POLITECNICO DI TORINO

Master Degree in Cybersecurity

Master Degree Thesis

Packet-Level eBPF Traffic Obfuscation for QUIC to Mitigate Website Fingerprinting

Supervisors

Prof. SACCO ALESSIO

Dr. RINAUDI FEDERICO

Prof. MARCHETTO GUIDO

Candidate

WENHUI LI

March 2026

Abstract

Website Fingerprinting (WF) shows that encrypting content is not enough to protect browsing privacy. A passive observer can still infer visited sites by analyzing traffic patterns such as packet sizes, packet timing, direction, and burst structure. Recent deep learning methods make this inference easier and more robust, as they can learn directly from sequences of sizes and inter arrival times even under noisy network conditions. With the widespread use of QUIC and HTTP/3, these patterns remain observable at the network level and therefore continue to be a relevant side channel.

This thesis presents a client-only countermeasure for QUIC and HTTP/3 traffic implemented with eBPF. The system operates at packet level and requires no changes to applications or to the QUIC stack. It modifies downlink traffic through selective packet drop and uplink traffic through dummy packet injection, with the goal of making traffic traces less distinguishable across sites. The drop and injection policies are configurable to control the trade-off between protection and overhead, and can adapt automatically to traffic conditions by reducing their intensity under high load.

Experimental evaluation shows that the defense alters the statistical structure of traffic traces in ways that are consistent with reduced fingerprintability. Distributional analysis of defended traces shows increased overlap across sites, and unsupervised clustering on defended traffic yields lower accuracy compared to undefended traffic. The added delay is negligible relative to typical page load times.

This work extends packet-level eBPF-based traffic obfuscation, previously applied to TCP (Cactus), to QUIC and HTTP/3, where existing client-side defenses (QCSD) operate at the application layer. The result is a packet-level, application-transparent defense for QUIC traffic that requires no changes to applications, protocol stacks, or server infrastructure.

Table of Contents

List of Figures

Acronyms

QUIC
: Quick UDP Internet Connections

HTTP/3
: Hypertext Transfer Protocol version 3

TCP
: Transmission Control Protocol

UDP
: User Datagram Protocol

TLS
: Transport Layer Security

HTTPS
: Hypertext Transfer Protocol Secure

VPN
: Virtual Private Network

MPLS
: Multiprotocol Label Switching

BPF
: Berkeley Packet Filter

eBPF
: extended Berkeley Packet Filter

XDP
: eXpress Data Path

TC
: Traffic Control

NIC
: Network Interface Card

CFG
: Control Flow Graph

DPDK
: Data Plane Development Kit

WF
: Website Fingerprinting

IAT
: Inter-Arrival Time

CNN
: Convolutional Neural Network

SVM
: Support Vector Machine

PLT
: Page Load Time

RTO
: Retransmission Timeout

HOL
: Head-of-Line

ACK
: Acknowledgment

CWND

Congestion Window

CDN

Content Delivery Network

RFC

Request for Comments

CDP

Chrome DevTools Protocol

PCAP

Packet Capture

Chapter 1

Introduction

The internet transport layer is experiencing a major transition from legacy TCP to QUIC and HTTP/3. Led by many large internet companies and widespread standardization, QUIC adoption has grown quickly, offering significant improvements in latency, multiplexing, and connection migration. Unlike TCP, QUIC encrypts its transport headers, including packet numbers and control flags, alongside the application payload using TLS 1.3. However, this robust encryption does not automatically guarantee browsing privacy. Website Fingerprinting (WF) attacks have shown that passive eavesdroppers can still identify the specific web pages a user visits by analyzing structural metadata. This includes packet size distributions, inter-arrival timings, directionality, and burst sequences. Recent research reveals that despite its encrypted headers, QUIC traffic remains highly vulnerable to these sophisticated machine learning classifiers.

Mitigating these attacks introduces a difficult challenge between privacy and performance specific to the QUIC protocol. Some strong protection methods, such as user-space proxies or adding more data to hide information, can slow down the low latency and high throughput advantages that make QUIC desirable in the first place. Conversely, some lightweight protection methods relying on simple padding are easily defeated by modern deep learning classifiers. Furthermore, existing mitigation strategies are mostly incompatible with modern web architectures. Kernel defenses designed for TCP, such as Cactus, rely on tracking cleartext transport headers and sequence numbers. This approach is made completely ineffective by the full encryption of QUIC. On the other hand, application layer defenses like QCSD require invasive modifications to the underlying QUIC stack and the applications themselves. As a result, there is currently a critical gap in the literature. No packet-level, client-only, and application-transparent WF defense currently exists for QUIC.

Where we place a defense in the network system is very important for its reliability and practicality. Because a normal user cannot control the configuration

of remote web servers, and modifying every single application individually is highly impractical, the most viable approach is to intercept traffic at the operating system level. Application layer defenses handle data before it passes through the operating system. Because of this, they cannot guarantee that the padding or timing they add will stay the same when the data finally becomes real packets [1]. This makes it harder to stop an attacker who is watching the traffic. In contrast, a defense operating directly at the kernel-level within the data plane intercepts and manipulates the exact packet-level traffic that a passive eavesdropper would capture. Operating at this bottom layer ensures that the injected obfuscation accurately translates to the network wire, making the defense more reliable and better at stopping traffic analysis.

To address these challenges, we present a novel eBPF defense operating natively at the Traffic Control (TC) layer of the Linux kernel, directly in the data plane. The core idea of our approach is based on the difference between the ingress and egress channels. We call this a 'Combined Asymmetric strategy'. On the downlink, we apply a selective packet drop mechanism to naturally trigger the congestion control and loss recovery of QUIC, which heavily mutates the temporal burst structures. On the uplink, we inject synthetic dummy packets asynchronously to mask client request patterns. Because it operates strictly within the data plane on the actual outgoing and incoming datagrams rather than application level abstractions, the system requires absolutely no modifications to the user applications, the client QUIC stack, or the remote server infrastructure.

Our practical experiments using automated traces demonstrate the effectiveness of this system. Under natural conditions, a K-Means clustering adversary achieved a baseline WF accuracy of 72.4%. Our Combined Asymmetric strategy successfully reduced this accuracy to 31.2%, while incurring a highly manageable latency penalty of approximately 400 milliseconds and a network bandwidth overhead of only 21%. Visual analyses using t-SNE dimensionality reduction further supported these findings, displaying a complete dissolution of traffic clusters. This establishes an optimal balance between security and usability, sharply contrasting with isolated strategies. A Drop Only approach provides strong privacy but suffers from unacceptable latency degradation, whereas a Dummy Only approach yields minimal overhead but provides insufficient protection against structural traffic analysis.

Our main contribution is a client-only, application-transparent, and packet-level defense for QUIC using eBPF at the TC layer. This system combines selective packet dropping on the downlink with dummy packet injection on the uplink to achieve robust traffic obfuscation with low overhead.

The remainder of this thesis is structured as follows. Chapter 2 covers the technical background and formulates the problem statement. Chapter 3 reviews the related work in Website Fingerprinting and corresponding defenses. Chapter 4 describes the detailed system design and implementation of our eBPF mitigation

architecture. Chapter 5 presents the experimental methodology and a comprehensive evaluation of the results. Finally, Chapter 6 draws conclusions and outlines potential avenues for future work.

Chapter 2

Background

The internet transport layer is undergoing a major change. For decades, the global internet relied almost entirely on the Transmission Control Protocol (TCP) to ensure reliable data delivery. However, we are now witnessing a steady migration toward the UDP based QUIC protocol. First pushed by companies like Google and now officially standardized by the Internet Engineering Task Force (IETF), QUIC has been established as the mandatory foundation for HTTP/3. Because modern web browsers and major tech companies have rapidly adopted this standard, QUIC is quickly dominating global internet traffic [2, 3].

The design philosophy of QUIC directly addresses the fundamental architectural limitations of TCP. First, TCP requires a time consuming three ways handshake to establish a connection, followed by a separate cryptographic handshake for security. QUIC solves this latency problem by deeply integrating the cryptographic and transport handshakes together. This allows clients to achieve 0-RTT (Zero Round Trip Time) latency, meaning a returning user can start sending encrypted application data immediately in their very first network packet [4, 5].

Second, QUIC eliminates the common Head-of-Line (HOL) blocking problem. In a traditional TCP connection, all data travels in a single, ordered sequence. If a single packet is lost during transmission, the entire connection must halt and wait for that specific packet to be retransmitted, delaying all subsequent data. QUIC, however, utilizes a multiplexed stream architecture. It divides the web page into multiple independent streams. If a packet belonging to an image stream is lost, only that specific image is delayed, while the streams carrying the HTML text and CSS files continue to render seamlessly [4, 6]. Furthermore, QUIC introduces connection migration, allowing mobile users to switch between WiFi and cellular networks without dropping their active downloads.

However, this pursuit of performance and structural efficiency raises a fundamental question: has our online privacy been equally strengthened? Because QUIC integrates TLS 1.3 by default, it encrypts more transport metadata than TCP,

including packet numbers and control flags. Many assume this deep encryption guarantees absolute privacy. Unfortunately, the encrypted payload only hides the literal content of the communication (the "what"), but it fails to hide the behavioral shape of the communication (the "how").

Because of this vulnerability, QUIC remains highly susceptible to sophisticated traffic analysis techniques, specifically Website Fingerprinting (WF) [7, 8]. WF is a serious privacy attack where a passive network adversary, such as an Internet Service Provider (ISP) or a malicious eavesdropper on a public WiFi network can accurately identify the exact websites a user is visiting, even if the user employs HTTPS, a Virtual Private Network (VPN), or the Tor anonymity network.

The mechanism behind Website Fingerprinting is effective. Instead of attempting to break the unbreakable TLS encryption, the attacker analyzes the statistical metadata patterns of the encrypted traffic stream. Every website is built differently. A webpage featuring massive, high resolution images will naturally generate a heavy burst of large packets traveling from the server to the client. Conversely, a simple login page will generate a few small packets with distinct timing gaps. By passively recording metadata such as packet sizes, traffic direction, Inter-Arrival Time (IATs), and burst sequences, the attacker compiles a unique digital fingerprint for every website [7].

Recent research confirms that state of the art WF attacks, particularly those utilizing deep learning algorithms like Convolutional Neural Networks (CNNs), maintain alarmingly high identification accuracy against QUIC traffic [9]. Today, we entrust an increasing amount of global internet traffic to a high-performance protocol, yet this protocol still leaks critical privacy information through side channels. Protecting the network infrastructure of this protocol without compromising its performance advantages is the core focus of this paper.

2.1 The Challenge of Defending QUIC

Stopping Website Fingerprinting (WF) attacks on QUIC is difficult because there is a strong trade-off between privacy and speed. Since QUIC is designed to be very fast, any new protection method must be carefully tested against this high-speed standard. If the defense slows down the connection too much, it would destroy the main reason why people use QUIC in the first place.

The first main challenge is related to encryption. Unlike older protocols where TCP headers were sent in cleartext, QUIC uses TLS 1.3 to encrypt almost everything by default. This includes not only the data but also the transport information, such as packet numbers. Because of this, traditional network tools cannot see what is happening inside the connection. They can no longer use sequence numbers to decide when to add padding. This makes older traffic protection methods ineffective

for QUIC.

This means, existing defense strategies present a severe dilemma when applied to QUIC. Heavyweight solutions, such as user-space routing proxies or Tor like anonymity networks, introduce massive context switching and memory copying overhead. Because QUIC transmits data over UDP, moving thousands of UDP datagrams between the operating system kernel and a user-space proxy creates a severe processing bottleneck. This routing latency directly negates the core benefits of QUIC, specifically its rapid connection establishment and zero-RTT handshakes [10]. Conversely, lightweight solutions, such as simple constant rate packet padding or basic dummy packet injection, can often be easily defeated by advanced deep learning classifiers and impose unacceptable bandwidth overheads on the network [11].

An alternative and highly promising approach is traffic obfuscation through intentional packet dropping. By deliberately discarding a small %age of incoming packets, this method triggers the inherent loss recovery and retransmission mechanisms of the QUIC protocol (as illustrated in Figure 2.1). This organic recovery process dynamically alters the total packet counts and distorts the Inter-Arrival Time (IATs) of the traffic bursts, generating enough statistical noise to confuse WF classifiers.

This packet loss strategy is possible because of how QUIC is built. In older TCP connections, losing even one packet can cause Head-of-Line (HoL) blocking. This stops all other data from moving and makes the webpage freeze. However, QUIC uses independent streams. This means even if some packets for images are dropped on purpose, the streams for HTML and CSS can still work. Because of this, we can use packet loss to change the timing of traffic without breaking the website for the user.

The main problem is how to actually do this. Losing packets in both directions can make the network very slow. For example, if packets are lost when the client sends data (uplink), it can break the handshake or HTTP requests. This might leave the user looking at a blank screen. On the other hand, too much packet loss from the server (downlink) makes the server reduce its sending speed a lot. Therefore, creating a defense that changes traffic patterns but keeps QUIC fast is still a big challenge. This thesis aims to solve this problem.

Modern network architecture increasingly favors Data Plane Programmability. This approach separates the control plane from the data plane, allowing developers to dynamically inject custom packet processing logic directly into the network path without altering the underlying hardware or recompiling the operating system kernel.

