Anomaly Detection in In-Network Fast ReRoute Systems

Divya Pathak, Harish S A, Sree Prathyush Chinta, Dilip Kumar Reddy, Praveen Tammana

Indian Institute of Technology Hyderabad, India

***Abstract*—High-speed programmable data planes provide opportunities to implement data-driven fast reroute systems that quickly adapt to varying network conditions (e.g., congestion, failures) and improve network performance. The core of these systems has packet-processing algorithms running in the data plane that continuously look for traffic patterns (e.g., too many retransmissions) specific to a network condition (e.g., link failure) and take appropriate action (e.g., reroute). Despite their benefits, it also increases the potential attack surface. Adversaries can generate malicious traffic patterns similar to those a fast reroute system is looking for and trick the system. Doing so would lead to poor network performance due to incorrect reroute decisions.**

In this paper, we propose a mechanism to detect whether the fast reroute systems are under the influence of malicious traffic patterns. Our key idea is to model the expected behavior using benign traffic features and use the model as a reference to determine whether the system is under the influence of adversaries. Using realistic attack traces, we demonstrate attacks on two fast reroute systems and successfully detect those attacks using the proposed detection mechanism.

I. Introduction

High-speed programmable data planes [1]–[3] coupled with domain-specific programming languages (*e.g.,* P4 [4]) have emerged as promising building blocks for the development of in-network applications [5]–[13]. The core of these in-network applications has packet-processing algorithms designed to run entirely in the data plane and make fast, precise, and data-driven decisions to improve network performance. For instance, a category of work [10]–[19] makes reroute or load balance decisions on network traffic as a reaction to dynamic network conditions such as link failures and congestion. We call such in-network applications as Fast ReRoute systems (FRR).

Packet-processing algorithms in FRR systems operate in two phases: monitoring and control. During the monitoring phase, the algorithm processes incoming packet headers and maintains (or updates) a state across packets. In the control phase, the algorithm checks for traffic patterns (*e.g.,* many TCP flows experiencing retransmissions) by analyzing the state, infers associated network condition (*e.g.,* downstream link failure), and takes an appropriate decision (*e.g.,* reroute a prefix traffic to a backup path).

Though such novel FRR systems help to improve network performance, they increase the attack surface and are hence vulnerable to adversarial inputs not seen before. More specifically, an attacker can exploit the semantics of each phase and craft adversarial inputs such that the FRR system's decisions are influenced and degrade the performance of a significant portion of the traffic. Interestingly, to generate adversarial inputs, the attacker does need specific privileges: the attacker can use compromised hosts to generate flows with crafted packet headers with an objective of polluting the state maintained by the FRR system. One approach to protect the FRR system state is to authenticate traffic using cryptographic approaches [20]. However, the P4 data plane supports a limited set of operations with no support for loops and recursions making it challenging to run sophisticated cryptographic primitives in the data plane.

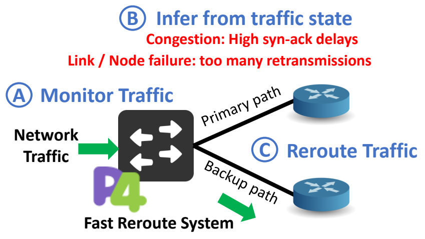
ISBN 978-3-903176-63-8 © 2024 IFIP

Our goal is to protect FRR systems from adversarial network inputs. Towards this goal, in this paper, we propose a mechanism to detect whether the state maintained by an FRR system in the data plane is manipulated by adversaries. Our approach is to gather features of traffic flows updating the state and compare these features' behavior under normal (benign) conditions (*expected behavior*) with the behavior of features observed at runtime. If the observed behavior significantly deviates from the expected behavior, then alerts are raised to help take appropriate action (*e.g.,* redirect traffic to a deep inspection system for further analysis).

There are two main challenges in realizing our approach. First, it is hard to differentiate whether the data plane state is manipulated by adversarial traffic or is updated by abnormal traffic under benign network conditions (*e.g.,* link failure, congestion). To address this, while constructing the expected behavior, we carefully incorporate features' behavior under various benign network conditions such that our detection system can distinguish attacks from abnormal but benign situations (more details in §IV). Second, due to limited data plane resources and hardware constraints, it is crucial to identify a small set of important traffic features (over many) to be collected to capture actual (or observed) behavior without compromising the detection accuracy. We address this by carefully analyzing the detection accuracy of various features and picking those features that give the best accuracy (more details in §V).

The main contributions of this paper are as follows:

- We demonstrate attacks on two FRR systems: Blink [10] and RouteScout delay monitor [12] (§III). Due to the lack of attack datasets, we generate realistic attack traces by interlacing attack traffic with original CAIDA traces [21] so that traffic characteristics observed in a real network can be preserved.



B Infer from traffic state
Congestion: High syn-ack delays
Link / Node failure: too many retransmissions
A Monitor Traffic
Network Traffic
Primary path
C Reroute Traffic
Backup path
Fast Reroute System

Fig. 1. Phases in Fast Reroute (FRR) systems.

- We design and develop an anomaly detection mechanism[1] that helps to determine whether the switch state is under the influence of adversarial traffic (§IV).
- We perform experiments using realistic traces and demonstrate the effectiveness of the proposed mechanism in detecting attacks on the two FRR systems (§V).

II. Background and Problem Statement

Table I shows FRR systems built on top of high-speed programmable data planes. The data plane maintains a state that represents traffic characteristics of various network conditions (*e.g.,* normal, link failure, high path delays, congestion). For instance, Blink's [10] data plane processes TCP packet headers to distinguish whether the incoming packet is original or retransmitted and updates the number of flows (state) experiencing retransmissions. Next, it periodically analyzes the count, infers network conditions, and reroutes traffic if necessary. In line with this FRR paradigm, RouteScout's [12] data plane maintains a time difference between TCP SYN-ACK headers (state) and computes the associated next hop's average delay. Based on the delays observed for each next hop, it adjusts traffic splitting among the hops.

In general, due to limited memory (100's of MBs [22]) and a limited set of per-packet operations allowed in the data plane [1], the design of FRR systems' packet-processing algorithms is constrained in several ways; hence, trust incoming packets without checking for discrepancies in their header values. For instance, some high-speed switch targets natively support unsecured hash functions like CRC32 using which FRR systems compute 5-tuple hash. Also, to minimize memory overheads, the FRR systems choose to monitor only a small subset of traffic (or flows) and use the state of the sampled traffic for making decisions. This is implemented by using consecutive monitoring ranges [23], where a flow is sampled only if its 5-tuple hash value falls within a specific range.

