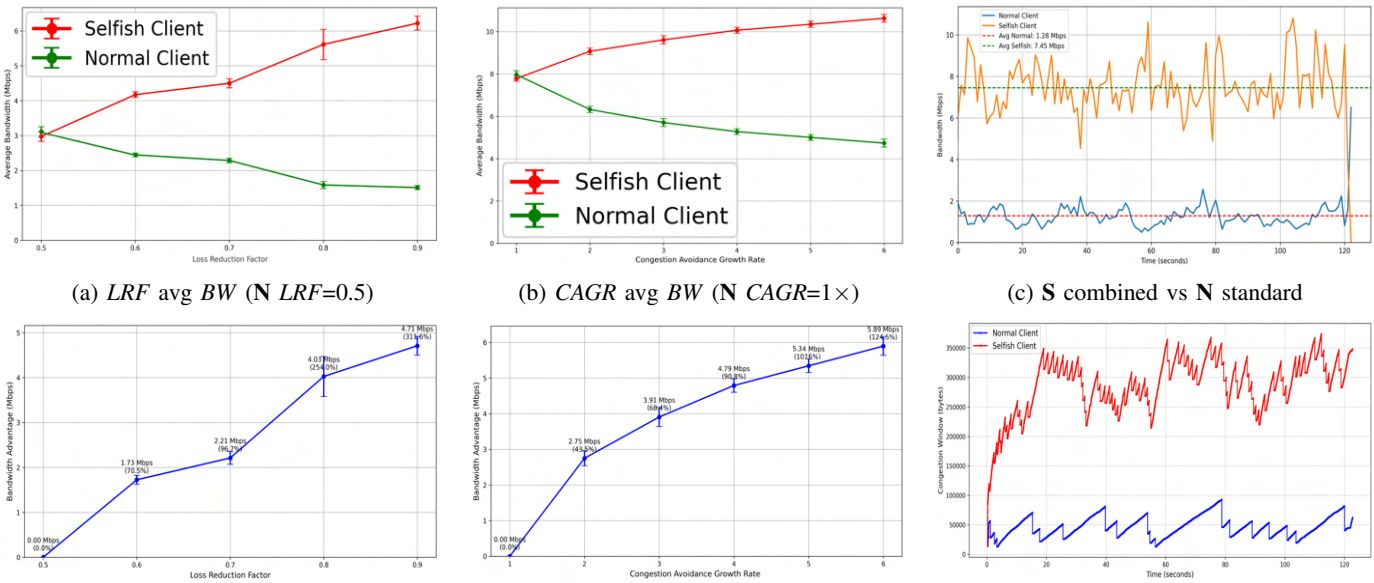
3) Combined Parameters (LRF+CAGR) Manipulation: We conducted controlled experiments using the same VM setup and methodology to investigate simultaneous manipulation of multiple parameters and evaluate combined exploitation effects. We tested scenarios where the selfish client em-



(a) Standard *BW*: *LRF*=0.5, *CAGR*=1× (b) *BW*: **S** *LRF*=0.9 vs **N** *LRF*=0.5 (c) *BW*: **S** 6×*CAGR* vs **N** 1×*CAGR*

(d) Standard *CWND*: *LRF*=0.5, *CAGR*=1× (e) *CWND*: **S** *LRF*=0.9 vs **N** *LRF*=0.5 (f) *CWND*: **S** 6×*CAGR* vs **N** 1×*CAGR*

Fig. 3: NewReno parameter manipulation analysis showing *BW* and *CWND* behavior (**S**=Selfish, **N**=Normal)



(a) *LRF* avg *BW* (**N** *LRF*=0.5) (b) *CAGR* avg *BW* (**N** *CAGR*=1×) (c) **S** combined vs **N** standard

(d) **S** *BW* advantage (*LRF*) (e) **S** *BW* advantage (*CAGR*) (f) **S** combined vs **N** standard