In 1993, McCanne and Jacobson introduced the BSD Packet Filter (BPF) [12]. They designed this solution to allow fast and effective network monitoring in Unix systems. The main purpose of BPF was to filter specific packets and decide which

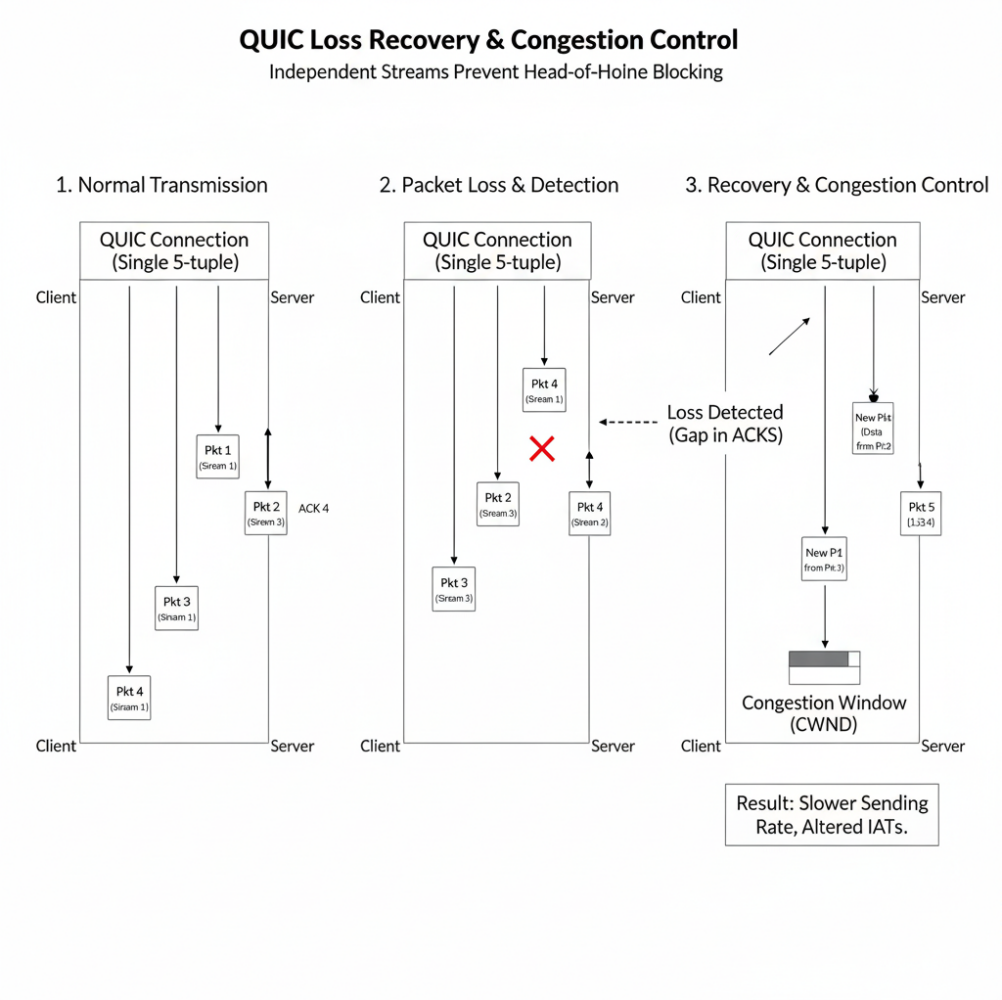

Figure 2.1: Conceptual diagram of QUIC's stream based reliability and loss recovery mechanism. When a packet is lost, QUIC re-bundles data into a new packet and retransmits, significantly altering the original Inter-Arrival Time (IATs) and traffic sequences.

portions should be sent to monitoring applications.

Before BPF, older applications like CPFS were used for similar purposes. BPF kept some core ideas from CPFS, such as filtering packets as an array of bytes using a pseudo machine language. However, BPF significantly improved performance by introducing a new register based architecture to replace the old stack based one. It also used a new control flow graph (CFG) representation to minimize redundant computations. In 1997, BPF was officially introduced into the Linux kernel starting with version 2.1.75.

Starting with kernel version 3.18 in 2014, BPF evolved into extended BPF (eBPF). Today, modern eBPF operates as a secure, kernel-integrated virtual machine. It allows sandboxed programs to run within the Linux kernel without modifying the source code or loading unstable kernel modules [13]. To achieve this safely and efficiently, the architecture relies on three core components: eBPF Maps, the Verifier, and the Just-In-Time (JIT) Compiler.

eBPF Maps are highly efficient key value data structures residing in the kernel memory. They can be accessed by both the eBPF programs running in the kernel and the control applications residing in the user-space. This feature enables seamless, real time information sharing and dynamic policy updates without restarting the network connection.

The eBPF verifier is a critical security component that ensures the absolute safety of eBPF programs. It operates by checking the code to prevent infinite loops, validating the Control Flow Graph (CFG), and simulating the execution of each instruction to detect any unsafe memory access before the program is allowed to run [14].

Once a program successfully passes the verifier, it is handed over to the Just-In-Time (JIT) compiler. As highlighted in the official eBPF documentation [15], the JIT compiler translates the generic eBPF bytecode into native machine-specific instruction sets. This step is crucial because it allows sandboxed eBPF programs to run at native execution speeds, matching the high performance of standard kernel modules.

Furthermore, to safely interact with the underlying operating system, eBPF programs are not allowed to call arbitrary kernel functions. Instead, they must utilize a restricted and stable set of Kernel Helper APIs provided by the ecosystem [15]. These helpers facilitate complex tasks, such as generating random numbers or redirecting packets, ensuring both robust security and wire-speed packet processing.

Because of these powerful features, eBPF is widely used today for performance tracing, improving network visibility, and preventing malicious activities. To support these applications, developers can use Software Development Kits (SDKs) like libbpf [16] and aya [17] to write eBPF code in high-level languages such as C, Go, or Rust. Building on this, large-scale projects like Cilium [18] help manage eBPF programs efficiently in complex cloud environments.

As illustrated in Figure 2.2, an eBPF program can be attached to various points in the network stack. These connection points are called hooks, and each hook provides specific capabilities.

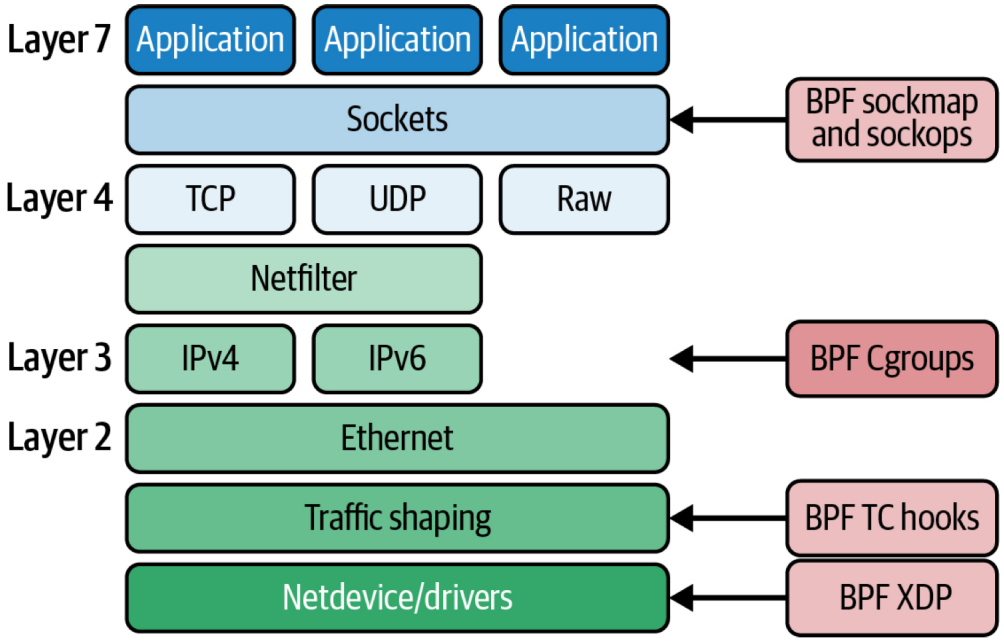

Figure 2.2: BPF program types hooked into various points in the network stack. Image extracted from [19].

For network packet processing, eBPF provides two primary hook points: eXpress Data Path (XDP) and Traffic Control (TC).

A study by Høiland Jørgensen et al. proposes XDP as an innovative solution for programmable packet processing [20]. XDP executes programs directly inside the Network Interface Card (NIC) drivers. It processes packets as soon as they arrive from the hardware, before the main kernel stack touches them. This allows XDP to achieve extreme performance comparable to kernel bypass systems like DPDK, while remaining much less complex. According to the authors, XDP is highly effective for routing and DDoS mitigation. Although it is slightly slower than DPDK on a single core, it can process up to 24 million packets per second and scales CPU utilization much better than DPDK. Furthermore, studies by Feng Wang et al. show that offloading XDP programs onto SmartNIC hardware can improve forwarding rates by 30 times compared to standard software mode [21].

While XDP offers extreme performance for early packet dropping, it only provides access to raw packet data. In contrast, the Traffic Control (TC) hook operates slightly later in the networking subsystem. The TC hook grants access to the socket buffer structure, which contains rich contextual metadata about the traffic.

Characteristic	XDP (eXpress Data Path)	TC (Traffic Control)
Hook Point	Network Driver (Earliest)	TC Subsystem (Later)
Data Context	Raw packet data	Rich socket metadata
Primary Goal	Extreme Performance	Flexibility and Traffic Shaping
Use Cases	Load Balancer, DDoS Firewall	Bandwidth Control, Complex Obfuscation

Table 2.1: Comparison of XDP and TC Network Hook Points.

Because Website Fingerprinting defenses require careful manipulation of traffic timing and the ability to inject cloned packets asynchronously, the rich flexibility of the TC layer makes it the optimal choice for the architecture proposed in this thesis.

Chapter 3

Related Work

3.1 Control Plane and Data Plane Separation

The control plane uses algorithm based routing protocols to determine the best path for information to take between network devices. One common example of this practice is Multiprotocol Label Switching (MPLS). In this system, the control plane determines the optimal route for data to travel through a network. It then assigns a unique label to the data before sending it to the data plane [22].

Both the control plane and the data plane are conceptually located at the network layer. This is the part of the computer network architecture that enables different devices to communicate over a wide area. The data plane, which is also known as the forwarding plane, is specifically responsible for moving data packets. These packets are small units of information collected for network transmission. The data plane controls the real time movement of packets across the network based on the routing rules and protocols it receives from the control plane [22].

Furthermore, the data plane constantly checks the routing table built and maintained by the control plane. This table provides guidance on how data should be directed. Based on these rules, the data plane forwards packets to the correct destination. It can also perform specific processing tasks, such as updating a packet header or classifying information, to meet strict security requirements [22].

A fundamental concept in modern programmable networking is the absolute separation of the control plane and the data plane [23]. The control plane is responsible for high level decision making and defining management policies. Meanwhile, the data plane focuses entirely on the efficient forwarding and manipulation of network packets [23]. In our proposed system, this separated architecture is implemented using eBPF. This ensures that complex strategy calculations in the user-space do not interfere with the wire speed performance of packet processing inside the kernel. The user-space application acts as the control plane to define obfuscation levels,

while eBPF programs at the Traffic Control (TC) layer serve as a high speed data plane [23].

3.2 Website Fingerprinting Attacks

Website Fingerprinting is a powerful type of traffic analysis attack that targets encrypted network traffic. Its core threat lies in the fact that even if a user browses the web using HTTPS, a Virtual Private Network, or the Tor anonymity network, attackers can still infer which website is being visited. They achieve this by simply observing the external characteristics of the communication stream without ever breaking the underlying cryptographic algorithms.

To fully understand this threat, it is important to define the position of the attacker. In a standard threat model, the adversary is assumed to be a passive eavesdropper located anywhere between the victim and the first encrypted gateway. This could be a local network administrator, a malicious WiFi hotspot operator, an Internet Service Provider, or even a compromised router on the public internet. The attacker does not alter the traffic or inject malware; they merely record the encrypted data packets flowing through their observation point.

In any network, although the actual payload content of the data packets is securely encrypted, several external metadata characteristics remain fully exposed to these passive observers. Firstly, the direction of the data packet is visible, meaning the attacker knows whether a packet is a request sent by the client or a response received from the server. Secondly, the size of the data packet is exposed. Because the size of images, scripts, and HTML files varies greatly across different web pages, packet sizes leak critical structural information. Thirdly, the time interval between data packets reveals latency patterns. When a browser downloads an initial HTML file, it parses the code and subsequently requests additional resources like fonts and scripts. This browser rendering engine behavior causes distinct timing gaps. Finally, the total number of data packets and their burst sequences provide a macro level view of the webpage weight. When combined, these specific characteristics form a unique and identifiable traffic fingerprint for every single website.

In the academic literature, the effectiveness of these attacks is typically evaluated in two distinct scenarios: the closed world and the open world. The closed world is a simplified experimental scenario. In this model, the attacker assumes that the user will only visit a website from a small, predefined list of monitored targets. The classification algorithm only needs to decide which website from the list was visited. While this yields very high accuracy, it is not realistic. Therefore, modern defenses must consider the open world scenario. In the open world, the user can visit millions of unmonitored background websites. The attacker must first determine if

the captured traffic belongs to a monitored target at all, and only then attempt to classify it. This introduces massive background noise and makes the attack significantly more difficult, perfectly representing real world internet browsing.

To better understand how this passive attack works, Figure 3.1 illustrates a standard Website Fingerprinting threat model.