Due to the data plane constraints, the attack surface increases and makes FRR systems more vulnerable to adversarial traffic. More specifically, an attacker may send crafted flows with packet headers to manipulate the switch state and influence the decisions of FRR systems. We argue protecting the FRR systems from adversarial inputs is essential for them to be deployed widely.

Many recent works [23]–[28] argue the need for securing data plane systems against adversarial inputs. While these works identify vulnerabilities of diverse data-driven data plane systems and discuss the potential attacks on these systems, this paper focuses on vulnerabilities of FRR systems and detects whether the systems' are under the influence of adversaries.

[1]https://github.com/networked-systems-iith/FRR-Attacks

TABLE I
FastReRoute systems and their state in the data plane

Reroute decisions	FRR systems	State
Reroute traffic to backup paths	Blink [10], HULA [13]	**Blink**: The count of flows experiencing retransmissions, **HULA**: Per-path delay
Load balance or traffic splitting	RouteScout [12], RCP [14], SmartNIC-based Load Balancers [15], [16], SmartLB [17], Turbo [18], RingLeader [19]	**RouteScout**: per-flow SYN-ACK delay and per-flow packet losses, **RCP, XCP**: sending rate & RTT, **SmartNIC-based Load Balancers**: one-way packet delay

III. Attacks on FRR Systems

In this section, we define a threat model and present how attackers can exploit vulnerabilities of two FRR systems, Blink [10] and Routescout's Delay Monitor [12], and influence their fast reroute decisions.

A. Threat model

Attacker privileges. We consider an attacker at the host-level or the Man-in-the-Middle (MitM)-level [29], [30] with an ability to inject crafted packets into a network where FRR systems are deployed.

Attack target. The attacker aims to target the state maintained by FRR systems in the data plane to trick the FRR system and influence reroute decisions. As a consequence, it would result in poor QoS and DoS for legitimate users.

Attacker knowledge. We assume that the attacker possesses knowledge of the packet-processing behavior and the state maintained by the FRR systems, perhaps through their open-source implementations [31], [32].

B. Possible attacks

*1) **Blink: Fast connectivity entirely in the data plane**:* Blink [10] leverages TCP-induced signals to detect downstream link failures in the P4 data plane. Blink's flow selector logic samples up to 64 active TCP flows for a destination prefix and maintains them in a register called *flow selector*. A flow is considered active if the flow's packet is observed within a 2-second time window. It detects retransmission timeout timer (RTO) induced retransmissions by storing the sum of the sequence number and the payload length of the incoming packet for each sampled flow. If the number of sampled flows observing retransmissions exceeds a threshold (*e.g.,* 32 or 50% of sampled flows), Blink infers this as a link failure in the network and reroutes associated prefix traffic to any one of the available backup paths.

(A) Flow selector vulnerability. A TCP flow needs to be active for a longer duration such that Blink samples the flow;

Algorithm 1 Blink (A1) Fast-release

```
1: $F \leftarrow [F_1, F_2, ..., F_M]$ {$\|F\|$= a% of $\|N\|$ benign flows}
2: for each $f$ in $F$ do
3:     SendPacket($f$)
4: end for
```

Algorithm 2 Blink (A2) Gradual-release

```
1: $F \leftarrow [F_1, F_2, ..., F_M]$ {$\|F\|$= a% of $\|N\|$ benign flows}
2: $malicious\_flows \leftarrow [\ ]$
3: for every $t$ seconds do
4:     move $x$ flows from $F$ to $malicious\_flows$
5:     for each $f$ in $malicious\_flows$ do
6:         SendPacket($f$)
7:     end for
8: end for
```

an active flow consistently occupies an entry in the flow selector register until a *FIN* packet is detected or a hard reset timeout (8.5 mins) is triggered. If the sampled flows are malicious, they can pollute the switch state and eventually lead to incorrect reroute decisions. We consider this as a vulnerability because an attacker can inject malicious flows with an objective of making the majority of the flows sampled as malicious.

(B) Attack strategy. The attacker sends packets within a 2-second time window such that the associated flow is active and occupies an entry in the flow selector register. Subsequently, the attacker proceeds to imitate RTO-induced retransmissions by repetitively sending the same packet. The success criterion for the attack is defined as the percentage of sampled malicious flows observing retransmissions exceeding 50%.

Attacker flow injection: For injecting malicious active TCP flows, the attacker can opt for two strategies: (A1) fast-release approach and (A2) gradual-release approach. The attacker uses the A1 strategy for simultaneous transmission of all flows, taking a more aggressive approach. On the other hand, the A2 strategy is chosen when the attacker aims to be less aggressive, strategically evading detection and requiring less compute. In the fast-release approach, an attacker injects M packets one each from M malicious flows where M is equal to the $a\%$ of the number of benign flows (N) in a trace. This attack approach is summarized in Algorithm 1. In the gradual-release approach, the attacker introduces a slow buildup over time; instead of all M packets at the same time, the attacker introduces x malicious packets cumulatively out of M packets for every t seconds. The attacker's objective is to maximize impact with minimal resources, that is, the values of (a, x, t) should be small. This approach is summarized in Algorithm 2. For both A1 and A2, at least one packet of an injected flow is sent within 2 seconds so that the associated flow is active.

(C) Attack demonstration. We use real network traffic traces from CAIDA [21], specifically, the 2018 *dirA March equinix nyc* traces [33]. As Blink operates on a prefix basis, we filter the CAIDA trace based on top-100 /24 prefixes.