Fig. 4: NewReno statistical validation and combined manipulation effects on *BW* and *CWND* (**S**=Selfish, **N**=Normal)

ployed both *LRF*=0.9 and *CAGR*=6× simultaneously while the normal client maintained standard parameters (*LRF*=0.5, *CAGR*=1×). Each configuration was repeated 20 times, following identical data collection with *BW* measurements extracted from PCAP files and *CWND* data from JSON files. Combined parameter manipulation reveals catastrophic fairness collapse that exceeds individual vulnerabilities. The selfish client achieves severe *BW* monopolization, capturing approximately 85% of available *BW* while the normal client receives substantially reduced capacity (Figure 4c). The underlying mechanism shows the selfish client maintaining dramatically larger *CWND* values compared to the normal client, creating persistent capacity advantage through compounding effects (Figure 4f). In the NewReno algorithm, this combined exploitation significantly surpasses individual parameter manipulation, demonstrating amplified unfairness beyond single-parameter manipulation. This dual manipulation strategy represents a fundamental threat to QUIC fairness, demonstrating how reduced loss response enables maintenance of higher windows during congestion, while increased growth

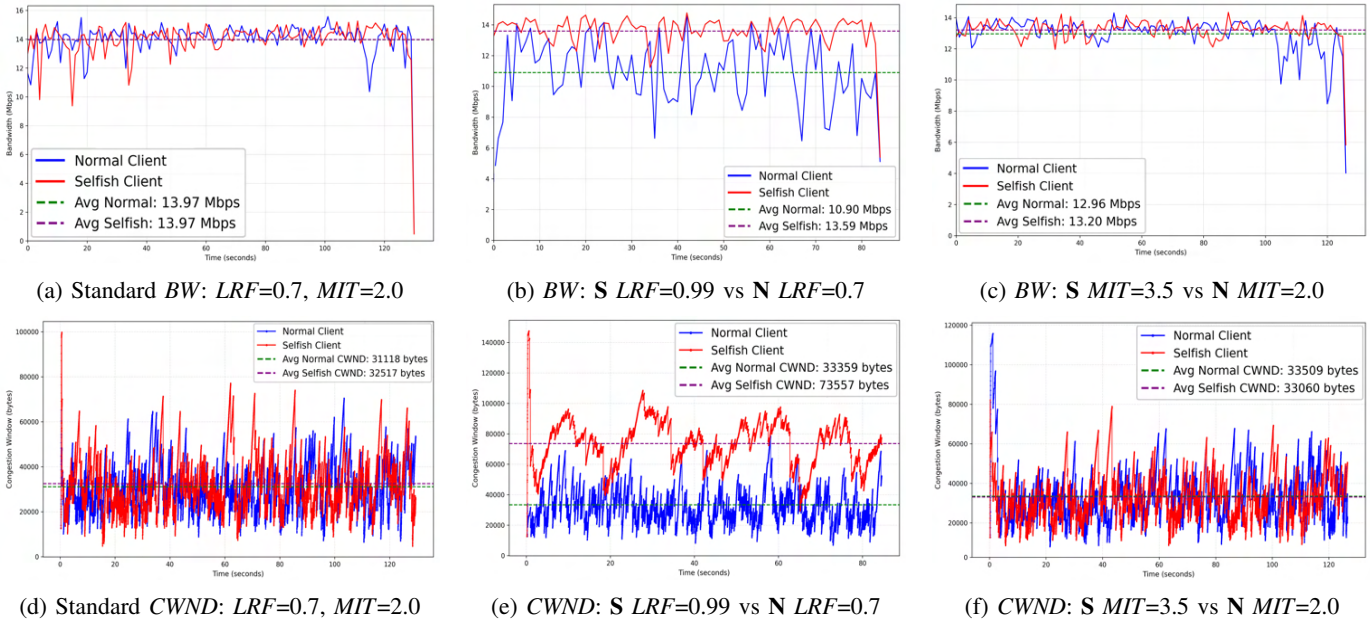
rate facilitates aggressive recovery, creating sustained dominance rather than brief advantage periods.