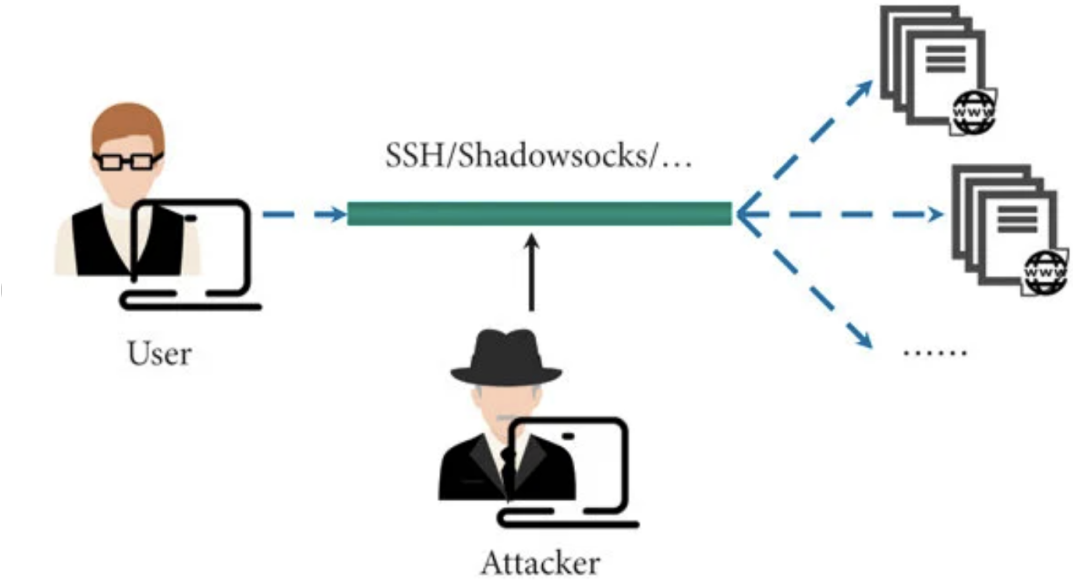

Figure 3.1: A standard Website Fingerprinting threat model. The passive attacker intercepts the encrypted traffic between the client and the external network. (Image adapted from [24])

As shown in the figure, the user communicates with the web server through a secure encrypted tunnel. Because of this strong encryption, the attacker cannot see the actual HTML code, passwords, or images inside the tunnel. However, the attacker can still clearly record the external flow of the network traffic. By feeding these recorded packet sizes, directions, and timing features into a machine learning classifier, the attacker can successfully guess which target website the user is visiting. This visual model highlights a crucial point: simply encrypting the data payload is not enough to protect user privacy against modern traffic analysis.

The evolution of these Website Fingerprinting attacks has been rapid over the last decade. Early research primarily utilized classical machine learning techniques.

The evolution of these Website Fingerprinting attacks has been rapid over the last decade. Early research primarily utilized classical machine learning techniques. These traditional methods relied heavily on manual feature engineering, where

human experts had to manually define mathematical rules to extract packet statistics. For example, Hayes et al. proposed the k Fingerprinting attack, which used Random Forest classifiers to achieve high accuracy by analyzing manual packet size distributions and ordering [25]. Similarly, the CUMUL attack utilized Support Vector Machines combined with cumulative packet lengths to successfully identify websites [26].

Other recent studies have focused on improving classification accuracy in real-world network environments [27, 28]. For example, researchers have explored new ways to handle background noise and extract better traffic features [29]. Furthermore, new machine learning methods have been proposed to make the attacks more stable across different web browsers and HTTP versions [30, 31].

However, recent advancements in the field are completely dominated by Deep Learning approaches [23]. Modern models, such as the Deep Fingerprinting model proposed by Sirinam et al., leverage Convolutional Neural Networks to automatically extract complex hidden patterns directly from raw traffic traces [32]. By treating the sequence of network packets almost like a one dimensional image, these deep learning models can discover extremely subtle correlations that human experts cannot manually define. More importantly, these models do not require manual feature engineering and have proven highly resilient against traditional network noise. This massive leap in offensive attack capabilities underscores the critical and urgent need for more robust, kernel based traffic obfuscation systems like the one proposed in this thesis.

Recent literature has expanded on this by introducing new deep learning classifiers. Some studies focus on understanding the context of the network flow to improve the attack accuracy [33, 34]. Other researchers have developed models that can analyze traffic behavior graphs or handle encrypted proxy traffic [35, 36]. These new methods show that attackers can easily adapt their models to different network setups [37].

3.3 Evolution of Traffic Obfuscation Defenses

To protect user privacy against the powerful classifiers mentioned above, researchers have developed various Website Fingerprinting defenses. Overall, all of these defense mechanisms can be divided into two main categories: regularization and randomization [23].

Regularization approaches aim to completely erase the traffic patterns that attackers exploit. They achieve this by forcing the network traffic of all websites into strict and predetermined shapes. For instance, constant rate padding methods like BuFLO and Tamaraw enforce a strict transmission schedule [38]. They send packets at fixed time intervals and pad them to a fixed size. If the real application

has no data to send, the system transmits dummy packets to maintain the constant rate. Because all websites are forced to look exactly the same, regularization is usually very effective at stopping attacks. However, this strict control is highly time consuming. It requires holding packets in memory to delay them, which imposes massive bandwidth overheads and severe latency penalties, making it impractical for regular daily usage.

On the other hand, randomization approaches aim to add noise to the existing traffic patterns rather than completely erasing them. These defenses introduce unpredictable elements, such as injecting synthetic dummy packets, splitting packets through fragmentation, or altering timing behaviors. This generated noise successfully masks the real traffic fingerprint. A prominent example is the WTF PAD system, which uses adaptive padding to hide traffic bursts without heavily delaying real packets [39]. Because randomization does not force a strict schedule, it adds much less delay. If the goal is to protect traffic with a low performance overhead, randomization is the preferred method. For example, traditional kernel defenses like Cactus use randomization to introduce dummy packets and manipulate fragmentation with random probabilities [23].

In this thesis, we explicitly adopt the randomization approach for our defense mechanism. We chose randomization for several important technical reasons. The most critical reason is the architectural limitation of our chosen technology. Our system operates entirely within the data plane using eBPF at the Traffic Control layer. As mentioned previously, regularization requires introducing specific delays between packets and holding them in memory buffers. However, the internal eBPF verifier strictly forbids delaying or buffering packets in the data plane. An eBPF program must process each packet immediately. Because we cannot delay packets, regularization is impossible to implement in a pure eBPF data plane. Therefore, using randomization techniques, such as injecting synthetic dummy packets and selectively dropping packets on the fly, is the only viable path to build an eBPF defense.

To counter these new attacks, several advanced defense strategies have been proposed recently. Some researchers try to anonymize the traffic clusters in real time [40, 41]. Others use graph neural networks or side-channel feature techniques to hide the user behaviors [42, 43]. All of these new studies share the same goal: finding a better balance between protecting user privacy and maintaining network speed [44].

3.4 The Shift to the QUIC Protocol

The critical limitations of traditional kernel defenses become particularly obvious with the widespread adoption of the Quick UDP Internet Connections (QUIC)

protocol. Originally proposed by Google to accelerate web traffic and recently standardized by the Internet Engineering Task Force (IETF), QUIC has been officially established as the underlying transport foundation for HTTP/3. It fundamentally redesigns the traditional internet stack to overcome the intrinsic performance and security limitations of the legacy TCP and TLS pairing.

Unlike the legacy stack, where TCP handles reliable data transport in cleartext and a completely separate TLS layer sits on top to handle encryption, QUIC runs over the connectionless UDP protocol and deeply integrates TLS 1.3 directly into its own transport layer. This unified architecture provides a massive performance boost. In a traditional TCP/TLS setup, a client must wait for multiple round trips (the TCP three-way handshake followed by the TLS cryptographic handshake) before sending any real web data. By merging these steps, QUIC significantly reduces connection establishment latency. For returning users who have previously communicated with the server, QUIC can often achieve zero-RTT (Zero Round Trip Time) handshakes, meaning the encrypted HTTP requests are sent immediately in the very first packet [45].

Furthermore, QUIC completely eliminates the common head-of-line (HoL) blocking problem inherent to TCP [46]. TCP enforces a strict, in-order delivery mechanism across a single logical pipe. If a single packet is lost in transit, the entire TCP connection must stall and wait for the retransmission of that specific packet, severely delaying all other unrelated data behind it. QUIC solves this by introducing stream multiplexing. A single QUIC connection can carry multiple independent data streams simultaneously. If a packet belonging to an image stream is dropped, only that specific image is delayed, while the streams carrying the HTML text and CSS continue to be processed by the browser seamlessly.

More importantly for privacy research, this deep integration fundamentally changes the encryption boundaries. While QUIC enhances user privacy through deep header encryption and stream multiplexing, recent research confirms that it remains highly vulnerable to Website Fingerprinting attacks. In the TCP era, transport headers and control flags, such as Sequence Numbers (SEQ), Acknowledgments (ACK), SYN, and FIN were transmitted in pure cleartext. Because of this, traditional kernel defenses like Cactus could easily monitor the progression of the web page load and intelligently decide when to inject padding [23].

However, QUIC encrypts almost all control information. This means modern kernel tools cannot see these signals anymore. Since they cannot access the cleartext information, the old methods used by Cactus no longer work. Therefore, traditional kernel-level defenses are not effective for QUIC.

Finally, QUIC possesses its own highly advanced, built in congestion control and loss recovery mechanisms. As defined in RFC 9002, the protocol involves complex, dynamic state transitions between various phases such as Slow Start, Recovery, and Congestion Avoidance [47]. When the network experiences packet loss, the

protocol automatically enters a recovery state, This changes its transmission speed and bursting behavior.

This protocol shift creates an urgent need for a new generation of kernel level and application-transparent defenses that do not rely on reading cleartext headers. Instead of attempting to parse unreadable encrypted streams, our research takes a different approach. We exploit the encrypted loss recovery and congestion control mechanisms of QUIC using eBPF. By dropping packets on the downlink, we force the QUIC protocol into its recovery phase, reshaping the traffic patterns to provide robust privacy without modifying the end user applications or breaking the encryption standards [23, 47].

Chapter 4

Solution and System Design

4.1 Challenges and Design Choices

Designing an effective and lightweight Website Fingerprinting defense at the packet-level presents multiple challenges, particularly when operating within the modern web stack.

Defending against traffic analysis is complex because adversaries do not rely on reading the contents of the packets. Instead, they exploit metadata, specifically packet size distributions, burst sequences, and directionality to identify unique patterns. Altering these structural patterns typically requires injecting large amounts of padding or delaying packets, both of which introduce bandwidth overhead and latency, degrading the user experience.

Protecting the QUIC protocol is more difficult than defending legacy protocols like TCP. In the past, security tools such as Cactus could monitor network traffic because TCP headers were sent in plain text. This allowed defenders to follow the progress of a connection by looking at sequence numbers and various flags. However, QUIC changes this situation because it uses TLS 1.3 encryption by default.

Unlike legacy protocols, QUIC encrypts almost every part of the packet. This includes the data being sent and even the technical information in the header, such as packet numbers. Furthermore, because QUIC multiplexes multiple logical streams into a single connection, outside observers only see a unified flow of encrypted UDP datagrams sharing the same Connection ID (CID). Because of this strict level of privacy, standard network equipment cannot see what is happening inside the connection. Security systems must now manage raw UDP data without knowing the specific purpose of each packet. For example, a defender cannot distinguish between a simple acknowledgment and actual website content. This lack of visibility makes it very hard to apply traditional stateful defense methods to modern web traffic.

Running security tools directly in the Linux kernel with eBPF provides high processing speed, but it also faces many strict rules. These rules are enforced by a component called the verifier. The verifier ensures that any code added to the kernel does not cause the computer to crash or stop working. Because of these safety requirements, developers cannot use the same techniques in the kernel that are used in regular user-space software.

One major problem is that eBPF cannot easily delay or store data packets. Many traditional regularization defense methods rely on this buffering ability. For instance, Tamaraw operates by forcing all network traffic into a strict transmission schedule. It holds incoming application data in a memory buffer and releases packets only at fixed, predetermined time intervals. If there is no real data available in the buffer, Tamaraw transmits a synthetic dummy packet to maintain the constant-rate transmission. Similarly, Congestion-Sensitive BuFLO (CS-BuFLO) attempts to hide the real timing of the traffic by dynamically adjusting its sending rate based on network conditions. However, it still fundamentally requires queuing packets in a memory buffer to reshape the traffic flow.

While these buffering processes hide the real communication timing, they are impossible to implement in the eBPF Traffic Control layer. In the Linux kernel data plane, it is not possible to hold a packet for a long time. Attempting to queue packets or pause the execution would block the network interface queue and immediately cause a kernel panic or severe packet dropping. As a result, an eBPF program must make an immediate, independent choice for every single packet: it can either allow the packet to pass through or drop it, but it cannot wait.

There are also limits on how much memory a program can use and how it processes information. The verifier forbids unbounded loops, which limits the computational complexity of the code. Furthermore, the amount of memory available for temporary storage is restricted; for instance, the stack memory is limited to only 512 bytes. This prevents the use of large arrays or dynamic memory allocation inside the fast path. This means, complex stateful defenses that need to remember the exact sequence of the last hundreds of packets are incompatible with this environment. Most advanced defenses are designed for the user-space where they have abundant memory, making them difficult to port into the restrictive eBPF data plane.

Because of these strict limitations, specifically the inability to buffer packets and track long term states, we must adopt a randomization approach rather than a strict regularization approach. We cannot force the traffic into a predefined schedule. Instead, we rely on a probabilistic model, deciding for each passing packet whether to apply a countermeasure based on dynamic probabilities. This randomization introduces enough entropy to disrupt the attacker's machine learning classifiers without violating the strict rules of the eBPF verifier.

To navigate these constraints, we designed a client-side defense tailored for eBPF

and QUIC, founded on three core design choices:

- **Client-Only Deployment**: Similar to Cactus, our defense is deployed solely on the client machine. This completely eliminates the need for server-side modifications, ensuring immediate and practical applicability across the existing internet infrastructure without requiring cooperation from content providers.
- **Downlink Obfuscation via Selective Drop**: Because the client cannot force the remote server to inject dummy packets, we introduce entropy on the downlink (ingress) by selectively dropping incoming packets. This forces the server's congestion control mechanism to react naturally, altering the traffic sequence naturally.
- **Uplink Obfuscation via Dummy Packets**: On the uplink (egress), where the client has full control over packet generation, we implement a lightweight dummy packet injection mechanism. This pads the outgoing requests without dropping them, preserving the responsiveness and speed of user interactions.

4.2 Defense Architecture

The defense system is implemented at the Traffic Control (TC) layer within the Linux kernel. To understand the significance of this placement, it is important to note that the TC layer acts as the final gateway for all network traffic entering or leaving the machine. At this level, the eBPF program directly interacts with the `sk_buff` (socket buffer) structure, which provides comprehensive metadata about every packet.

As shown in Figure 4.1, the architecture utilizes eBPF technology to bypass the severe context switching overhead that typically occurs when moving massive amounts of network data back and forth between the kernel and user-space applications. This design ensures that packet processing remains highly efficient and occurs at wire-speed, long before the data is decrypted and reaches the browser application layer.