TABLE II
Attack details. NR = Normal, TS = Traffic Spikes

Prefix (/24)	**Network condition and % of flows experiencing retransmissions**	**Avg. pkts/ sec**	**# TCP flows**	**(a, x) values for A2**
64.5.155	NR (1.5625)	$65K$	$15K$	$(1\%, 3)$
251.96.218	NR (3.75)	$3K$	$8K$	$(1\%, 2)$
135.79.171	TS (6.25)	$12K$	$11K$	$(1\%, 2)$
205.112.118	TS (7.8125)	$4K$	$13K$	$(0.7\%, 2)$
198.231.53	TS (9.375)	$20K$	$6K$	$(0.4\%, 1)$
205.139.35	TS (10.9375)	$4K$	$2K$	$(0.4\%, 1)$
250.178.65	TS (12.5)	$2K$	$10K$	$(0.2\%, 1)$
205.164.190	TS (14.065)	$6K$	$9K$	$(0.2\%, 1)$

Attack dataset: Due to the lack of a dedicated attack dataset (traces), we generate realistic attack traces by interlacing attack traffic with original CAIDA traces so the traffic characteristics in a real network are preserved. We randomly select $a\%$ of benign flows in the CAIDA trace as attack flows. From the selected flows, we randomly pick packets and re-transmit them by replaying at a randomly selected packets-per-second rate, spanning from 0 to 2 seconds. This process ensures that the flow remains active with retransmissions for a duration of 60 seconds which is the total duration of a CAIDA trace. Finally, we interleave these malicious flows with the original (base) benign trace using A1 and A2 strategies for 8 different prefixes as illustrated in Table II.

As discussed earlier, we set t to 1 second and vary (a, x) based on the network conditions observed in the base benign trace. In our examination of over 500 prefixes in CAIDA traces [21], we categorize prefix traces based on the percentage of flows experiencing retransmissions. If the percentage falls within the range $(0 - 5]\%$, we consider such trace as normal (NR). Range $(5 - 15]\%$ represents traffic spikes (TS), that is more retransmissions due to either packet loss or congestion. Beyond 15% represents a potential indication of real link failure (more details in §IV-A).

Attack results: (1) **A1 strategy:** From the experiments, we observe attacker's A1 strategy is unsuccessful, even for larger values of a within the range of 2 to 32; a burst of $a\%$ malicious flows in a small interval would not pollute flow selector's register because Blink packet sampling rate is very low. (2) **A2 strategy:** On the other hand, using A2 strategy the attacker succeeds with only a few malicious flows. Moreover, this strategy is less aggressive and demands fewer computing resources from the attacker. As shown in Figure 2, we observe that for 64.5.155/24 prefix with only 1.5% of flows in the trace experiencing retransmissions (normal), the attacker is successful with $x = 3$ attack flows. That is, a total of 144 flows (close to 1% of benign flows in the trace) are sufficient to pollute the flow selector's register and successfully mislead the fast reroute decisions in less than 50 seconds. The key insight is that since malicious flows are active for a longer duration, they get sampled over a while and successfully pollute the flow selector's register. From the second column in Table II, as the percentage of flows experiencing retransmissions increases,

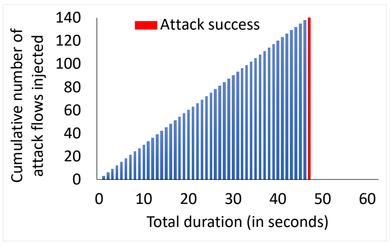


Fig. 2. Successful attack on Blink with A2 strategy for 64.5.155/24 prefix with a=1%

especially 9-14%, the attacker would need less effort (small a and x) to successfully mimic a link failure and trick the Blink system. This is because the network is already congested or experiencing more packet losses, thus it becomes easier for the attacker to trick the system by injecting less malicious traffic.

*2) **RouteScout: A performance-driven internet path selection entirely in the data plane:*** RouteScout [12] is a novel software-hardware co-design for performance-aware routing that runs at the edge of the network and independently controls the traffic paths. More specifically, RouteScout's delay monitor logic in the data plane computes the per-hop average delay for every epoch, and at the end of each epoch, if the percentage of change in average delay among the next hops is greater than some user-defined threshold, RouteScout changes the traffic splitting percentage among the next hops as defined in the network policy.

(A) Delay monitor vulnerability. For each next hop, RouteScout's delay monitor (DM) stores ingress timestamps of all SYN packets, and when an ACK corresponding to a SYN arrives, it calculates the difference between both timestamps and adds the delay to the running sum. At the end of each epoch, the per-hop average delay is calculated using the sum and the number of SYN-ACK pairs. It considers only those SYN-ACK pairs whose 5-tuple hash falls within the *monitoring subrange* of a designated next-hop. However, the per-hop state can be manipulated as the delay monitor logic cannot differentiate between authentic and malicious SYN-ACK pairs, potentially leading to performance degradation. **(B) Attack strategy.** An attacker sends SYN-ACK pairs with either very small or very high time gaps with the intent to pollute a specific next hop's average delay such that the percentage of change in an epoch between two next hops (say, A and B) exceeds the threshold (we consider 10%) [12]. More specifically, an attacker can use two approaches for crafting malicious flows: (A1) batch-release and (A2) gradual-release. In the batch-release approach, the attacker injects M malicious SYN-ACKs where M is equal to $a\%$ of the total benign SYN-ACKs (N) pairs in a real trace. All M malicious SYN-ACKs are injected at once for every t seconds. In the gradual-release approach, the attacker introduces x malicious SYN-ACKs from M for every t seconds. Let y be the time difference between crafted SYN and ACK. In A1, we use a constant y for all SYN-ACK pairs, whereas in A2, we vary y for each SYN-ACK pair. The strategies are summarised in

Algorithm 3 RouteScout's DM (A1) Batch-release

```
1: $F \leftarrow [F_1, F_2, ..., F_M]$ {$\|F\|$ = $a\%$ of $\|N\|$ benign SYN-ACKs}
2: for every $t$ second do
3:   for each $f$ in $F$ do
4:     SENDSYN($f$)
5:     WAIT($y$)
6:     SENDACK($f$)
7:   end for
8: end for
```

Algorithm 4 RouteScout's DM (A2) Gradual-release

```
1: $F \leftarrow [F_1, F_2, ..., F_M]$ {$\|F\|$ = $a\%$ of $\|N\|$ benign SYN-ACKs}
2: $y \leftarrow$ [RANDOM($min, max$) for each $F_i$]
3: $malicious\_pairs \leftarrow$ [ ]
4: for every $t$ second do
5:   move $x$ SYN-ACKS from $F$ to $malicious\_pairs$
6:   for each $f$ in $malicious\_pairs$ do
7:     SENDSYN($f$)
8:     WAIT($y$)
9:     SENDACK($f$)
10:  end for
11: end for
```

Algorithm 1 and 2, respectively. We assume that attacker is aware of the avg RTT for a prefix and adjusts a and y to maximize impact with less attack traffic.