D. CUBIC Experimental Results and Analysis

1) Loss Reduction Factor (LRF) Manipulation: We evaluated *LRF* manipulation effects on CUBIC congestion control using the same established VM experimental environment with concurrent 100 MB file transfers from both clients. Our testing included one baseline scenario (both clients at *LRF*=0.7) and multiple manipulation configurations where the selfish client employed *LRF* values ranging from 0.7 to 0.99 while the normal client maintained standard *LRF*=0.7. We repeated each configuration 20 times, collecting individual PCAP and JSON files for both clients in every trial, then extracted *BW* measurements from PCAP files and *CWND* data from JSON files to create a CSV dataset containing 20 samples with *BW* measurements, *BW* ratios, average and maximum *CWND* values, and *CWND* ratios for each *LRF* configuration. Two statistical analyses were performed on the CSV dataset with 95% confidence intervals (CI) and t-distribution: *BW* allocation analysis examining normal versus selfish client performance and *BW* advantage analysis measuring relative unfairness. CUBIC congestion control demonstrates greater resilience to parameter manipulation compared to NewReno, still remains exploitable through strategic modifications. Under standard baseline conditions, both clients achieve equitable *BW* distribution (Figure 5a) with balanced *CWND* dynamics showing coordinated congestion responses (Figure 5d). However, *LRF* manipulation (*LRF*=0.99) creates measurable unfairness and significant *QoS* degradation for the normal client, manifesting as *BW* instability rather than absolute reduction, where the selfish client maintains stable *BW* while the normal client experiences severe oscillations and unpredictable performance drops (Figure 5b). The underlying mechanism shows the selfish client maintaining substantially larger average windows with stable performance, while the normal client exhibits dramatic *CWND* fluctuations causing unpredictable *BW* and potential service interruptions (Figure 5e). The *BW* allocation analysis reveals controlled increase in unfairness across *LRF* values, with the selfish client maintaining consistent performance while the normal client shows decreasing capacity (n=20, 95% CI) (Figure 6a). The *BW* advantage analysis demonstrates that the selfish client achieves linear growth from essentially zero to moderate *BW* advantages over the normal client at maximum *LRF* values (n=20, 95% CI) (Figure 6d). While CUBIC's polynomial growth provides better resistance than NewReno, the consistent linear progression demonstrates that *LRF* manipulation remains an effective exploitation vector, particularly damaging to normal client's *QoS* through sustained performance instability.

2) Maximum Idle Time (MIT) Manipulation: We evaluated *MIT* manipulation effects using the same experimental setup with one standard baseline scenario (both clients at *MIT*=2.0) and manipulation configurations where the selfish client employed *MIT* values ranging from 2.0 to 3.5 while the normal client maintained standard *MIT*=2.0. Each configuration was repeated 20 times, following identical data collection procedures to create a CSV dataset containing 20 samples with *BW* measurements, *BW* ratios, average and maximum *CWND* values, and *CWND* ratios for each *MIT* configuration. We applied two analytical methods to the CSV dataset using 95% confidence intervals (CI) and t-distribution: *BW* allocation analysis and *BW* advantage analysis. *MIT* manipulation (*MIT=3.5*) demonstrates minimal exploitation potential under standard CUBIC configurations. The selfish client achieves only marginal *BW* advantages with modest distribution shifts (Figure 5c), while *CWND* behavior reveals similar average window sizes with synchronized oscillatory patterns, demonstrating CUBIC's inherent resistance to *MIT* manipulation (Figure 5f). The *BW* allocation analysis shows lack of systematic exploitation reliability, with performance remaining relatively stable across different *MIT* values (n=20, 95% CI) (Figure 6b). The *BW* advantage analysis confirms minimal variation with overlapping confidence intervals (n=20, 95% CI) (Figure 6e). This demonstrates that *MIT* manipulation does not translate into statistically significant exploitation potential, as CUBIC's synchronized congestion control effectively neutralizes attempts to exploit the system through *MIT* parameter manipulation alone.