The overall architecture is deployed entirely on the target client machine. As illustrated in the system diagram, the environment is strictly divided into the user-space and the Kernel Space. The user-space hosts the regular network applications, such as modern web browsers, which generate the initial internet traffic. It also hosts our eBPF control application.

The Kernel Space hosts our custom eBPF defense logic. Within the Kernel Space, the system is structured into two separate processing paths that operate independently on the network interface. In eBPF terminology, a *hook* is a specific attachment point in the network stack that automatically triggers the execution

of a custom program when a network event occurs. Specifically, the egress path intercepts outgoing traffic to probabilistically inject synthetic dummy packets, while the ingress path intercepts incoming data to execute selective packet drops.

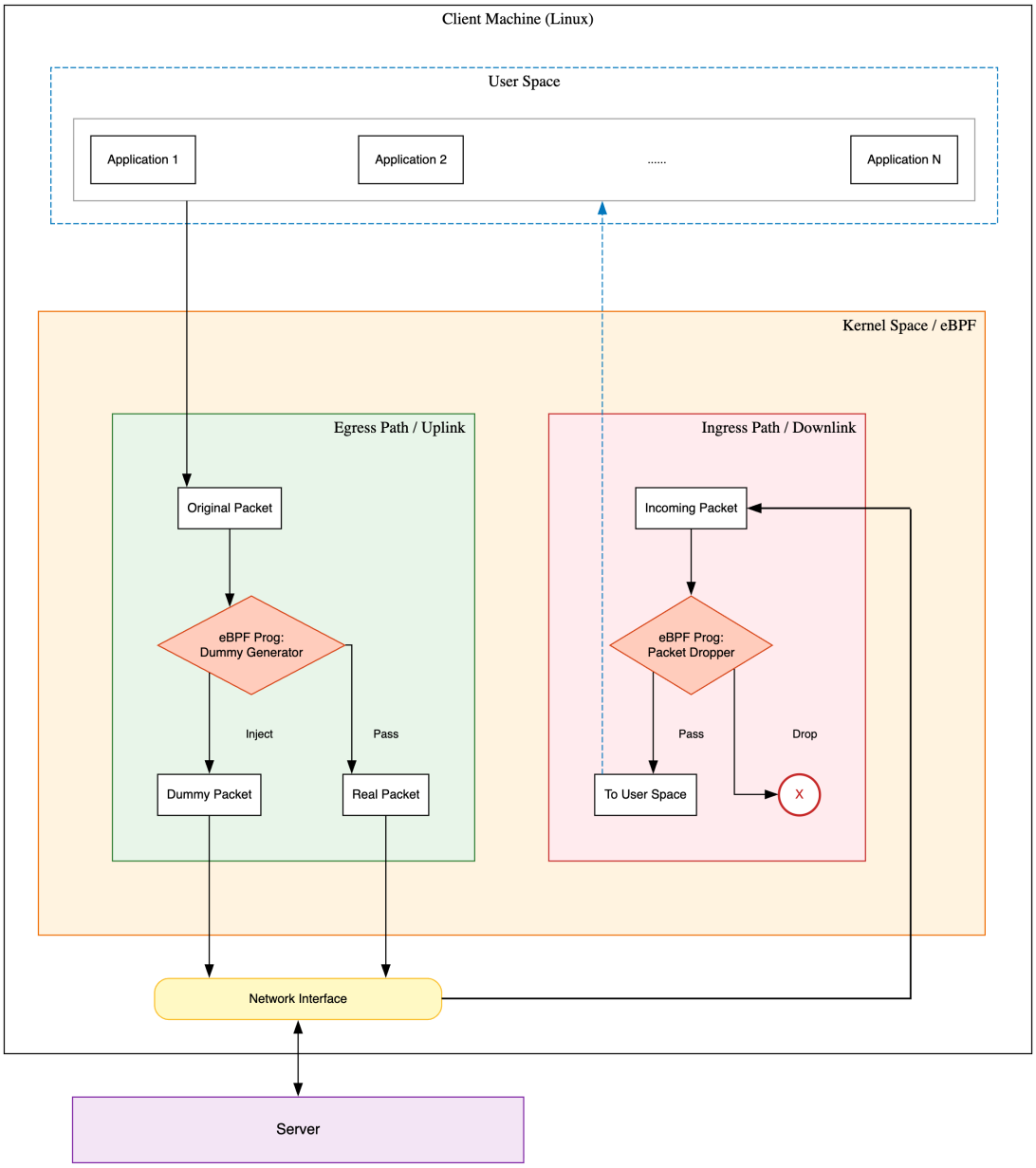

Figure 4.1: Overview of the eBPF-based Defense Architecture. The system operates within the Linux kernel space at the Traffic Control layer, consisting of two asymmetric components: the Egress Hook for synthetic dummy packet injection and the Ingress Hook for probabilistic packet dropping.

4.3 Downlink Channel Obfuscation via Selective Packet Drops

The downlink channel is protected by randomly dropping some incoming packets. This creates unpredictable changes in the network traffic. As a client, we cannot force the external server to send fake packets to us. Therefore, we use an indirect mechanism to change the server behavior. When our eBPF program at the Traffic Control layer intentionally drops an incoming QUIC UDP packet, the server will eventually notice this loss. The server will infer that the physical network is congested. Because of this, the server will automatically start a Retransmission Timeout.

This action forces the server to change how it sends data. It will reduce its transmission speed and resend the missing data later. These automatic reactions break the original sequence of the traffic bursts. They also change the time gaps between packets. By flattening the traffic bursts and adding random delays, this method successfully hides the real identity of the website from the attacker.

Most importantly, this packet dropping method is particularly effective for the QUIC protocol. This is because QUIC has a feature called Stream Multiplexing. Older protocols like TCP have a significant limitation known as Head-of-Line blocking. In TCP, if one packet is lost, the whole connection must stop and wait. QUIC is different. It allows different parts of a web page to travel in separate streams. For example, if we drop a packet that belongs to a background image, the browser can still receive and show the HTML text and CSS files without stopping. Therefore, our defense can keep dropping packets to hide the traffic without making the web page too slow for the user.

4.4 Uplink Obfuscation via Synthetic Traffic Injection

Protecting the uplink channel requires a different approach. We cannot use the dropping method for client requests. If we drop the first few requests from the client, the web page will not load at all, and the user will experience a significant delay. Instead, our system adds fake packets, also known as dummy packets, to the outgoing traffic. This masks the real sending patterns of the user.

To determine exactly when to inject a dummy packet, the eBPF program acts as a probabilistic filter. For every outgoing packet that passes through the Traffic Control (TC) layer, the program computes a dynamic pseudo-random value. It then compares this value against the target injection rate defined by the user-space (for example, 20%). If the probability condition is met, the system triggers the packet cloning process.

To do this, we rely on a specific kernel function called `bpf_clone_redirect`. This function allows our eBPF program to make a copy of a network packet directly inside the kernel memory. This method is called zero-copy, and it is much faster than doing it in the user-space. It avoids the computational overhead of creating entirely new packets or moving data between the operating system and the user applications.

After the eBPF program copies the packet, it must modify it to look like normal background traffic. First, the program changes the size of the payload to a random number. This alters the size patterns of the outgoing traffic. Because the payload size changes, the system must also appropriately update the corresponding packet headers and recompute the network checksums (using helpers like `bpf_csum_diff`). The checksum serves as a strict validation mechanism on the internet. If the checksum is wrong, the network card or the internet routers will classify the packet as corrupted and drop it immediately.

Finally, this dummy packet injection is compatible with the QUIC security model. All QUIC traffic is encrypted using TLS 1.3. Because our eBPF program clones an existing valid packet, the dummy packet keeps the correct UDP routing information and the active QUIC Connection ID (CID). Therefore, it successfully reaches the target server. When the dummy packet arrives, the server cannot tell if it is a real request or a fake one just by looking at the outside. When the server tries to decrypt it, the cryptographic validation fails. According to the official QUIC protocol rules, the server will discard the unreadable data without terminating the active connection. This allows our eBPF system to generate statistical noise without requiring any modifications on the server.

4.5 Design of the Experimental Adaptive Strategy

During the development of this thesis, in addition to the static countermeasures, we also designed and implemented an experimental adaptive system. The primary goal of this adaptive approach was to dynamically adjust the defense intensity based on real time network conditions, rather than relying on static, predefined probability rates.

In theory, an adaptive system should provide superior performance and bandwidth efficiency. It was designed to monitor the active traffic load by calculating the Packets Per Second (PPS). The core logic dictates that the system intelligently applies a higher rate of dummy packets during idle network periods, while scaling back the injection rate during heavy traffic bursts to preserve bandwidth.

To implement this dynamic behavior, we built a control logic loop in the user-space loader. To ensure wire-speed performance, the user-space control plane and

the kernel data plane communicated strictly through shared memory structures known as eBPF Maps. The kernel program continuously monitored the traffic flow and updated the packet statistics in the shared memory.

Meanwhile, the user-space control program utilized a sleep interval to periodically wake up and poll these statistics. It calculated the current PPS, processed the required adjustments on the fly, and then updated the defense configuration parameters in the eBPF Maps. The kernel program would instantly read these new threshold values for the next incoming packet.

Chapter 5

Results

5.1 Experimental Methodology

To test the performance and security of our eBPF defense on QUIC traffic, we built an isolated and automated data collection system.

5.1.1 Testbed Architecture and Network Isolation

A common problem in network measurement is background noise from the host operating system. Operating systems continuously generate unpredictable traffic, such as Network Time Protocol (NTP) synchronizations, Address Resolution Protocol (ARP) broadcasts, and automated software update checks. If captured alongside our dataset, this background noise would introduce severe anomalies into our machine learning features.

To resolve this, it is important to understand the concept of a Linux Network Namespace. In a standard setup, all applications share a single global network stack, meaning browser traffic mixes with system background processes. A Network Namespace, however, is a kernel-level virtualization feature that creates a completely isolated network environment with its own dedicated virtual interfaces and routing tables. By strictly executing our headless Chrome crawler inside this isolated namespace, we guarantee that the resulting PCAP files contain exclusively the QUIC traffic generated by the target website, effectively eliminating system-level noise and ensuring the dataset is what we want.

To ensure that our experiments are reproducible by other researchers, we deployed our isolated testbed on the FABRIC experimental network platform. Specifically, we provisioned a dedicated virtual machine instance hosted within a North American FABRIC cluster. The machine was configured with a 4 core processor and 4 Gigabytes of RAM. It ran a clean installation of the Ubuntu 22.04 LTS operating system. Because eBPF relies on modern kernel features, we ensured

the system utilized a recent Linux kernel version compatible with Traffic Control hooking.

We connected the isolated namespace to the host WAN interface using a virtual Ethernet pair. Inside the namespace, we applied strict network rules to keep a controlled environment. First, we needed to force the browser to use only the QUIC protocol. To do this, we added kernel level nftables rules. These rules explicitly accepted UDP port 443 traffic but blocked older TCP port 443 connections.

Second, we needed to make sure the network speed was stable. If the network speed changes naturally, the Page Load Time measurements become unreliable. To fix this, we used Linux Traffic Control with a Hierarchical Token Bucket on the WAN interface. This acts as a speed limit. It stopped host network overload from affecting the results and kept the bandwidth stable. Finally, we used a custom C-based loader to add our eBPF programs to the network interfaces dynamically. This allowed us to easily switch between different defense countermeasures during our automated tests.

5.1.2 Automated Data Collection Pipeline

For the automated crawling system, we used Python scripts combined with Selenium WebDriver and a Headless Google Chrome browser.

To control the browser, we started Chrome with specific command line flags. For example, we used `--enable-quic` and `--enable-features=UseDnsHttpsSvcb,UseDnsHttpsSvcbAlpn` to force the browser to use HTTP/3. Also, we used `--disk-cache-size=1` and `--disable-application-cache`. We combined these with Chrome DevTools Protocol commands to completely clear the browser cache before every visit. This step is important. It ensures that every page load downloads all images and scripts directly from the network again. This simulates the worst-case scenario for Website Fingerprinting, where the attacker can see the maximum amount of traffic.

To collect the data fairly, our script employed a round-robin approach. Instead of collecting all data for one setup at once, the system switched between different defense setups in small batches. For every target website on our list, the automated browser first performed the required DNS queries to find the server IP address, and then loaded the webpage. We recorded the traffic without any defense. Then, we immediately recorded the traffic with the defense activated. We repeated this process to collect hundreds of automated network traces. This high number of repetitions is necessary to reduce the impact of normal network changes over time.

We captured all network traffic leaving the namespace using tcpdump on the virtual interface. We filtered the capture to only include UDP and port 443. This kept the PCAP file sizes small and easy to process.

After collecting the PCAP files, we analyzed them using a custom Python script

based on the scapy library. We extracted key features typically used by Website Fingerprinting attackers. These features included:

- **Packet Size and Directionality:** We tracked the total bytes and packet counts for both the uplink (from client to server) and the downlink (from server to client).
- **Inter-Arrival Time (IAT):** We calculated the precise microsecond delays between consecutive packets on both channels. Because QUIC's TLS 1.3 encryption hides all internal payload content, attackers rely heavily on these timing intervals to infer the structure of the webpage's DOM tree (e.g., guessing when a large image finishes downloading). By analyzing the IAT, we can precisely observe how our synthetic dummy packets and intentional drops successfully shatter these recognizable timing bursts into random noise.

To measure the cost for the user, we recorded the Page Load Time directly from the browser using the official W3C Performance Navigation Timing API.

The W3C (World Wide Web Consortium) is the main international standards organization for the World Wide Web. Their Performance API is a standard tool built directly into modern web browsers, such as Google Chrome. It provides highly accurate, millisecond level timing data about the entire webpage loading process.

Instead of using a simple external timer in our Python script, which can be inaccurate due to system delays, this built in API allows us to see exactly when the browser's internal engine finishes downloading and rendering the webpage. Using this standardized tool ensures that our latency measurements are scientifically accurate and perfectly reflect the real waiting time experienced by a human user.