(C) Attack demonstration. We use network traffic traces from CAIDA [21], [33], specifically, the 2018 dirB March equinix nyc CAIDA traces. RouteScoute operates on a prefix basis, so we filter the CAIDA trace based on top-10 /16 prefixes. *Attack dataset:* We interlace the crafted SYN-ACK pairs with the original benign trace using A1 and A2 strategies. We assume the attacker knows the target prefix's average SYN-ACK RTT either by monitoring the prefix's trace or by analyzing the public dataset. We vary y from 2 to 32 times the average RTT (*i.e.,* 2, 4, 8, 16, and 32). In A1, we fix y for all SYN-ACK pairs and iterated through the list. In A2, y is set to a value selected randomly between 2 to 32. We observe that the attack is not successful when y is small (close to 0), so we focus our discussion mainly on large y values.

Attack results: Table III show the results for A1 strategy. When $y = 2$, the attacker is successful for different prefixes with a value less than 2%. This means less than 2% of the benign SYN-ACK pairs is sufficient to influence the RouteScout splitting decisions. We observe as y increases from 2 to 32, the attacker is successful for smaller a values (2 to 0.5). This is because high SYN-ACK RTT delays (y) influence the average delay much faster. Using the A2 strategy, we observe that the attacker is successful when a is between 0.6 to 1.2% across different prefixes.

TABLE III
ATTACKING DELAY MONITOR USING A1 STRATEGY WITH $y = 2$

Prefix (/16)	**Avg. # SYN-ACKS per hop**	a **value**
213.175	$1.1K$	2%
213.173	$1.5K$	1.5%
239.213	$1.4K$	1.6%
239.213	$1K$	2%
213.179	$1.8K$	1.2%
49.97	$1.3K$	1.8%
41.246	$1.4K$	1.6%
213.187	$1.1K$	2%
76.177	$1.2K$	1.8%
213.184	$1.5K$	1.5%

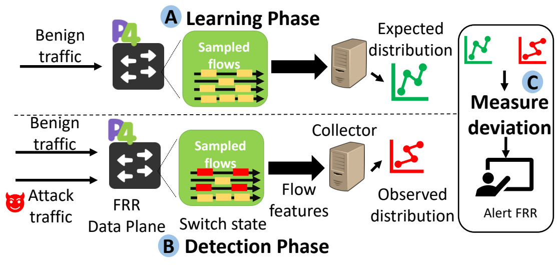


Fig. 3. Detection workflow

IV. Attack Detection

In this section, we present our anomaly detection approach to detect whether the state maintained by an FRR system in the data plane is manipulated by adversaries. We define objectives, pinpoint challenges, and then delve into our proposed methodology.

Objective. Our objective is to detect (at runtime) potential manipulation of the switch state by adversarial traffic before a routing decision is initiated by the FRR system at runtime. Doing so would enable the prevention of network performance degradation.

Overview. Figure 3 shows the workflow of our detection approach. It has three main phases:

- **Expected distribution.** We aim to gain a comprehensive understanding of various traffic patterns associated with benign network behavior. To do so, we collect benign traffic features under various normal scenarios (congestion, packet loss, and failures) and derive the expected distribution from the traffic features.
- **Observed distribution.** At runtime, the collector gathers traffic features with both benign and malicious flows and constructs observed distribution.
- **Measure deviation.** We measure the deviation between the expected distribution and the observed distribution using a chi-squared test [34]. If the deviation crosses a threshold, then alerts are generated.

Another alternative is to protect the switch state using crypto-based authentication and key-sharing schemes. However, incorporating crypto-based schemes [20], [30], [35] in the data plane is challenging in a high-speed programmable switch like Tofino [1] due to a limited set of operations with no support for loops and recursions which are in general necessary for cryptographic primitives. Moreover, creating, distributing, and managing secure keys between hosts and core networks has practical challenges [35].

TABLE IV

Benign traffic conditions	**% of flows experiencing retransmissions**	**# prefixes (out of 500)**
Normal	$0 - 5$	244
Traffic spikes	$5 - 15$	201
Failure	> 15	55

TABLE V
FLOW FEATURES EXPLORED FOR DETECTING ATTACKS

FRR	**Features**	**Set of flows**
Blink [10]	FD, FS, IPT	Sampled flows experiencing retransmissions.
RouteScout's delay monitor [12]	RTT_d	Sampled flows with SYN-ACK delay greater than average RTT.

A. *Benign traffic conditions*

To realize our approach, one challenge is that for a given switch state, it is hard to differentiate between whether it is manipulated by adversarial inputs or it represents an abnormal state under various benign network conditions (*i.e.,* link failure, congestion). To address this, we build expected behavior from *feature characteristics of benign flows* contributing to the switch state *under various benign network conditions.* This is based on the observation that at the switch level, the visibility of most normal traffic is not available to the attacker. Therefore, the characteristics of flow features derived from the majority of normal traffic would not only help to distinguish malicious behavior from normal but also make it hard for the attacker to trick the detection system due to the lack of visibility over the majority of normal traffic that the switch has. We consider three benign network conditions, categorized based on the percentage of flows experiencing retransmissions, as shown in Table IV. We process over 500 prefixes (/24) in CAIDA 2018 *March* traces [33] where each trace is of 60 seconds duration, and tag each prefix with normal, or traffic spike (due to congestion or packet losses), or link failure (too many flows experience retransmissions).

B. *Characteristics of flow features*

Due to limited data plane memory availability and strict hardware constraints, it is important to keep the data plane resource overheads low without compromising detection accuracy. We achieve this in two ways. First, we identify a small set of important per-flow features (*e.g.,* Flow duration (FD)[2], Flow size (FS)[3], SYN-ACK RTT[4], packet counts, inter-

[2]It is characterized as the time duration that a flow is sampled in the monitoring data structure of the respective FRR system (*i.e.*, flow selector in Blink, Accumulator in RouteScout).

[3]It is characterized as the number of bytes of a flow sampled by the respective FRR system.

[4]The time gap between the SYN and ACK in a TCP session's three-way handshake.

arrival time (IAT)[5]) to be collected at run time such that the characteristics of the selected flow features should accurately distinguish whether the switch state used by the FRR systems is manipulated by malicious flows or updated by benign flows under various network conditions. Second, instead of maintaining the selected features for all flows in the data plane memory, we maintain only for those flows considered by the respective FRR system. That is, for the Blink system, we collect features of a small number of flows (64) sampled by the flow selector. Similarly, for the Routescout system, we collect features of those flows that fall in the monitoring subrange. This way, we keep the switch data plane resource overheads low without impacting detection accuracy.