3) Combined Parameters (LRF+MIT) Manipulation: In this experiment, we investigated multi-parameter manipulation effects on CUBIC congestion control to evaluate combined exploitation impacts. We tested scenarios where the selfish client employed both *LRF*=0.99 and *MIT*=3.5 simultaneously while the normal client maintained standard parameters (*LRF*=0.7, *MIT*=2.0). Each combined manipulation configuration was repeated 20 times, following identical data collection procedures with *BW* measurements extracted from PCAP files and *CWND* data from JSON files. Multi-parameter manipulation in CUBIC demonstrates combined effects that amplify individual parameter impacts. Combined modifications achieve substantial *BW* allocation shifts, enabling the selfish client to capture around 59% of available *BW* resources compared to single-parameter modification alone (Figure 6c). The desynchronization creates persistent advantages rather than transient spikes, with the selfish client maintaining consistently high *BW* while the normal client suffers pronounced oscillations and frequent drops, contrasting sharply with the synchronized oscillation patterns observed in single-parameter manipulation cases. This severely impacts the normal client's *QoS*, creating unpredictable performance with dramatic *BW* fluctuations that would degrade user experience and application reliability. The underlying mechanism shows how the selfish configuration amplifies *CWND* sizes through complete desynchronization of window evolution, with window size ratios increasing dramatically compared to single-parameter modification (Figure 6f). The dual strategy operates through complementary mechanisms where reduced loss response maintains larger windows during congestion, while extended multiplicative increase threshold enables aggressive expansion during cubic growth. This vulnerability demonstrates a critical weakness in CUBIC's design within QUIC. While CUBIC provides superior resistance to



(a) Standard *BW*: *LRF*=0.7, *MIT*=2.0 (b) *BW*: **S** *LRF*=0.99 vs **N** *LRF*=0.7 (c) *BW*: **S** *MIT*=3.5 vs **N** *MIT*=2.0

(d) Standard *CWND*: *LRF*=0.7, *MIT*=2.0 (e) *CWND*: **S** *LRF*=0.99 vs **N** *LRF*=0.7 (f) *CWND*: **S** *MIT*=3.5 vs **N** *MIT*=2.0

Fig. 5: CUBIC parameter manipulation analysis showing *BW* and *CWND* behavior (**S**=Selfish, **N**=Normal)

single-parameter manipulation compared to NewReno through coordinated flow management, strategic multi-parameter exploitation systematically disrupts this coordination, creating sustained unfair advantages that fundamentally compromise the protocol's fairness guarantees and severely affect the *QoS* of normal client behavior.

E. Cross-Algorithm Performance Comparison

In these experiments, we examine competitive behavior and synchronization dynamics between CUBIC and NewReno algorithms operating simultaneously. We investigate inter-algorithm coordination when different CCA compete for shared resources using the same four-VM network topology (Figure 2) with NewReno client (192.168.56.103) and CUBIC client (192.168.56.102) under identical network constraints (15 Mbps *BW*, 15 ms delay, 0.3% loss) within the *Linux tc with netem* router VM.