Finally, we must state clearly that we did not evaluate our system against real world deep learning attacks. Instead, we use the extracted packet sequences into an unsupervised K-Means clustering algorithm. While modern attackers use Deep Learning (as discussed in Chapter 3), we chose K-Means as our baseline evaluation. This is because K-Means directly measures the fundamental mathematical distance between traffic features. If our defense can destroy the clusters at this basic level, it proves the core concept. We used this algorithm to calculate the V-Measure accuracy score to see if our eBPF defense successfully confused the traffic patterns.

5.1.3 Combined Defense Latency Overhead

We designed our combined defense mechanism to solve the delay problem discussed above while maintaining strong privacy. By keeping the packet drop rate on the downlink strictly at 5% and increasing the non-blocking dummy packet injection on the uplink (up to 20%), we successfully controlled the overall delay. A 5% drop rate is a careful choice; it is enough to trigger the server's congestion control without freezing the webpage entirely.

The results, presented in Figure 5.1, show that the Page Load Time for this complete approach is 1648 ms at a 5% dummy rate and reaches 1825 ms at a 15% rate. At the highest 20% rate, the PLT stabilizes at 1800 ms.

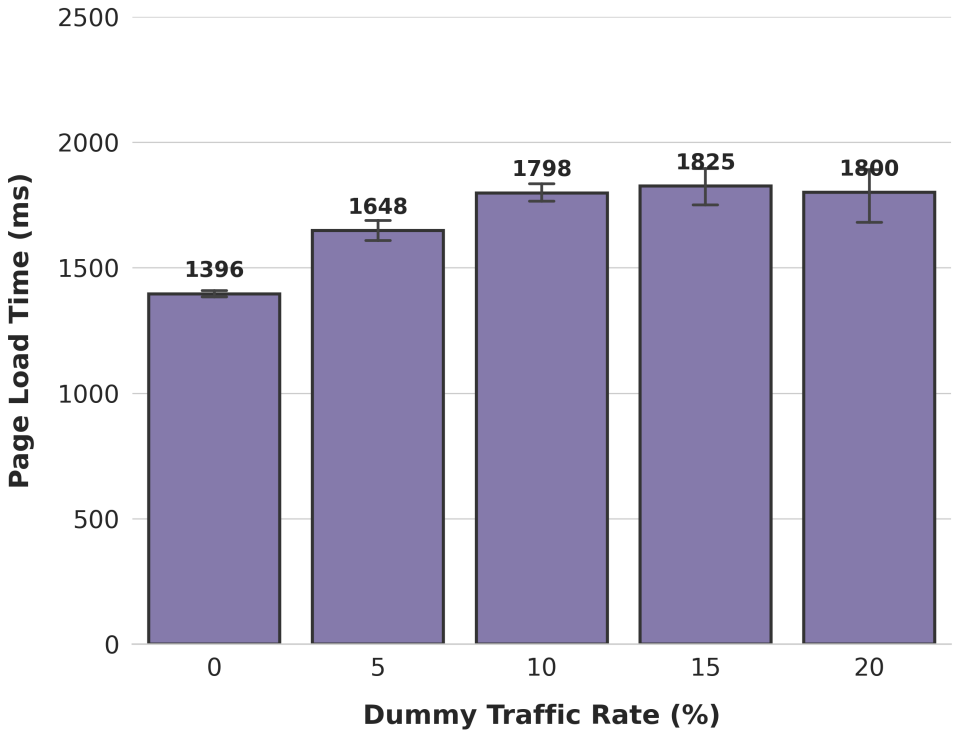

Figure 5.1: Page Load Time (PLT) using the combined defense mechanism (5% drop rate with varying dummy rates).

From a network traffic perspective, this minor stabilization suggests that the asynchronous dummy packets smoothly mix with regular traffic. They provide the necessary statistical noise to confuse classifiers without overwhelming the link capacity. Overall, the combined approach adds only about 400 ms of delay compared to the baseline, proving that splitting the defense asymmetrically effectively avoids severe latency degradation.

5.2 Bytes Overhead Evaluation

While Page Load Time shows the performance perceived by the user, network traffic overhead shows the bandwidth cost of a defense. In this section, we measure the extra bandwidth used by our dummy packet injection method.

As shown in Figure 5.2, the byte overhead grows linearly with the dummy injection rate. At a 5% setting, the system adds an average of 5.2% extra bytes to the flow. This increases to 10.5% and 16.1% as the defense intensity rises. At the maximum 20% rate, the total bandwidth overhead is capped at 21.0%. The small variations observed across the tests reflect the natural differences in web asset sizes. Furthermore, the selective packet drop countermeasure also incurs a tiny byte overhead because it forces the server to resend lost frames and control messages.

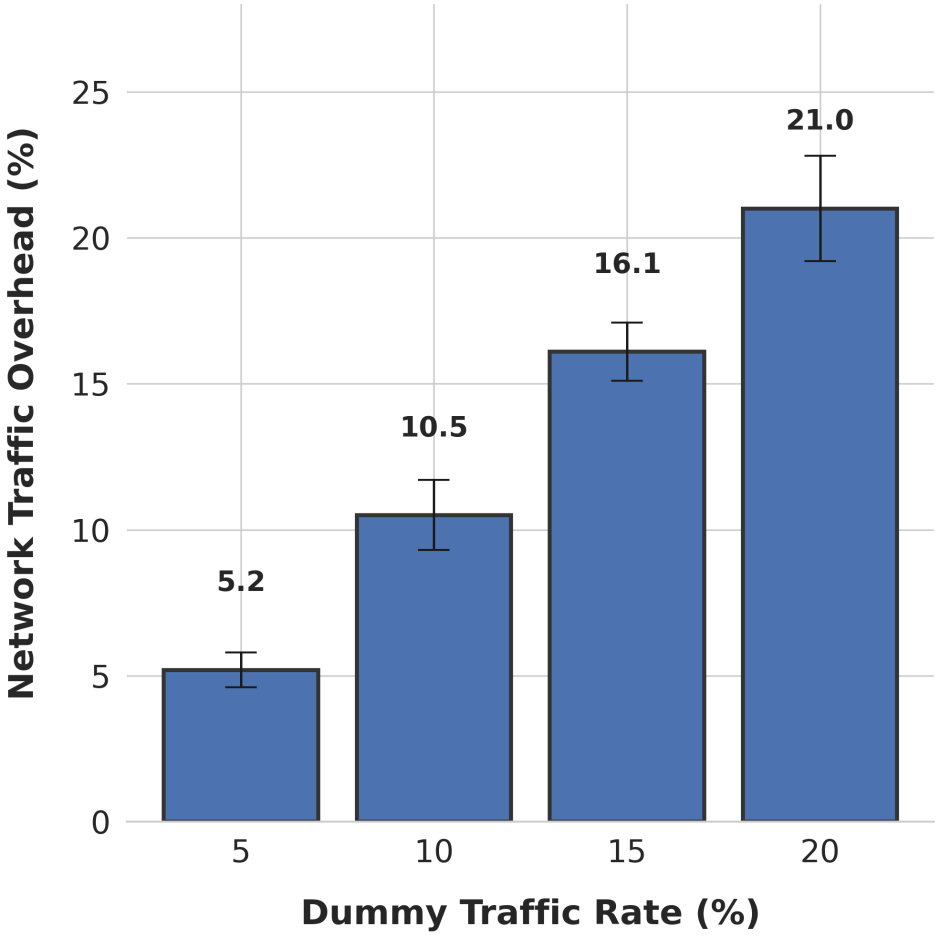

Figure 5.2: Additional network byte overhead across different dummy traffic injection rates.

A maximum overhead of 21.0% is a practical and positive result. Traditional defense methods, such as Tamaraw, often use constant rate padding. This means

they force the system to send dummy packets continuously even when the network is idle and the user is doing nothing. This strict rule frequently inflates bandwidth costs by over 100%.

Our system achieves a much better balance because of three core design choices. First, we use low level packet cloning. By utilizing the `bpf_clone_redirect` helper, the system copies an existing packet directly inside the kernel memory. This saves CPU resources because it does not need to allocate new memory or build new protocol headers from scratch.

Second, we apply asynchronous decorrelation. Although the dummy packets follow real traffic, the system adds random size and timing changes. This creates enough statistical noise to drown out the obvious website fingerprints.

Third, the system operates with almost zero latency overhead. Because the eBPF program runs at the Traffic Control layer, it injects dummy packets alongside the real traffic instantly. This avoids the queuing delays common in user-space applications. Compared to traditional schemes that enforce constant traffic rates, our reactive injection achieves a much better balance between privacy and performance. By linking dummy traffic to real user activity, the system keeps the bandwidth overhead at approximately 21% while maintaining a strong defense against statistical feature analysis.

5.3 Defense Effectiveness on Traffic Patterns

Before measuring the exact numerical accuracy of the attacker, we used a visual tool called t-SNE to further understand our results. This method reduces complex network data into two dimensions so we can observe it visually. If the dots representing different websites group tightly together, the attacker can easily identify them. If the dots mix completely, the defense is successful.

The axes, Dimension 1 and Dimension 2, do not represent specific physical metrics (e.g., packet size or time). Instead, they are artificial coordinates generated by the t-SNE dimensionality reduction algorithm. Our raw traffic feature consists of a multidimensional vector. t-SNE projects this high-dimensional space into a 2D plane to visualize the relative similarity between traffic flows. Proximity in this 2D space implies mathematical similarity in the original high-dimensional traffic fingerprint.

5.3.1 Progression of the Dummy Injection Countermeasure

Figure 5.3 and Figure 5.4 show the visual changes when we test the pure dummy injection countermeasure at lower rates. At a 5% and 10% injection rate, the website groups are clearly separate. The clustering algorithm can easily distinguish the traffic based on these visual boundaries.

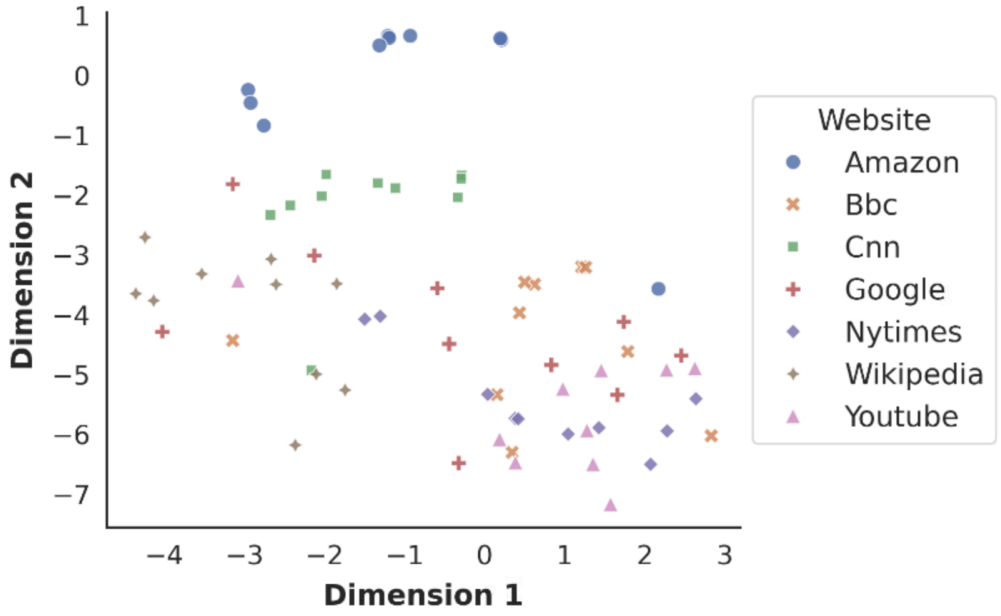

Figure 5.3: t-SNE visual analysis for the dummy injection countermeasure at a 5% injection rate.

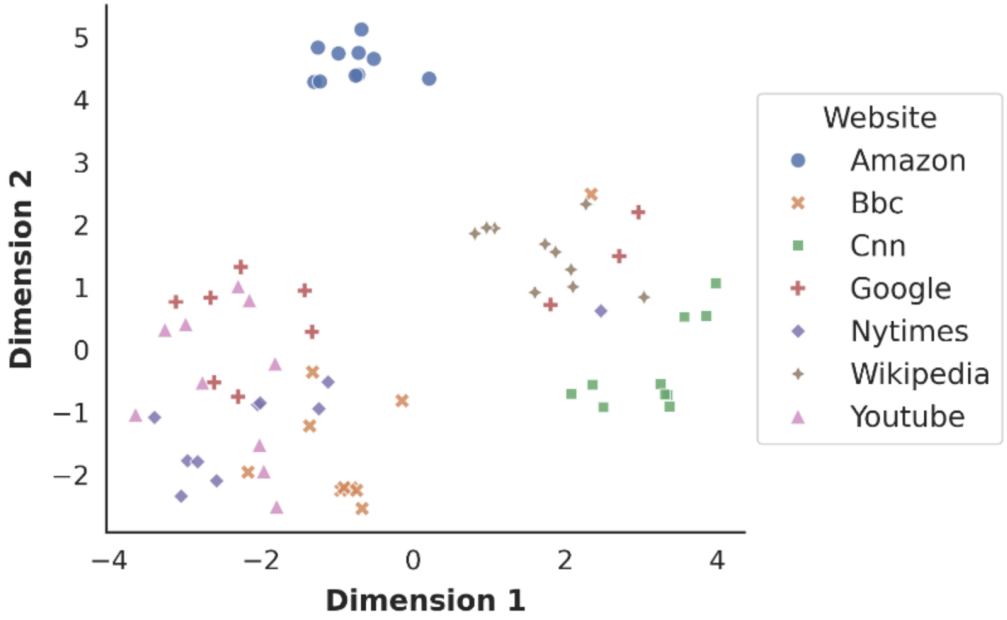

Figure 5.4: t-SNE visual analysis for the dummy injection countermeasure at a 10% injection rate.

As we increase the rate to 15% and 20% in Figure 5.5 and Figure 5.6, the points

spread out slightly. However, the main clusters remain visible. This matches our earlier conclusion that simply adding extra packets is not enough to completely hide the website traffic patterns.