Table V shows the flow features that we explored for Blink and RouteScout systems. For distributions, we use both individual features and a metric derived from a feature. For instance, in RouteScout's delay monitor, we use a metric called RTT distance (RTT_d) which is the difference between the average per-hop SYN-ACK delay and individual flow's SYN-ACK delay. We observe FD for Blink and RTT_d for RouteScout's delay monitor are giving high detection accuracy (more details in §V-A).

In this paper, we identified the important flow features and associated metrics after manually analyzing the accuracy results. As part of our future work, we intend to automate the extraction of k-best features using machine learning and collect those features at runtime – this will enable the generalization of our detection mechanism for a wide range of FRR systems. As mentioned earlier, the important features are collected only for those flows monitored by the respective FRR systems that influence the switch's state. One could get these features by adding a feature extraction plug-in to the existing FRR system's P4 code and executing the modified code in the data plane (more details in §V).

C. Measure deviation

We build the expected distribution from the flow-level features (Table V) of benign traffic (Table IV) collected at regular intervals named epoch windows. For each epoch window, we build the expected distribution (E). The range of possible feature values (or metrics) is divided into n bins where each bin is of size k. For example, consider the flow duration (FD) feature. Here, bin_q denotes the flow duration falling within the range of q to $q * k + 1$ seconds. For a given window w_i, the expected distribution Ew_i is defined as:

$$Ew_i = (Ew_i(1), Ew_i(2), Ew_i(3), ..., Ew_i(n)) \tag{1}$$

where $Ew_i(1)$, $Ew_i(2)$, ..., $Ew_i(n)$ denote the frequencies (or count) corresponding to bins 1,2...n, respectively. Let Sw_i be the total number of flows sampled by an FRR system in the window, then:

$$Sw_i = \sum_{j=1}^{n} Ew_i(j) \tag{2}$$

[5]The time gap between two consecutive packets of a flow.

We collect Ew_i and Sw_i for multiple epoch windows and model probability for each bin to define the NULL hypothesis. We define the NULL hypothesis in terms of the set given by:

$$M = (p_1, p_2, p_3, ..., p_n) \tag{3}$$

where p_1, p_2, ..., p_n denotes the probability corresponding to bins 1,2...n. To be specific, if there are W windows, the probability corresponding to each bin is defined as,

$$p_j = \frac{\sum_{i=1}^{W} Ew_i(j)}{\sum_{i=1}^{W} S_i} \tag{4}$$

We use M as the expected distribution and use it as a reference to validate an observed distribution in the detection phase.

More specifically, at runtime, for each epoch window we collect features from the observed traffic which may or may not have malicious flows interlaced. Formally, the observed distribution Ow_i for the current window w_i is defined as:

$$Ow_i = (Ow_i(1), Ow_i(2), Ow_i(3), ..., Ow_i(n)) \tag{5}$$

and Sw_i be the total number of flows sampled by the FRR system in this window, given by:

$$Sw_i = \sum_{j=1}^{n} Ow_i(j) \tag{6}$$

We measure the deviation between the expected distribution and the observed distribution using a chi-squared test [34]. The chi-squared test for independence compares two frequency (M, and O) distributions to see if they are related. To be specific, we define χ^2_{wi} for the window w_i as:

$$\chi^2_{wi} = \sum_{j=1}^{n} \frac{(Ew_i(j) - Ow_i(j))^2}{Ew_i(j)} \tag{7}$$

where expected frequency of bin j, $Ew_i(j)$ is defined as:

$$Ew_i(j) = p_j * Sw_i \tag{8}$$

A small χ^2_{wi} denotes that the observed distribution fits well as expected. A large value indicates a potential deviation from the expected distributions. As the chi-square test is highly dependent on the sample size S, we also consider employing sample-specific probabilities. For each sample size range (r, R], we derive probabilities (p_r) specific to that range.

D. Implementation and other approaches.

In this paper, we detect anomalies by performing a statistical analysis of flow features at the control plane. This approach keeps the data plane overheads low but at the cost of data-plane and control-plane communication overhead and feature processing overheads. As a potential future work, one could analyze the features at line rate by adopting the quantization techniques [28] designed to run entirely in the data plane. Alternatively to the proposed statistical analysis-based approach, one could explore machine learning-based traffic analysis [36], [37] using Isolation forest, autoencoders, or support vector machines (SVMs). However, realizing ML-based approaches in the data plane becomes more challenging due to computing and memory constraints.

V. Evaluation

Our main goal for evaluation is to study the following two questions. **Q1:** Which flow features (or associated metrics) of an FRR system enable the detection of attacks accurately with minimal false positives? **Q2:** Is our detection approach able to detect attacks before the attacker is successful, that is, before rerouting traffic? We choose two systems, Blink and RouteScout, for analysis because their switch state is updated by two different traffic characteristics; Blink updates based on header field values, whereas RouteScout relies on packet-delay statistics. We could extend the proposed approach towards the generalization by evaluating other FRR systems that broadly depend on these two traffic characteristics.

A. Identification of flow features

As outlined in §IV-B and Table V, we analyze the accuracy results of various flow features for an FRR system and pick those giving the best accuracy. For the Blink system, we analyze flow duration feature for more than 500 (/24) prefixes and found that the majority of the flows experiencing retransmissions under normal network conditions have flow duration ranging between 8 and 12 seconds (as shown in Figure 4(a)). For the attack to be successful, the malicious flows must be active for a longer duration (more than 12 seconds) so that they get sampled and pollute the flow selector's state. This means a significant deviation in flow duration distribution of flows in the attack scenario and the normal scenarios. Based on this observation, we pick flow duration as a feature to collect at runtime and construct the expected distribution. We did not find such trends for other features (flow size and IPT). Hence, we chose to collect the flow duration (FD) feature only for those sampled flows experiencing retransmissions.