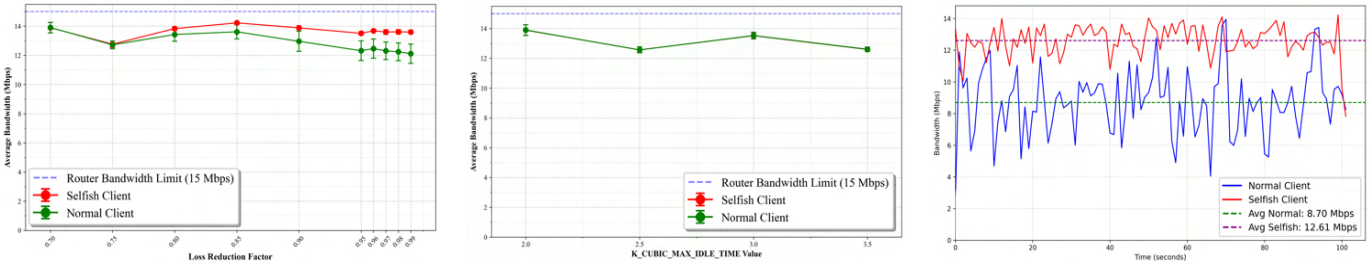
1) Normal (Standard) Operation Comparison: In this experiment, we evaluated baseline performance comparison between CUBIC and NewReno algorithms under standard configurations. We tested scenarios where one client used CUBIC congestion control while the other used NewReno, both maintaining default parameters. Each configuration was repeated 20 times using the same established experimental (VM) setup with the same data collection methodology of capturing PCAP files for *BW* measurements and JSON files for *CWND* analysis. In standard configurations, CUBIC achieves improved *BW* performance compared to NewReno due to CUBIC's more efficient polynomial growth function against NewReno's linear growth approach (Figure 7a). Both algorithms converge rapidly but exhibit asynchronous oscillations, indicating different congestion response characteristics that result in asynchronous behavior. The underlying mechanisms reveal NewReno showing aggressive growth with sharp reductions, while CUBIC maintains more moderate windows with smoother transitions (Figure 7b). The independent window reductions confirm the absence of inter-algorithm coordination, with each algorithm responding independently to congestion events, establishing clear baseline performance differentials between the two approaches.

2) Selfish Behavior (LRF=0.99): In this experiment, we evaluated the impact of aggressive parameter settings by configuring both NewReno and CUBIC clients with *LRF*=0.99 while maintaining all other parameters at default values. Each configuration was repeated 20 times using the same VM setup with the same data collection methodology, capturing PCAP files for *BW* measurements and JSON files for *CWND* data. Aggressive *LRF* values for both algorithms demonstrate that CUBIC's algorithmic advantages are neutralized when both protocols employ equally aggressive strategies (Figure 7c). This creates intense resource competition where neither protocol gains significant *BW* advantage, substantially reducing the performance gap compared to standard conditions. The corresponding *CWND* behavior shows both algorithms exhibit sustained growth patterns, eliminating traditional sawtooth oscillations, with both reaching substantially larger window sizes (Figure 7d). Despite CUBIC maintaining larger congestion windows, network bottleneck saturation prevents meaningful *BW* advantages, resulting in balanced competition where capacity limitations dominate over algorithmic differences.

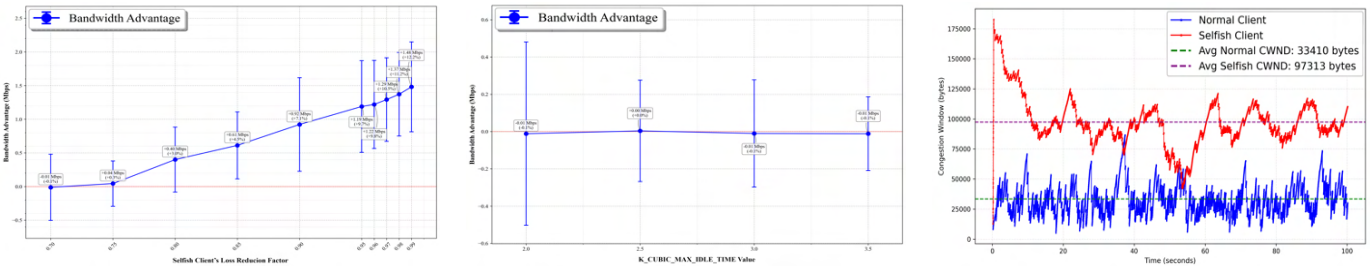
IV. Discussion

This section discusses a comprehensive vulnerability analysis, future defense mechanisms & evaluations, and limitations.