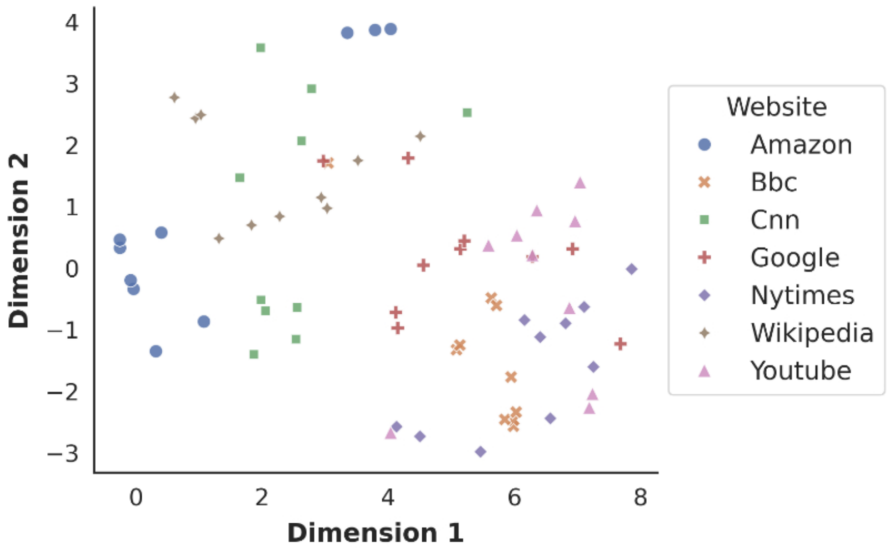

Figure 5.5: t-SNE visual analysis for the dummy injection countermeasure at a 15% injection rate.

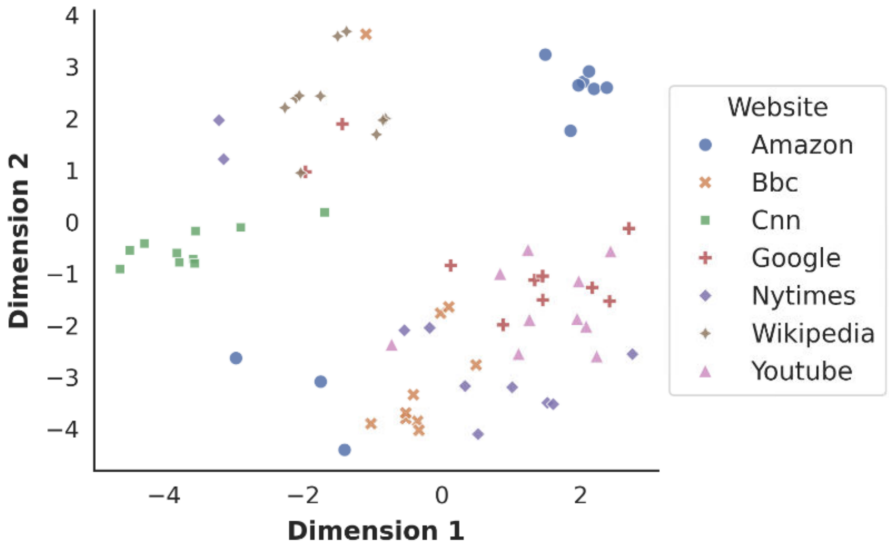

Figure 5.6: t-SNE visual analysis for the dummy injection countermeasure at a 20% injection rate.

5.3.2 Progression of the Packet Drop Countermeasure

The following figures display the results when testing only the packet drop countermeasure at initial stages. Even at 5% and 10% drop rates, the clusters begin to break apart significantly. The original traffic sequences are clearly taking damage.

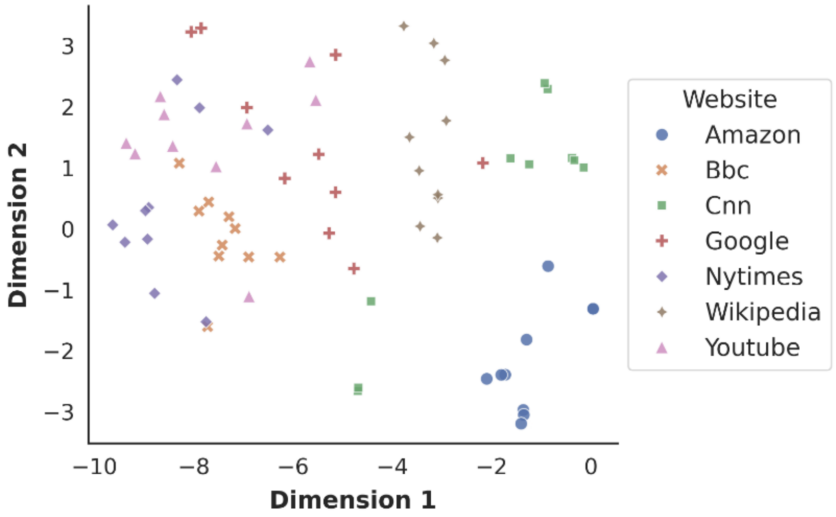

Figure 5.7: t-SNE visual analysis for the packet drop countermeasure at a 5% drop rate.

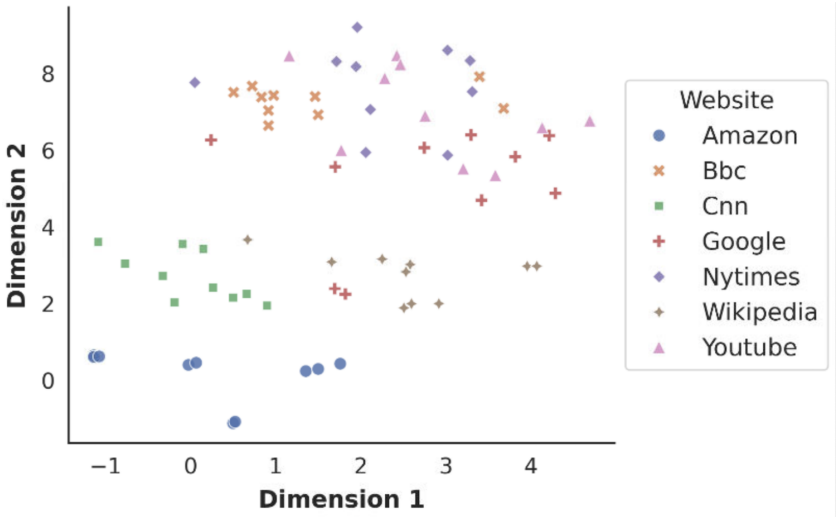

Figure 5.8: t-SNE visual analysis for the packet drop countermeasure at a 10% drop rate.

When the drop rate reaches 15% and 20%, the points scatter widely across the graph. The boundaries between different websites disappear entirely. This visual shows strong security, but as discussed before, it causes an unacceptable delay for the user.

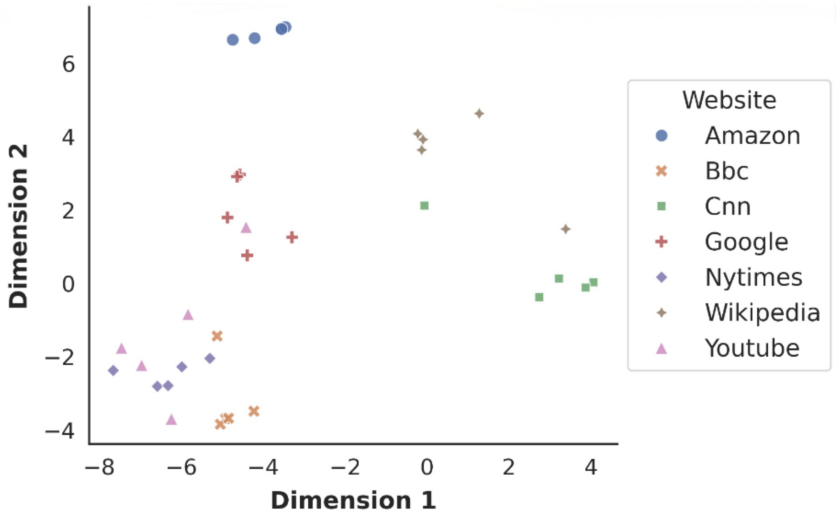

Figure 5.9: t-SNE visual analysis for the packet drop countermeasure at a 15% drop rate.

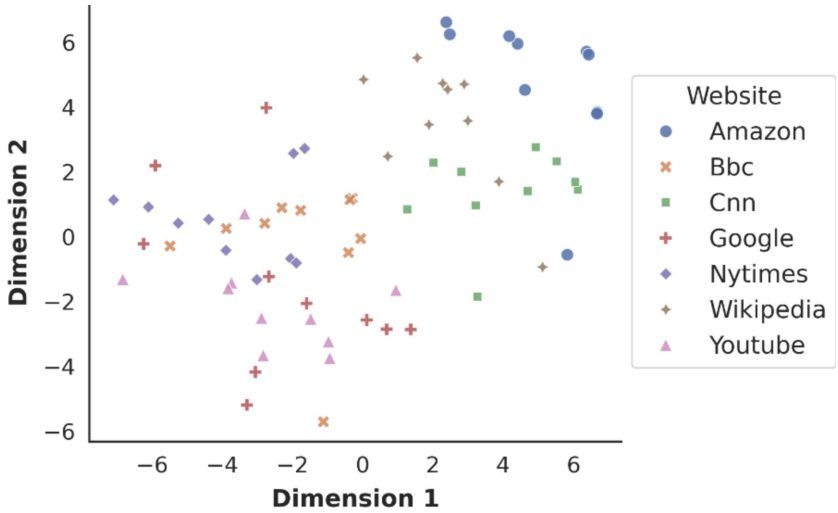

Figure 5.10: t-SNE visual analysis for the packet drop countermeasure at a 20% drop rate.

5.3.3 Progression of the Combined Defense

Finally, we present our complete combined defense. Because we keep a fixed 5% packet drop rate, even a small 5% or 10% dummy injection causes the groups to start mixing. The initial drops disrupt the timing, making the dummy packets more effective.

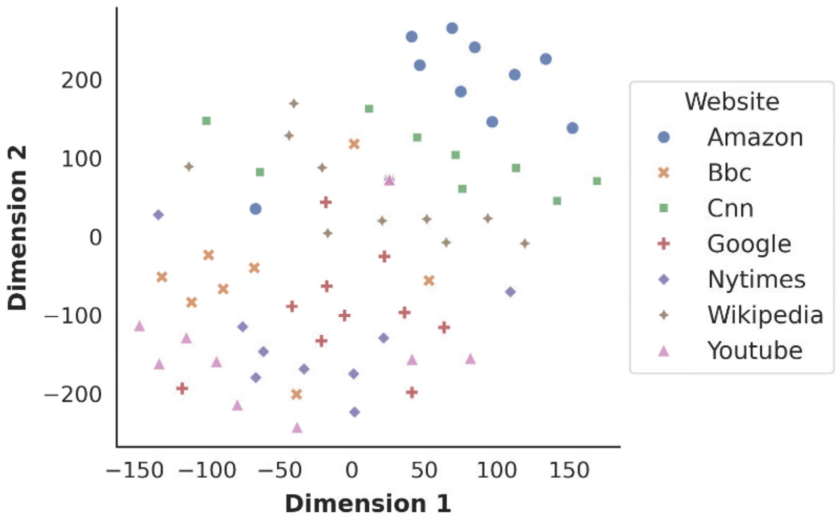

Figure 5.11: t-SNE visual analysis for the combined defense (fixed 5% drop) with a 5% dummy rate.

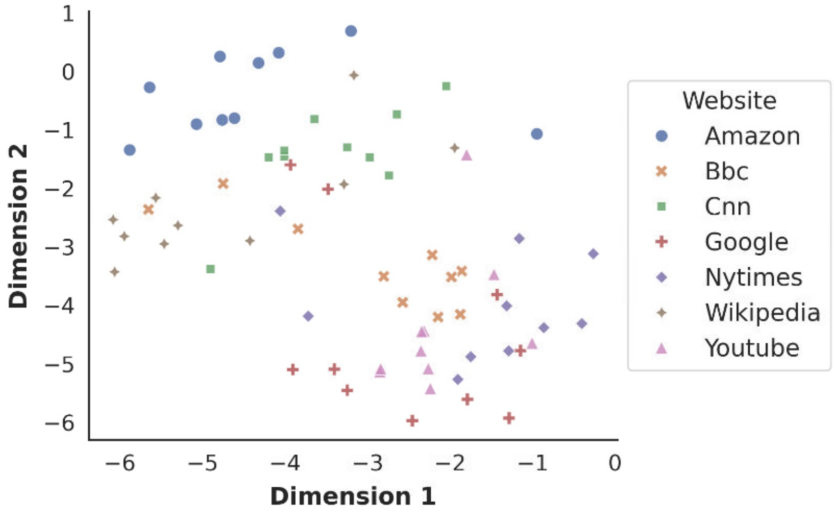

Figure 5.12: t-SNE visual analysis for the combined defense (fixed 5% drop) with a 10% dummy rate.

By the time we reach the 15% and 20% dummy rates, the data points overlap completely into a random cloud. The clustering algorithm can no longer find any clear features. This perfectly proves that combining a small drop rate with extra dummy packets successfully secures the network connection.

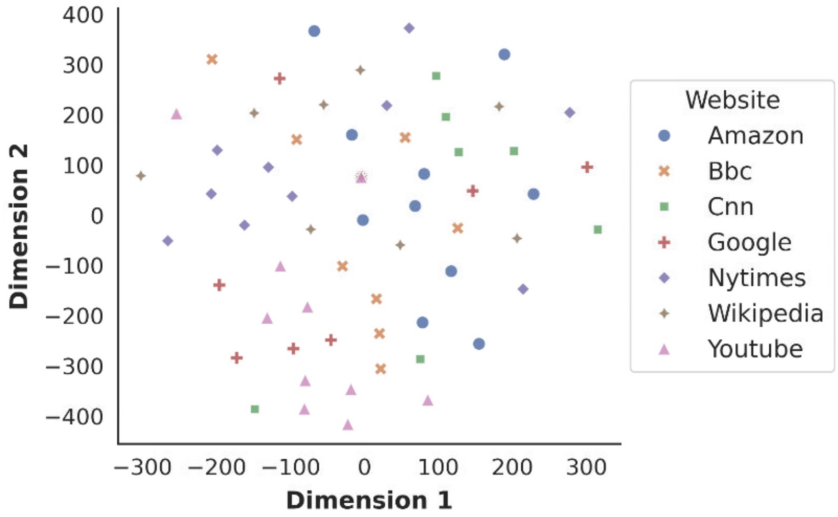

Figure 5.13: t-SNE visual analysis for the combined defense (fixed 5% drop) with a 15% dummy rate.