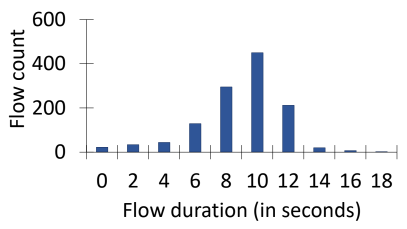
Similarly, for the Routescout system, we analyze SYN-ACK delays of top-20 /16 prefixes and found that flows with SYN-ACK RTT delay either greater or close to the average delay would significantly influence the average delay. As shown in Figure 4(b), 99.9% of flows monitored for next-hops A and B are less than the average delay, but with less influence on the average delay. However, the 0.1% flows that are above the average delay have more influence on the delay, thus the delays of these flows play a major role and cause the delay difference between a pair of next hops to exceed the threshold (i.e., 10%). To understand this influence, we define a derived feature named RTT distance (RTT_d) to capture the distance between a flow's SYN-ACK delay and the average delay. Figure 4(c) shows the distribution of the RTT_d feature for those flows with greater than average delay. Our key observation is that most of the 0.1% benign flows with SYN-ACK delay above average have RTT_d between 0 and 1 (falls in bin_0) *i.e.*, they are very close to the average. For an attack to succeed, the malicious flows should have bigger RTT_d, that is away from the average, starting from bin_1 and beyond. Based on this observation, we choose the RTT_d feature for the Routescout FRR system.
Dataset for expected distribution: As explained in (§IV-C), we compute probabilities (p) for each bin's associated flow feature (*i.e.*, FD for Blink and RTT_d for RouteScout).
Blink: We collect FD of sampled flows experiencing retransmissions for top-10 benign CAIDA prefix(/24) traces under failure category (Table IV). The data is collected for every 0.6 seconds (shorter than Blink's epoch window, 0.8 seconds). We observe that there are 110 windows with $> 15\%$ of flows experiencing transmissions. The FD feature range is divided into 10 bins, each of size 6 seconds. Each bin can be represented as: bin_i= $\{$FD falls between (i, (i*6)+1] seconds$\}$. p in eq. 3 is derived from the data for 110 windows.
RouteScout: For the RouteScout system, from the top-10 benign CAIDA prefix(/16) traces, we obtain 850 windows with average delay difference between a pair of next hops exceeding 10%. RTT_d value range is divided into 10 bins, each of size 1 second. Each bin is represented as: bin_i= $\{RTT_d$ from average delay falls in (i, i+1] seconds$\}$. We derive p for each of these bins using the data of 850 windows.
Testing dataset: We test the computed probabilities (p) using the dataset for two traffic scenarios: Case 1: Benign traffic without attacker flows; and Case 2: A small number of malicious flows interlaced with a large number of benign flows using the strategies outlined in §III. For case 1, the Blink dataset includes 28 prefix traces, with 8 falling under the normal category, 10 under traffic spikes, and 10 under failures. For case 2, we generated 8 traces. For Routescout, case 1 comprises 27 prefix traces, and case 2 has 8 base prefixes (as mentioned in Table III) which we interlaced malicious flows using A1 and A2 strategies totalling to 48 traces.

B. Detection accuracy

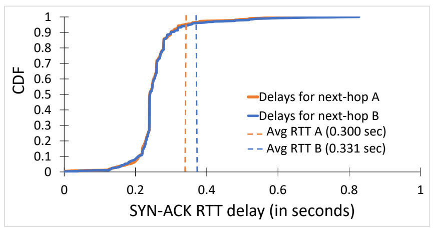
We measure accuracy in terms of false positive rate (FPR) and false negative rate (FNR). FPR represents the proportion of normal windows that are incorrectly predicted as attack, and FNR represents the proportion of attack windows that are incorrectly predicted as normal. To our knowledge, there is no other work that we should consider for comparing the accuracy of the proposed approach.

We emphasize that Blink's mechanism operates on a time-series basis, where feature value in a time window is cumulatively aggregated with its preceding consecutive windows. In contrast, Routescout does not depend on consecutive windows, as flows get reset periodically. Consequently, when labeling windows as either benign (0) or attack (1), we follow two different approaches one for each system. For *case 1*, in both Blink and Routescout datasets, all windows are labeled as benign (0). However, for case 2 in Blink, windows are labeled as benign (0) when the flow selector's register has no attack flows, and as an attack (1) if malicious flows are sampled until the attack is successful. For case 2 in Routescout, benign (0) is assigned if the attack is unsuccessful, and attack (1) if the attack is successful. The chi-square threshold is derived using the standard chi-square distribution table [34].

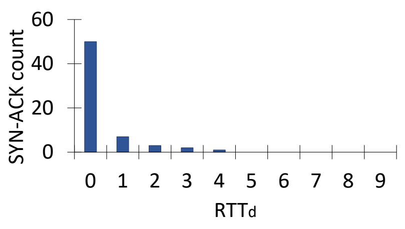
1) Blink results: The distributions for benign traffic are illustrated in Figure 5, where Figure 5(a) represents observations for the normal category, Figure 5(b) for traffic spikes, and Figure 5(c) for failures. The X-axis shows epoch windows



(a) Most of the normal flows with retransmissions have flow duration between 8-12 sec.

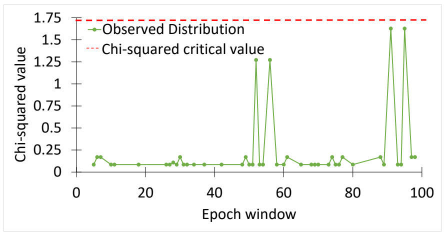


(b) CDF of SYN-ACK delays for a prefix with $>$ 10% difference in average delays of hop A and B.

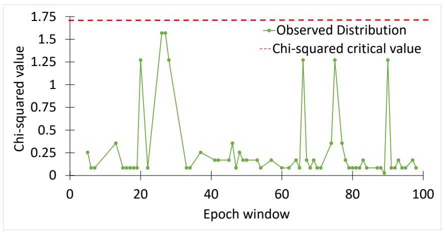


(c) RTT_d distribution for flows above average delay

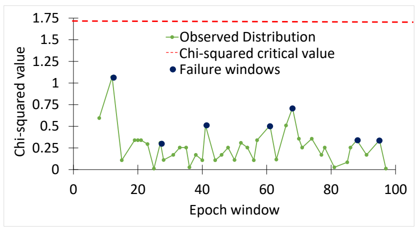
Fig. 4. Benign traffic analysis for identifying important flow features



(a) Normal

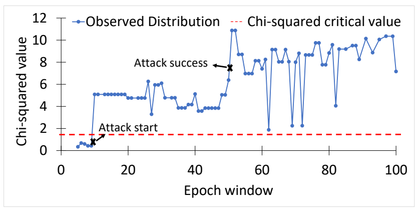


(b) Traffic spikes

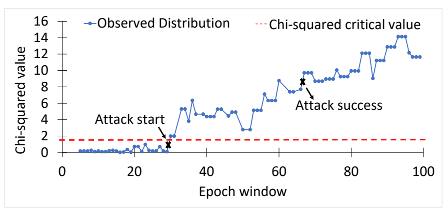


(c) Link failures

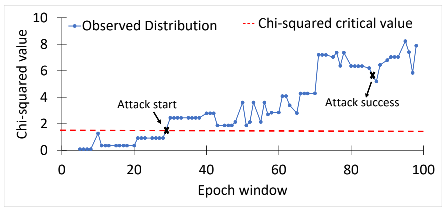
Fig. 5. Blink: The chi-square value for the observed traffic under various benign network conditions is below the threshold. The windows with traffic spikes and link failures are classified as benign.