Vulnerability Analysis: Our empirical study reveals a major security vulnerability in QUIC's congestion control archi-

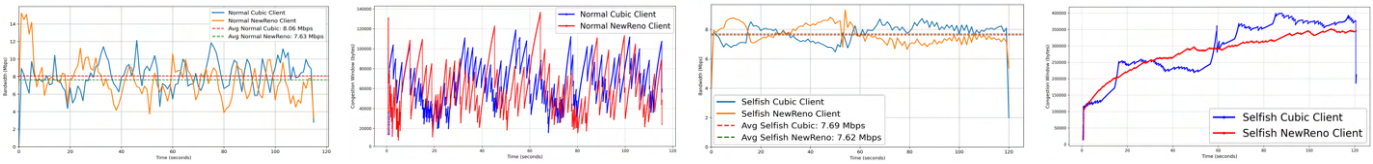


(a) *LRF* avg *BW* (**N** *LRF*=0.7) (b) *MIT* avg *BW* (**N** *MIT*=2.0) (c) **S** combined vs **N** standard



(d) **S** *BW* advantage (*LRF*) (e) **S** *BW* advantage (*MIT*) (f) **S** combined vs **N** standard

Fig. 6: CUBIC statistical validation and combined manipulation effects on *BW* and *CWND* (**S**=Selfish, **N**=Normal)



(a) **N** *BW* (b) **N** *CWND* (c) **S** *BW* (d) **S** *CWND*

Fig. 7: NewReno vs CUBIC performance comparison with normal and selfish configurations (**S**=Selfish, **N**=Normal)

tecture, enabling coordinated multi-parameter manipulation of CCAs (NewReno and CUBIC) that produces amplified effects and compromises protocol fairness. The most significant finding is the compounding effect when multiple parameters are manipulated simultaneously in both algorithms. While individual parameter manipulation creates measurable unfairness, combined manipulation reveals that QUIC's modular congestion control design becomes a critical liability. NewReno exhibits extreme vulnerability with severe *BW* monopolization, while CUBIC shows greater resilience but remains significantly exploitable with substantial *BW* monopolization by selfish clients, accompanied by severe *QoS* degradation for normal clients through *BW* instability and unpredictable performance drops. Furthermore, this vulnerability exposes a design assumption failure in QUIC's CCAs, which assume honest endpoint behavior under cooperative networking principles. When one endpoint manipulates parameters while the other remains cooperative, persistent unfairness emerges. However, when both clients use different algorithms, no clear winner emerges but overall network efficiency degrades, suggesting that widespread adoption of selfish behaviors could trigger a scenario where escalating client aggressiveness systematically degrades network performance. Importantly, the observed bandwidth measurement differences between NewReno and CUBIC reflect underlying internet connectivity variations rather than experimental inconsistencies. The same laptop was deployed across multiple physical locations with different internet connectivity using identical virtual topologies. Since the virtual topology remained fully controlled and isolated across all locations, these variations in absolute bandwidth measurements demonstrate the robustness and reproducibility of the CCPM phenomenon across different internet connectivity environments, thereby validating our findings under diverse network conditions.

Feasible Defense Mechanism and Extended Large-Scale Evaluation: To fight the CCPM attack, several potential approaches are possible. QUIC implementation libraries could enforce stricter parameter bounds to prevent manipulation, while server-side applications can implement connection-level monitoring to detect and restrict flows exhibiting monopolistic behavior patterns. These solutions can be implemented without requiring protocol modifications while providing effective defense capabilities. Our future research will focus on developing and evaluating prototype implementations of these detection and mitigation strategies across multiple QUIC implementations, including Google's *quiche*, Microsoft's *msquic*, and Meta's *mvfst*. Furthermore, we will extend vulnerability analysis to large-scale multi-client scenarios beyond two-client

configurations in various QUIC implementations and evaluate additional CCA including BBR [31].