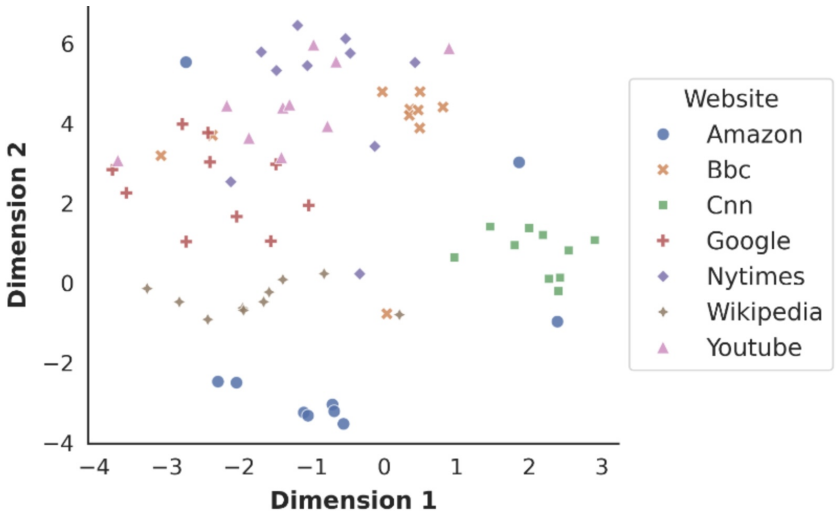

Figure 5.14: t-SNE visual analysis for the combined defense (fixed 5% drop) with a 20% dummy rate.

5.4 Defense Effectiveness on Clustering

Building on the visual observations from the t-SNE graphs, the primary objective of our defense system is to numerically prevent attackers from identifying target websites. We evaluate this security level using the V-Measure score from an unsupervised K-Means clustering algorithm. We chose V-Measure because it evaluates two important factors: homogeneity (whether a cluster contains only traffic from one specific website) and completeness (whether all traffic from that website is assigned to the same cluster). It outputs a final score between 0% and 100%. In our context, a high score means the attacker can cleanly group and identify different websites, while a lower score indicates that the defense successfully mixed the original traffic patterns.

5.4.1 Effect of Dummy Packets Injection on Clustering

As shown in Figure 5.15, without any defense, the attacker achieved a high accuracy of 72.4%. This result proves that regular QUIC traffic contains clear patterns that are easy to identify.

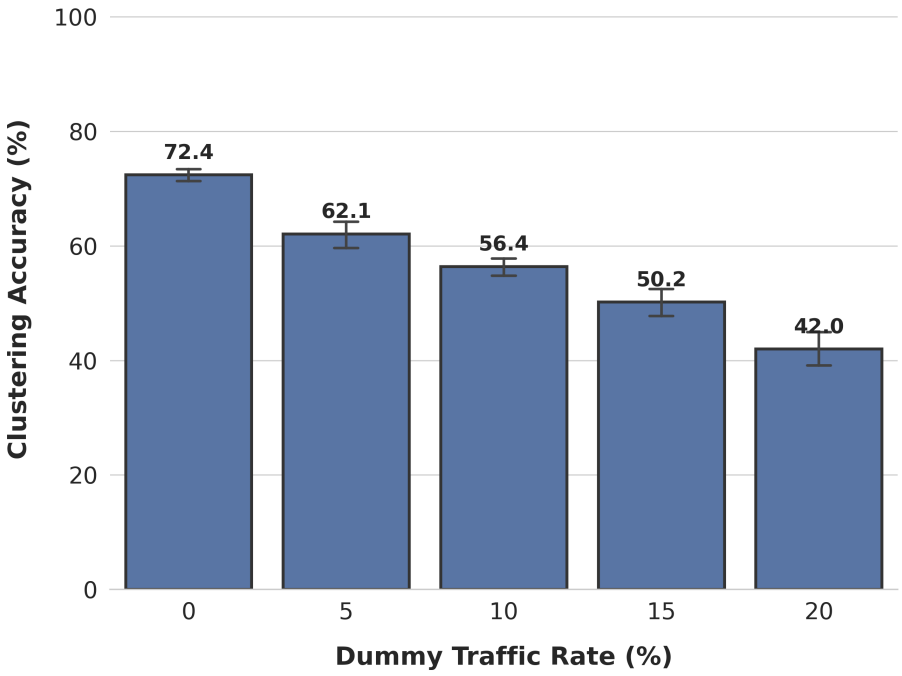

Figure 5.15: Clustering accuracy under varying dummy packet injection rates.

When we tested only the dummy packet injection countermeasure, the system provided moderate protection. As the injection rate increased, the accuracy slowly dropped. At the maximum 20% dummy rate, the accuracy decreased to 42.0%. Adding extra dummy packets helps confuse the classifier, but it cannot hide the large traffic bursts generated by heavy web pages. This is because injecting packets only adds volume, but the timing of the server's large downlink bursts remains unchanged and predictable.

5.4.2 Effect of Packet Drops on Clustering

The selective packet drop countermeasure was highly effective at breaking the original traffic structure. By intentionally discarding packets, the defense forces the server to resend data and modifies the congestion window. This process destroys the predictable packet sequences and creates random delays.

Figure 5.16 shows a steep decline in accuracy. At a 20% drop rate, the accuracy fell to a very low 24.5%. However, as discussed in the latency evaluation section, this method causes a massive delay of approximately 3150 ms. Therefore, it is not practical for real users, even though it provides strong security.

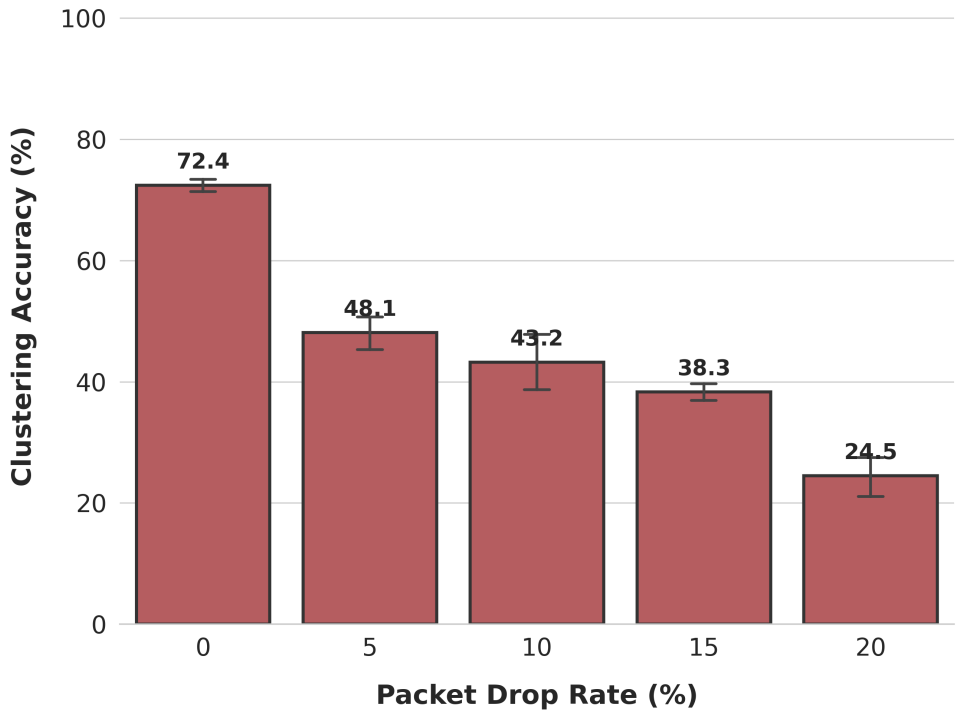

Figure 5.16: Clustering accuracy under varying packet drop rates.

5.4.3 Effect of the Combined Defense on Clustering

Our combined defense mechanism produced the best overall results. We kept the packet drop rate low at 5% to avoid huge delays and scaled up the dummy packets to 20%.

As illustrated in Figure 5.17, the accuracy dropped sharply from the baseline of 72.4% down to 31.2% at the maximum intensity. This trend demonstrates a strong combined effect. A small number of dropped packets slightly shifts the original traffic timing and creates random gaps. When the dummy packets fill these gaps unpredictably, it becomes extremely difficult for the K-Means algorithm to extract reliable statistical features.

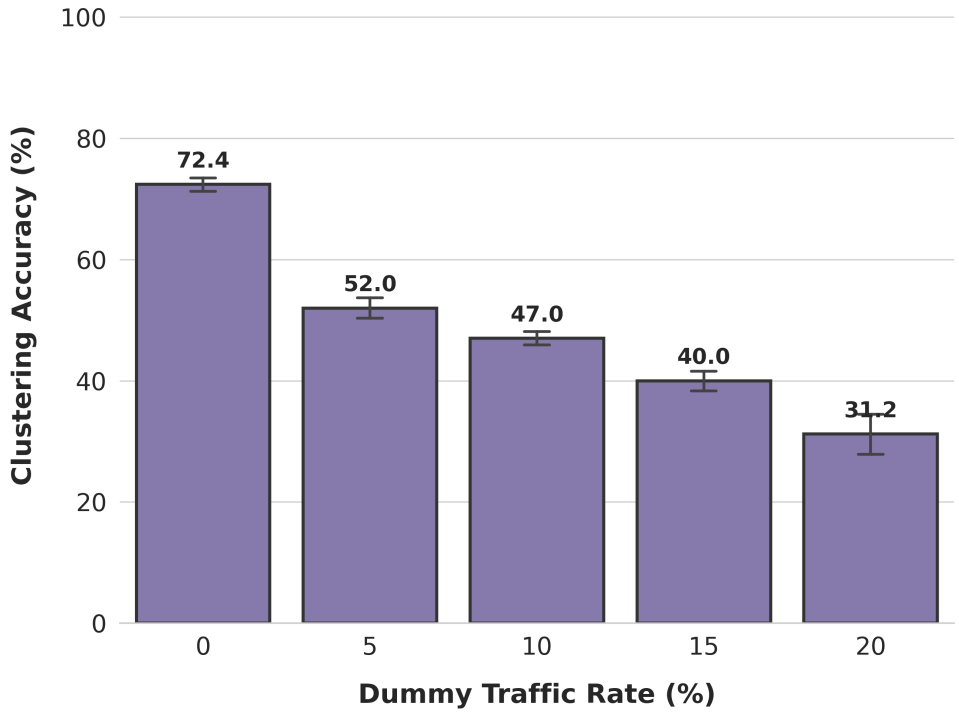

Figure 5.17: Clustering accuracy using the combined defense (5% drop rate with varying dummy rates).

5.5 Privacy and Latency Tradeoff

The overall success of our system is clearly visible in the security and usability tradeoff plot in Figure 5.18. A practical defense must sit in the bottom left area of the chart, which indicates both low attacker accuracy and low latency for the user.

The pure packet drop test (red circles) offers strong security but unacceptable delay. The pure dummy injection test (blue squares) maintains excellent delay but lacks deep security. Our combined defense (purple diamonds) achieves the ideal balance. It successfully reduces the accuracy to approximately 31% while keeping the Page Load Time highly usable at around 1800 ms.

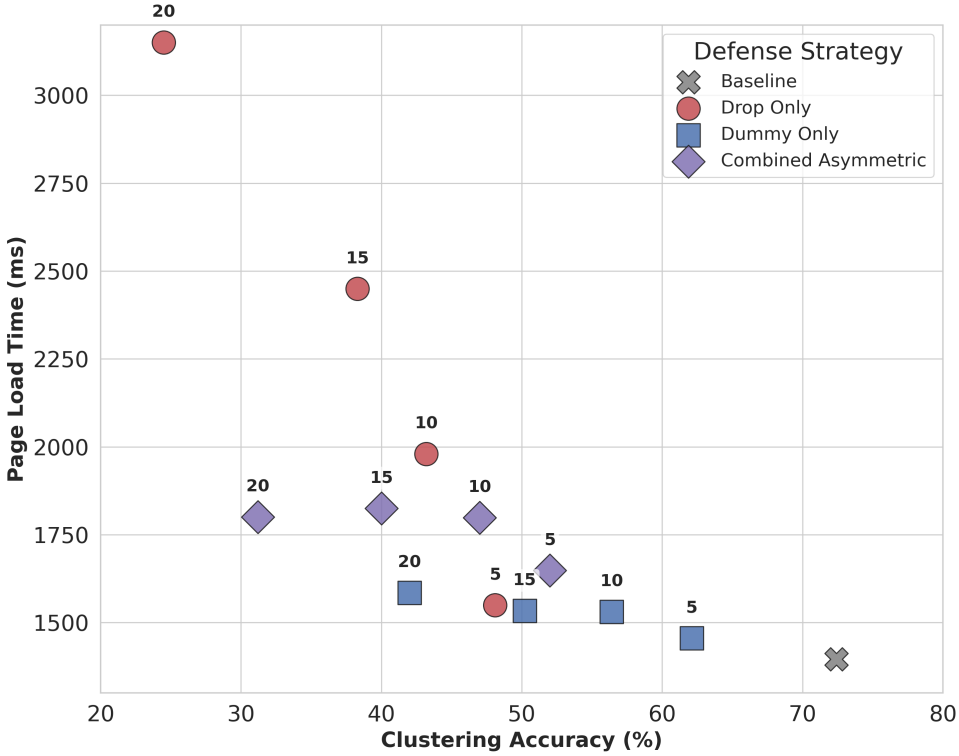

Figure 5.18: Tradeoff between Page Load Time and Clustering Accuracy across all tested conditions.