(a) Normal and attack traffic



(b) Traffic spikes and attack traffic



(c) Link failure and attack traffic

Fig. 6. Blink: The chi-square value for the observed traffic exceeds the chi-square threshold before the attack is successful.

each of 0.6 seconds, and the Y-axis represents the χ^2 value of each window. Notably, χ^2_{wi} of the observed traffic aligns well when attack traffic is not present, consistently falling below the standard threshold values [34]. In case 2, as shown in Figure 6, deviations in χ^2_{wi} from the threshold become apparent as the attack initiates, signifying the ability to distinguish attacks from various benign network conditions. From the Figure 6, it is evident that we detect attacks at an early stage before the attacker is successful (Q2). Furthermore, in case 1, among the 2800 benign (0) windows, we observe an FPR of 0.003. In case 2, among 200 (0) and 600 (1) windows, an FPR of 0.001 and an FNR of 0 are observed, respectively.

2) RouteScout results: For case 1, among the 1456 benign windows, we observe an FPR of 0.006. For case 2, chi-squared test considering all 10 bins, initially we observe high FNR at lower delays (*i.e.*, 2 times and 4 times the avg. RTT for A1), although we get 0 FNR for higher delays (8, 16 and 32 times the avg. RTT for both A1 and A2 strategy). This is because attack SYN-ACK pairs have delays very close to the average (bin_0), mimicking normal behavior and thus hard to distinguish. To address this challenge, we further subdivide bin_0 into sub-bins and learn expected probabilities. This extra step results in an FNR of 0 for lower delay values (Table VI).

TABLE VI
ROUTESCOUT RESULTS WITH MODIFIED CHI-SQUARED TEST

Attack strategy	**Delay set to y times avg. SYN-ACK RTT**	**$a\%$ range: (a_min, a_max)**	**Misclassification rate**	
			FPR	**FNR**
A1	2	$(1.2, 2)$	0.01	0
	4	$(1.5, 1.8)$	0.01	0
	8	$(1.2, 1.5)$	0	0
	16	$(0.9, 1.1)$	0	0
	32	$(0.3, 0.8)$	0	0
A2	[2 to 32]	$(0.6, 1.2)$	0.01	0

VI. RELATED WORK

Prior work [23]–[27], [38] motivates the need for securing data plane systems from adversarial inputs. This paper complements these efforts by focusing on possible attacks on Fast Reroute data plane systems. A recent work [39], [40] does adversarial testing by proposing a probabilistic program profiler based on symbolic execution and model counting. In contrast, we model the probability distribution of flow-level features, enabling deviation checks at runtime. Our approach complements such offline profilers, as we can explore potential corner cases and flow features at run-time not covered by offline profilers. A line of research work [41]–[43] focuses on verifying P4 program properties using static analysis or symbolic execution techniques. In constrast, we focus on the detection of attacks on P4-based FRR systems at runtime. Our anomaly detection approach is inspired from [26]–[28], [44]. Increasingly, the focus has shifted to collecting flow feature statistics entirely in the data plane. These efforts are directed towards enhancing anomaly detection capabilities using statistical and machine learning techniques.

VII. Conclusion and future work

In this paper, we demonstrate attacks on two FRR systems using realistic traces. We propose a mechanism to detect such attacks on fast reroute systems. Through experiments using realistic traces, we show the effectiveness of the proposed mechanism in detecting attacks. In our future work, we plan to (1) generalize by automatically identifying important top-k features specific to an FRR system; (2) detect attacks entirely in the data plane such that the feature collection and processing overheads in the control plane can be minimized; (3) extend and evaluate our detection mechanism against other FRR systems; and (4) implement the detection mechanism on a programmable network device and study the overheads.

Acknowledgement

This work is supported by the National Security Council Secretariat, India, TiHAN IIT Hyderabad, and the Prime Minister's Research Fellowship program, India.

References

[1] Open-Tofino. [Online]. Available: https://github.com/barefootnetworks/Open-Tofino/tree/master

[2] Netronome Agilio CX SmartNICs. [Online]. Available: https://www.netronome.com/products/agilio-cx/

[3] NVIDIA BlueField DPUs. [Online]. Available: https://www.nvidia.com/en-in/networking/products/data-processing-unit/

[4] P4-16 Language Specification. [Online]. Available: https://p4.org/p4-spec/docs/P4-16-v1.2.2.html

[5] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing key-value stores with fast in-network caching," in *ACM SOSP*, 2017.

[6] G. Li, M. Zhang, C. Liu, X. Kong, A. Chen, G. Gu, and H. Duan, "Nethcf: Enabling line-rate and adaptive spoofed ip traffic filtering," in *IEEE ICNP*, 2019.

[7] J. Xing, Q. Kang, and A. Chen, "Netwarden: Mitigating network covert channels while preserving performance," in *USENIX Security*, 2020.

[8] K.-F. Hsu, R. Beckett, A. Chen, J. Rexford, and D. Walker, "Contra: A programmable system for performance-aware routing," in *USENIX NSDI*, 2020.

[9] Y. Li, R. Miao, C. Kim, and M. Yu, "Flowradar: A better netflow for data centers," in *USENIX NSDI*, 2016.

[10] T. Holterbach, E. C. Molero, M. Apostolaki, A. Dainotti, S. Vissicchio, and L. Vanbever, "Blink: Fast connectivity recovery entirely in the data plane," in *USENIX NSDI*, 2019.

[11] H. Birge-Lee, M. Apostolaki, and J. Rexford, "It takes two to tango: cooperative edge-to-edge routing," in *ACM Workshop on HotNets*, 2022.