Limitations: Our study has several limitations that might constrain the generalization of our findings. We focused exclusively on the *aioquic* implementation and only examined the NewReno and CUBIC algorithms, while other QUIC implementations and CCAs such as BBR remain unexplored. Our controlled *Linux VM* environment with *TC netem* router occasionally allowed *BW* to exceed the configured limits, potentially affecting the precision of measurement.

V. Conclusion

In this paper, we systematically investigated *CCPM* attack in QUIC, revealing critical vulnerabilities that enable selfish clients to monopolize network resources. Through comprehensive experimentation with *aioquic*, we demonstrated that NewReno exhibits a severe vulnerability to compound parameter manipulation enabling substantial *BW* monopolization, while CUBIC shows greater resilience but remains exploitable with significant resource capture and *QoS* degradation for normal clients. Our empirical analysis reveals that QUIC's user-space implementation is subject to a new attack vector where compound parameter modifications produce substantially greater unfairness than individual parameter changes. This *CCPM* attack pose significant risks to network fairness and internet infrastructure stability, potentially enabling systematic *BW* theft in shared environments. While limited to *aioquic* within controlled environments, our findings highlight an urgent security concern that requires immediate attention. Future work should validate this vulnerability across diverse implementations and develop practical defences.

Acknowledgment

This work has been partially supported by the French National Research Agency under the France 2030 label (Superviz ANR-22-PECY-0008). The views reflected herein do not necessarily reflect the opinion of the French government.

References

[1] J. e. a. Mücke, "Reacked quicer: Measuring the performance of instant acknowledgments in quic handshakes," in *Proceedings of the 2024 ACM IMC*, 2024, pp. 389–400.

[2] J. Iyengar and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport," Internet Engineering Task Force, RFC 9000, May 2021. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc9000

[3] A. Mishra and B. Leong, "Containing the cambrian explosion in quic congestion control," in *Proceedings of the 2023 ACM on Internet Measurement Conference*, 2023, pp. 526–539.

[4] Y. Joarder and C. Fung, "Exploring quic security and privacy: A comprehensive survey on quic security and privacy vulnerabilities, threats, attacks and future research directions," *IEEE TNSM*, 2024.

[5] Y. A. Joarder and C. J. Fung, "A survey on the security issues of quic," in *2022 6th Cyber Security in Networking Conference (CSNet)*. Rio de Janeiro, Brazil: IEEE, October 2022, pp. 1–8.

[6] e. a. Langley, Adam, "The quic transport protocol: Design and internet-scale deployment," in *Proceedings of the ACM SIGCOMM*, 2017, pp. 183–196.

[7] "Usage Statistics of Site Elements for Websites, June 2025." [Online]. Available: https://w3techs.com/technologies/overview/site_element

[8] J. Iyengar and I. Swett, "QUIC Loss Detection and Congestion Control," RFC 9002, May 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc9002

[9] Y. Han, M. Zuo, H. Yuan, Y. Zhong, Z. Yuan, and T. Bi, "A QoS-based fairness-aware BBR congestion control algorithm using QUIC," *Wireless Communications and Mobile Computing*, vol. 2022, pp. 1–16, 2022.

[10] A. Mishra, L. Rastogi, R. Joshi, and B. Leong, "Keeping an eye on congestion control in the wild with Nebby," in *ACM SIGCOMM 2024 Conference (ACM SIGCOMM '24)*. Sydney, NSW, Australia: ACM, August 2024, pp. 136–150.

[11] M. e. a. Zverev, "Robust quic: Integrating practical coding in a low latency transport protocol," *IEEE Access*, vol. 9, pp. 138 225–138 244, 2021.

[12] aioquic contributors, "aioquic: A python library for the quic network protocol," https://github.com/aiortc/aioquic, 2024.

[13] A. Akella, S. Seshan, R. Karp, and S. Shenker, "Selfish behavior and stability of the internet: A game-theoretic analysis of TCP," in *Proceedings of 2002 ACM SIGCOMM*, 2002, pp. 117–130.