5.6 Evaluation of the Adaptive Strategy

As detailed in Chapter 4, we implemented an experimental adaptive system designed to dynamically adjust defense rates based on real time network conditions. However, the experimental results did not meet our initial expectations. When we evaluated this adaptive mechanism using the interleaved round-robin testing methodology, it performed slightly worse than our static combined defense. The system experienced unexpected latency spikes, and the overall clustering accuracy of the attacker was not significantly reduced.

After conducting a deep code level investigation into the underlying eBPF

implementation, we identified two major architectural bottlenecks that caused this performance degradation:

Firstly, the system suffered from a delayed reaction time. The user-space control program used a one-second sleep interval to poll the network metrics and calculate the current Packets Per Second (PPS). However, modern web navigation bursts over QUIC are fast and often finish entirely within one to two seconds. Because checking the traffic rate every full second is too slow, the control plane was always lagging behind the actual network state. As a result, the adaptive dummy packet injection often triggered just as the actual traffic burst was already ending. This injected noise at the wrong time, which unnecessarily delayed the tail connections and inflated the Page Load Time.

Secondly, the system suffered from an inadequate baseline threshold configuration. The dynamic minimum PPS threshold for triggering the countermeasures was set too low for modern web environments. When utilizing the QUIC protocol, a regular webpage load generates intense traffic bursts that easily exceed this low limit within just a few milliseconds. Because of this, the control logic falsely interpreted normal web rendering loads as massive traffic spikes. This caused the system to constantly trigger the maximum dummy injection rate. These unnecessary fake packets aggressively competed with legitimate application data for the limited bandwidth available at the Traffic Control layer.

Therefore, while our static combined defense proved to be highly effective and stable, evaluating the fundamental causes of this failure provides valuable engineering insights. Optimizing the polling interval and adjusting the PPS thresholds of this adaptive mechanism remain clear objectives for future research.

Chapter 6

Conclusions and Future Works

QUIC is rapidly becoming the dominant transport protocol, bringing improvements in speed. However, it remains highly vulnerable to Website Fingerprinting. Existing defenses have critical limitations. Some defenses only target older protocols like TCP (such as Cactus). Others operate at the application layer, which provides no guarantee of controlling the actual packet sequences that travel on the physical network. Specialized defenses like QCSD require significant modifications to the QUIC source code. Before this research, a client-only, application-transparent, and packet-level defense for QUIC did not exist.

To solve this problem, this thesis presented an eBPF defense at the Traffic Control (TC) layer. This system combines selective packet dropping on the downlink with dummy packet injection on the uplink. The system operates directly in the data plane, manipulating the exact packet-level traffic that an attacker observes. Importantly, it requires no changes to user applications, the QUIC stack, or the server infrastructure.

Our evaluation tested individual countermeasures and our complete defense mechanism. The pure dummy packet injection countermeasure provides low overhead but limited obfuscation. The selective packet drop countermeasure achieves strong obfuscation but causes an unacceptable latency, driving the delay up to 3150 ms at a 20% drop rate. Our combined defense (using a 5% drop rate and a 20% dummy rate) achieves the best balance. Under this approach, the clustering accuracy of the attacker was reduced from 72.4% down to 31.2%. This was achieved with only 400 ms of additional latency and a 21% bandwidth overhead. In addition, our visual analysis confirmed the successful mixing of traffic clusters.

It is important to acknowledge the limitations of this evaluation. The current assessment uses an unsupervised K-Means clustering algorithm rather than deep

learning classifiers. Additionally, the testbed is highly controlled and may not fully reflect unpredictable real network conditions. Finally, the website dataset is limited to seven target sites.

In conclusion, this work clearly demonstrates that packet-level and data plane traffic obfuscation for QUIC is both feasible and practical. It successfully opens a path toward privacy-preserving web browsing in the HTTP/3 era.

6.1 Future Works

This research builds a solid foundation and opens up several clear directions for future work.

First, the primary direction is to evaluate our defense against more powerful attackers. In this thesis, we only used a simple K-Means clustering algorithm to measure the security baseline. Future research should use advanced deep learning models, such as Convolutional Neural Networks. Testing against these complex models will help us understand how well the defense truly works in real-world scenarios.

Second, a major goal is to optimize the adaptive defense system. As discussed in the results chapter, our initial tests for the adaptive system underperformed because it reacted too slowly. The control program checked the network traffic only once every second, which is insufficient to catch the fast web traffic bursts. Future work should speed up this polling process to the microsecond level. By resolving this timing issue, the system could dynamically add more dummy packets when the network is free, and stop adding them when the network is busy, significantly saving bandwidth.

Finally, this defense could be deployed to the server side. Currently, our system only runs on the client. If a large Content Delivery Network (CDN) operator adopts our eBPF system on their servers, it can protect millions of users at the same time without requiring any client-side modifications. Furthermore, these eBPF programs can run directly on special network cards called SmartNICs. This means the defense will run purely on the hardware level, protecting user privacy without consuming the CPU power of the main web servers.

Bibliography

[1] Elisaveta Lavrentieva, Marc Juarez, and Michio Honda. «Rethinking the Role of Network Stacks for Website Fingerprinting Defenses». In: *Proceedings of the 24th ACM Workshop on Hot Topics in Networks*. HotNets '25. UMD Campus, College Park, MD, USA: Association for Computing Machinery, 2025, pp. 254–262. ISBN: 9798400722806. DOI: 10.1145/3772356.3772428. URL: https://doi.org/10.1145/3772356.3772428 (cit. on p. 2).

[2] IETF. *HTTP/3*. Tech. rep. RFC 9114, 2022 (cit. on p. 4).

[3] Cloudflare. *A QUIC look at HTTP/3*. 2021. URL: https://blog.cloudflare.com/http3-usage-and-performance/ (cit. on p. 4).

[4] IETF. *QUIC: A UDP-Based Multiplexed and Secure Transport*. Tech. rep. RFC 9000, 2021 (cit. on p. 4).

[5] IETF. *Using TLS to Secure QUIC*. Tech. rep. RFC 9001, 2021 (cit. on p. 4).

[6] IETF. *Problem Statement and Requirements for Increased User Privacy in the Modern Web*. Tech. rep. RFC 7498, 2015 (cit. on p. 4).

[7] Andriy Panchenko et al. «A survey of website fingerprinting techniques». In: (2016) (cit. on p. 5).

[8] Marc Juarez et al. «Effective and efficient website fingerprinting defenses». In: 2016 (cit. on p. 5).

[9] Martin Risch and Martin D. Tietz. «Website Fingerprinting in the Age of QUIC». In: 2021 (cit. on p. 5).

[10] Luigi Rizzo. «netmap: a novel framework for fast packet I/O». In: 2012 (cit. on p. 6).

[11] Payap Sirinam et al. «Deep fingerprinting: Undermining website fingerprinting defenses with deep learning». In: 2018 (cit. on p. 6).

[12] Steven McCanne and Van Jacobson. «The BSD Packet Filter: A New Architecture for User-level Packet Capture». In: *Proceedings of the USENIX Winter 1993 Conference*. USENIX Association, 1993 (cit. on p. 6).

[13] *eBPF.io: Introduction, Tutorials and Community Resources*. https://ebpf.io/. 2024 (cit. on p. 8).

[14] *Linux Kernel Documentation: eBPF Verifier*. https://www.kernel.org/doc/html/latest/bpf/verifier.html. 2024 (cit. on p. 8).

[15] eBPF Foundation. *What is eBPF? An Introduction and Deep Dive into the eBPF Technology*. https://ebpf.io/what-is-ebpf/. Accessed: 2026-03-14. 2024 (cit. on p. 8).

[16] *libbpf: C/C++ eBPF library*. https://github.com/libbpf/libbpf. 2024 (cit. on p. 8).

[17] *Aya: eBPF library for the Rust programming language*. https://aya-rs.dev/. 2024 (cit. on p. 8).

[18] *Cilium: eBPF-based Networking, Observability, Security*. https://cilium.io/. 2024 (cit. on p. 8).

[19] Liz Rice. *Learning eBPF: Programming the Linux Kernel for Enhanced Observability, Networking, and Security*. O'Reilly Media, 2023 (cit. on p. 9).

[20] Toke Høiland-Jørgensen, Jesper Dangaard Brouer, Daniel Borkmann, John Fastabend, Tom Herbert, David Ahern, and David Miller. «The eXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel». In: *Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. ACM, 2018 (cit. on p. 9).

[21] Feng Wang et al. «OXDP: Accelerating Packet Processing by Offloading XDP onto SmartNICs». In: *IEEE Transactions* (2023) (cit. on p. 9).

[22] IBM. *What is the control plane?* https://www.ibm.com/topics/control-plane. Accessed: 2024. 2024 (cit. on p. 11).

[23] X. Wang et al. «Cactus: Application-Transparent Traffic Obfuscation using eBPF». In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. 2024 (cit. on pp. 11, 12, 14–17).

[24] Maohua Guo and Jinlong Fei. «Website Fingerprinting Attacks Based on Homology Analysis». In: *Security and Communication Networks* 2021 (2021) (cit. on p. 13).

[25] Jamie Hayes and George Danezis. «k-fingerprinting: A Robust Scalable Website Fingerprinting Technique». In: *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, 2016, pp. 1187–1203 (cit. on p. 14).

[26] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. «Website Fingerprinting at Internet Scale». In: *Network and Distributed System Security Symposium (NDSS)*. 2016 (cit. on p. 14).

[27] Xinhao Deng, Qilei Yin, Zhuotao Liu, Xiyuan Zhao, Qi Li, Mingwei Xu, Ke Xu, and Jianping Wu. «Robust Multi-tab Website Fingerprinting Attacks in the Wild». In: *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2023, pp. 123–140 (cit. on p. 14).

[28] Alireza Bahramali, Amir Houmansadr, et al. «Realistic Website Fingerprinting By Augmenting Network Trace». In: *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2023 (cit. on p. 14).

[29] Meng Shen et al. «Subverting Website Fingerprinting Defenses with Robust Traffic Representation». In: *32nd USENIX Security Symposium (USENIX Security 23)*. 2023 (cit. on p. 14).

[30] Yuwen Cui, Guangjing Wang, Khanh Vu, Yao Liu, et al. «A Comprehensive Survey of Website Fingerprinting Attacks and Defenses in Tor: Advances and Open Challenges». In: *arXiv preprint arXiv:2510.11804* (2025) (cit. on p. 14).

[31] Yifei Cheng et al. «HOLMES & WATSON: A Robust and Lightweight HTTPS Website Fingerprinting through HTTP Version Parallelism». In: *Proceedings of the ACM on Web Conference 2025 (WWW)*. 2025 (cit. on p. 14).

[32] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. «Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning». In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2018, pp. 1928–1943 (cit. on p. 14).

[33] Xiaobo Ma et al. «Website Fingerprinting on Encrypted Proxies: A Flow-Context-Aware Approach and Countermeasures». In: *IEEE/ACM Transactions on Networking* (2024) (cit. on p. 14).

[34] Siyang Chen et al. «Causality Correlation and Context Learning Aided Robust Lightweight Multi-Tab Website Fingerprinting Over Encrypted Tunnel». In: *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*. IEEE. 2024 (cit. on p. 14).

[35] Diwen Xue et al. «Fingerprinting Obfuscated Proxy Traffic with Encapsulated TLS Handshakes». In: *33rd USENIX Security Symposium (USENIX Security 24)*. 2024 (cit. on p. 14).

[36] Hongbo Xu et al. «ProxyKiller: An Anonymous Proxy Traffic Attack Model Based on Traffic Behavior Graphs». In: *European Symposium on Research in Computer Security (ESORICS)*. Springer. 2024 (cit. on p. 14).

[37] Meng Shen et al. «Swallow: A Transfer-Robust Website Fingerprinting Attack via Consistent Feature Learning». In: *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. 2025 (cit. on p. 14).

[38] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. «Effective Attacks and Provable Defenses for Website Fingerprinting». In: *23rd USENIX Security Symposium (USENIX Security 14)*. USENIX Association, 2014, pp. 143–157 (cit. on p. 14).

[39] Marc Juarez, Mohsen Imani, Mike Perry, Claudia Diaz, and Matthew Wright. «WTF-PAD: Toward an efficient website fingerprinting defense». In: *European Symposium on Research in Computer Security*. Springer. 2016, pp. 27–46 (cit. on p. 15).

[40] Meng Shen, Kexin Ji, Jinhe Wu, Qi Li, Xiangdong Kong, and Ke Xu. «Real-Time Website Fingerprinting Defense via Traffic Cluster Anonymization». In: *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2024, pp. 3238–3256 (cit. on p. 15).

[41] L. Wang et al. «A Dynamic Website Fingerprinting Defense by Emulating Spatio-Temporal Traffic Features». In: *Electronics* 14 (2024) (cit. on p. 15).

[42] Faqi Zhao et al. «Nüwa: Enhancing Network Traffic Analysis With Pre-Trained Side-Channel Feature Imputation». In: *IEEE/ACM Transactions on Networking* (2025) (cit. on p. 15).

[43] Xinbo Han et al. «DE-GNN: Dual embedding with graph neural network for fine-grained encrypted traffic classification». In: *Computer Networks* (2024) (cit. on p. 15).

[44] Chenxu Wang et al. «VPNSniffer: Identifying VPN Servers Through Graph-Represented Behaviors». In: *Proceedings of the ACM on Web Conference 2024 (WWW)* (2024) (cit. on p. 15).

[45] Adam Langley et al. «The QUIC Transport Protocol: Design and Internet-Scale Deployment». In: *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2017, pp. 183–196 (cit. on p. 16).

[46] Y. Gong, T. Li, and G. Min. «Traffic Confirmation Attacks Despite Noise: A Case Study on Low-Latency Mix Networks». In: *Proceedings of the IEEE International Conference on Communications (ICC)*. 2020 (cit. on p. 16).

[47] J. Iyengar and M. Thomson. *QUIC Loss Detection and Congestion Control*. RFC 9002. 2021 (cit. on pp. 16, 17).