[12] M. Apostolaki, A. Singla, and L. Vanbever, "Performance-driven internet path selection," in *ACM SIGCOMM SOSR*, 2021.

[13] N. Katta, M. Hira, C. Kim, A. Sivaraman, and J. Rexford, "Hula: Scalable load balancing using programmable data planes," in *ACM SOSR*, 2016.

[14] N. K. Sharma, A. Kaufmann, T. Anderson, A. Krishnamurthy, J. Nelson, and S. Peter, "Evaluating the power of flexible packet processing for network resource allocation," in *USENIX NSDI*, 2017.

[15] K. Holzinger, F. Biersack, H. Stubbe, A. G. Mariño, A. Kane, F. Fons, Z. Haigang, T. Wild, A. Herkersdorf, and G. Carle, "Smartnic-based load management and network health monitoring for time sensitive applications," in *IEEE/IFIP NOMS*, 2022.

[16] T. Cui, W. Zhang, K. Zhang, and A. Krishnamurthy, "Offloading load balancers onto smartnics," in *ACM SIGOPS*, 2021.

[17] Z. Ni, C. Wei, T. Wood, and N. Choi, "A smartnic-based load balancing and auto scaling framework for middlebox edge server," in *IEEE NFV-SDN*, 2021.

[18] H. Seyedroudbari, S. Vanavasam, and A. Daglis, "Turbo: Smartnic-enabled dynamic load balancing of μs-scale rpcs," in *IEEE HPCA*, 2023.

[19] J. Lin, A. Cardoza, T. Khan, Y. Ro, B. E. Stephens, H. Wassel, and A. Akella, "{RingLeader}: Efficiently offloading {Intra-Server} orchestration to {NICs}," in *USENIX NSDI*, 2023.

[20] S. Yoo and X. Chen, "Secure keyed hashing on programmable switches," in *ACM SIGCOMM Workshop on SPIN*, 2021.

[21] CAIDA Macroscopic Internet Topology Data Kit. [Online]. Available: https://www.caida.org/data/internet-topology-data-kit

[22] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics," in *ACM SIGCOMM*, 2017.

[23] L. Wang, P. Mittal, and J. Rexford, "Data-plane security applications in adversarial settings," *ACM SIGCOMM Computer Communication Review*, 2022.

[24] R. Meier, T. Holterbach, S. Keck, M. Stähli, V. Lenders, A. Singla, and L. Vanbever, "(self) driving under the influence: Intoxicating adversarial network inputs," in *ACM Workshop on HotNets*, 2019.

[25] H. SA, K. S. Kumar, A. Majee, A. Bedarakota, P. Tammana, P. G. Kannan, and R. Shah, "In-network probabilistic monitoring primitives under the influence of adversarial network inputs," in *APNet*, 2023.

[26] A. Sanghi, K. P. Kadiyala, P. Tammana, and S. Joshi, "Anomaly detection in data plane systems using packet execution paths," in *ACM SIGCOMM workshop on SPIN*, 2021.

[27] S. Gao, M. Handley, and S. Vissicchio, "Stats 101 in p4: towards in-switch anomaly detection," in *ACM workshop on HotNets*, 2021.

[28] D. Barradas, N. Santos, L. Rodrigues, S. Signorello, F. M. Ramos, and A. Madeira, "Flowlens: Enabling efficient flow classification for ml-based network security applications." in *NDSS*, 2021.

[29] H. Liu, X. Chen, Y. Shen, Q. Huang, Z. Zhou, D. Zhang, and C. Wu, "Vulnerabilities and attacks of inter-device coordination in programmable networks," in *2023 IEEE/ACM IWQoS*, 2023.

[30] D. Kong, Z. Zhou, Y. Shen, X. Chen, Q. Cheng, D. Zhang, and C. Wu, "In-band network telemetry manipulation attacks and countermeasures in programmable networks," in *IEEE/ACM IWQoS*, 2023.

[31] Blink's github repository. [Online]. Available: https://github.com/nsg-ethz/Blink

[32] RingLeader's github repository. [Online]. Available: https://github.com/utnslab/RingleaderNIC

[33] CAIDA Datasets. [Online]. Available: https://www.caida.org/catalog/datasets/passive_dataset/

[34] (2023) Chi-squared test. [Online]. Available: https://en.wikipedia.org/wiki/Chi-squared_test

[35] I. Oliveira, E. Neto, R. Immich, R. Fontes, A. Neto, F. Rodriguez, and C. E. Rothenberg, "Dh-aes-p4: on-premise encryption and in-band key-exchange in p4 fully programmable data planes," in *IEEE NFV-SDN*, 2021.

[36] G. Zhou, Z. Liu, C. Fu, Q. Li, and K. Xu, "An efficient design of intelligent network data plane," in *USENIX Security*, 2023.

[37] A. T.-J. Akem, B. Bütün, M. Gucciardo, M. Fiore *et al.*, "Jewel: Resource-efficient joint packet and flow level inference in programmable switches," in *IEEE INFOCOM*, 2024.

[38] C. Black and S. Scott-Hayward, "Adversarial exploitation of p4 data planes," in *2021 IFIP/IEEE IM*, 2021.

[39] Q. Kang, J. Xing, and A. Chen, "Automated attack discovery in data plane systems." in *USENIX Security*, 2019.

[40] Q. Kang, J. Xing, Y. Qiu, and A. Chen, "Probabilistic profiling of stateful data planes for adversarial testing," in *ACM ASPLOS*, 2021.

[41] J. Liu, W. Hallahan, C. Schlesinger, M. Sharif, J. Lee, R. Soulé, H. Wang, C. Caşcaval, N. McKeown, and N. Foster, "P4v: Practical verification for programmable data planes," in *ACM SIGCOMM*, 2018.

[42] A. Nötzli, J. Khan, A. Fingerhut, C. Barrett, and P. Athanas, "P4pktgen: Automated test case generation for p4 programs," in *ACM SOSR*, 2018.

[43] R. Stoenescu, D. Dumitrescu, M. Popovici, L. Negreanu, and C. Raiciu, "Debugging p4 programs with vera," in *ACM SIGCOMM*, 2018.

[44] G. Zhou, Z. Liu, C. Fu, Q. Li, and K. Xu, "An efficient design of intelligent network data plane," in *USENIX Security*, 2023.