[14] H. Zhang, D. Towsley, and W. Gong, "TCP connection game: A study on the selfish behavior of TCP users," in *Proceedings of the 13th IEEE International Conference on Network Protocols (ICNP'05)*. IEEE, 2005, pp. 1–10.

[15] P. Chen, N. Gu, D. Liu, and Q. Yu, "A game theory perspective on TCP congestion control evaluation," in *2022 International Conference on Networking and Network Applications (NaNA)*. IEEE, 2022, pp. 189–193.

[16] B. Teyssier, Y. Joarder, and C. Fung, "An empirical approach to evaluate the resilience of quic protocol against handshake flood attacks," in *CNSM*. IEEE, 2023, pp. 1–9.

[17] Y. Joarder and C. Fung, "Quicwand: A machine learning optimization-based hybrid defense approach against quic flooding attacks," in *2024 20th International Conference DRCN)*. IEEE, 2024, pp. 92–99.

[18] B. Teyssier, Y. Joarder, and C. Fung, "Quicshield: A rapid detection mechanism against quic-flooding attacks," in *2023 IEEE VCC*. IEEE, 2023, pp. 43–48.

[19] Y. Joarder and C. Fung, "Quicpro: Integrating deep reinforcement learning to defend against quic handshake flooding attacks," in *Proceedings of the 2024 Applied Networking Research Workshop*, 2024, pp. 94–96.

[20] W. e. a. Chen, "Pbq-enhanced quic: Quic with deep reinforcement learning congestion control mechanism," *Entropy*, vol. 25, no. 2, p. 294, 2023. [Online]. Available: https://www.mdpi.com/1099-4300/25/2/294

[21] S. Floyd, T. Henderson, and A. Gurtov, "The newreno modification to TCP's fast recovery algorithm," *RFC*, vol. 3782, pp. 1–25, 2004, rFC 3782. [Online]. Available: https://www.rfc-editor.org/info/rfc3782

[22] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[23] J. Gao and N. S. Rao, "Tcp aimd dynamics over internet connections," *IEEE communications letters*, vol. 9, no. 1, pp. 4–6, 2005.

[24] G. Luan and N. C. Beaulieu, "Accurate mathematical modeling and solution of tcp congestion window size distribution," *Computer Communications*, vol. 181, pp. 284–293, 2021.

[25] M. Letourneau, G. Doyen, R. Cogranne, and B. Mathieu, "A comprehensive characterization of threats targeting low-latency services: The case of l4s," *JNSM*, vol. 31, no. 1, p. 19, 2023.

[26] P. Kharat and M. Kulkarni, "Modified QUIC protocol with congestion control for improved network performance," *IET Communications*, vol. 15, no. 9, pp. 1210–1222, 2021.

[27] A. Mishra, S. Lim, and B. Leong, "Understanding speciation in QUIC congestion control," in *Proceedings of the 22nd ACM Internet Measurement Conference (IMC '22)*. ACM, 2022, pp. 560–566.

[28] M. e. a. Arashloo, "Enabling programmable transport protocols in {High-Speed}{NICs}," in *17th USENIX NSDI 20*, 2020, pp. 93–109.

[29] Z. e. a. Zhang, "Pbq-enhanced quic: Quic with deep reinforcement learning congestion control mechanism," *Entropy*, vol. 25, no. 2, p. 294, 2023. [Online]. Available: https://www.mdpi.com/1099-4300/25/2/294

[30] T. H. S. Floyd and A. Gurtov, "The newreno modification to tcp's fast recovery algorithm," RFC 6582, IETF, 2012. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc6582

[31] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," in *ACM Queue*, vol. 14, no. 5. ACM, 2016, pp. 20–53.

[32] R. G. Garroppo, S. Giordano, S. Lucetti, and E. Valori, "The wireless hierarchical token bucket: a channel aware scheduler for 802.11 networks," in *WoWMoM*. IEEE, 2005, pp. 231–239.

[33] S. Hemminger, "Network emulation with netem," in *Linux conf au*, vol. 5, 2005, pp. 18–23.