TitanLog: Hierarchical and Elastic Logging for High-speed Network Data Stream

Yuanpeng Li*†, Xian Niu‡, Yikai Zhao*, Tong Yang*†, Yannan Hu†, Yuchao Zhang‡,
Xiangwei Deng*, Qiuheng Yin*, Ziyun Zhang*, Ruwen Zhang*, Yisen Hong§,
Kaicheng Yang*, Ruijie Miao*, Kun Meng$^{\P\|}$, Dahui Wang$^{\|}$, Yong Cui§†

***Abstract*—Logging network traffic plays a crucial role as it serves as the foundation for various network applications. As network scale continues to expand, contemporary network traffic becomes increasingly high-speed, high-volume, and dynamic. This growth poses challenges to traditional server-based solutions. In this paper, we propose TitanLog, a *hierarchical* and *elastic* logging system designed specifically for large-scale network traffic. TitanLog utilizes the hierarchical logging methodology, which aims to identify the importance of each packet in real-time and log packet data of different importance at different levels. To enhance efficiency, we propose a co-design of the emerging programmable switch and the server, incorporating sketches and RDMA to boost performance. To achieve elasticity, we design mechanisms for run-time adjustments and monitoring for resource insufficiency. TitanLog possesses the capability to switch between these modes at run-time. We fully implement TitanLog on a testbed and conduct extensive evaluations. The experimental results demonstrate that TitanLog supports logging of 100Gbps traffic with a zero packet loss rate and reduces the log volume by up to 96.28%.**

***Keywords*—Traffic logging, programmable switch, RDMA.**

I. Introduction

Traffic logs are essential for various network applications, including intrusion detection [1], [2], traffic replay [3], [4], and traffic analysis and prediction [5], [6]. These logs play a crucial role in enhancing network security and optimizing network performance. For instance, intrusion detection systems (IDS) rely on logged traffic data to refine their models, which helps in accurately identifying potential threats and vulnerabilities within the network. Similarly, traffic replay is useful for restoring and simulating network events, which aids in adjusting and improving network modules. Traffic analysis and prediction are important for understanding traffic characteristics like flow sizes and round-trip time (RTT).

Despite the critical role of traffic logs, recording all network traffic presents challenges. As networks grow in scale, they exhibit characteristics of *high-speed*, *high-volume*, and *dynamic* traffic, which can create significant bottlenecks in packet processing speed and storage capacity. Given these challenges, an ideal traffic logging system should record all useful traffic data rather than simply capturing everything. To achieve this, we argue that an effective logging system should be 1) *efficient*: the system must handle high-speed, high-volume traffic efficiently; and 2) *elastic*: the system should automatically adapt to changes in engineers' requirements and the dynamics of network environments.

Existing traffic logging solutions focus primarily on efficiency. They can be classified into two categories. The first category [7], [8], [9] speeds up the processing of each packet by minimizing copying and disk accesses. While these methods improve processing speeds, they do not adequately address the issue of storage volume, and they still fall short of keeping up with the rapid flow rates of current high-speed traffic. The second category [10], [11] aims to reduce the time and storage costs by truncating or sampling the packet data. However, these approaches may compromise the availability of necessary data for some applications, thus failing to meet the elasticity requirement.

In response to these limitations, we propose **TitanLog**, a *hierarchical* and *elastic* traffic logging system specifically designed to manage high-speed, high-volume, and dynamic network traffic. The key methodology of TitanLog is **hierarchical logging**, whose basic idea is to identify the category of each packet online and to apply variable levels of data compression based on the packet's significance. Hierarchical logging is based on the following observation: most applications only require a small portion of each packet or a small portion of the packets. For instance, traffic replay typically relies on packet header data, while IDS may need payloads of anomalous packets. Therefore, by focusing on the essential data required by applications, hierarchical logging reduces the unrealistic need to log all traffic and instead targets the necessary data for effective operation. For example, given that anomalous traffic typically only constitutes a small fraction of the total (e.g., around 1% [12]), and packet headers generally represent a small portion of the total packet size (e.g., about 5% [13], [14]), hierarchical logging can significantly decrease log volume by fully logging packets identified as anomalous and logging only the headers for benign traffic.

To realize hierarchical logging, we explore several implementation approaches. Pure software approaches are highly flexible but lack the throughput necessary to handle high-

*State Key Laboratory of Multimedia Information Processing, School of Computer Science, Peking University, Beijing, China.
†Zhongguancun Laboratory, Beijing, China.
‡School of Computer Science (National Pilot Software Engineering School), Beijing University of Posts and Telecommunications, Beijing, China.
§Department of Computer Science and Technology, Tsinghua University, Beijing, China.
¶College of Computer Science, Beijing Information Science and Technology University, Beijing, China.
$^{\|}$Guangdong Province LuiSuan Tech Co., Ltd., China.
Tong Yang (yangtong@pku.edu.cn), Yannan Hu (huyn@zgclab.edu.cn), and Yuchao Zhang (yczhang@bupt.edu.cn) are the corresponding authors.

speed network traffic efficiently. On the other hand, pure hardware approaches can manage high throughput but offer limited flexibility, which is crucial for adapting to dynamic network conditions and varying requirements. An emerging approach that balances these needs involves programmable switches, which provide high throughput alongside a degree of flexibility. However, programmable switches typically lack disk storage capabilities. Given these considerations, we opt for a switch-server co-design approach to implement TitanLog. This hybrid strategy leverages the high-throughput capabilities of programmable switches for the initial classification of packets – a process that requires rapid processing to keep pace with incoming traffic flow. The classified data is then managed through a combination of switch and server operations. This setup not only supports the high-speed requirements but also incorporates the flexibility needed to adapt storage and processing based on current network demands and the specific data relevance determined by hierarchical logging.

TitanLog is functionally segmented into three modules that collectively enhance both the efficiency and elasticity of the traffic logging process.

- **Packet classifier** employs a random forest (RF) within the switch data plane, enabling rapid and efficient packet classification. To enhance the feature extraction capabilities traditionally limited in hardware-based solutions, TitanLog incorporates sketches, a class of compact data structures that are adept at capturing detailed flow features with high accuracy while using minimal resources.
- **Packet data recorder** deploys RDMA (Remote Direct Memory Access) protocol on the programmable switch to enable direct and rapid data transfer to the server, bypassing traditional network stack bottlenecks. Additionally, a batch-based storage mechanism is designed to optimize the logging process, significantly reducing the frequency of disk accesses, thus enhancing overall storage performance.
- **Adaptive logging manager** ensures that TitanLog remains responsive to varying application demands and dynamic network conditions. This module offers multiple logging granularities and modes, enabling TitanLog to adapt dynamically to changes without requiring re-compilation. For instance, in scenarios where disk space is at a premium, TitanLog can switch to log-sampled mode, which logs only a subset of benign packets at a reduced granularity, thus conservatively managing disk usage. TitanLog also employs feedback mechanisms to monitor resource utilization and dynamically adjust logging modes based on current conditions to prevent data loss and ensure efficient resource use.

We fully implement TitanLog on our testbed and conduct extensive evaluations to assess its performance. The experimental results demonstrate that TitanLog reduces the volume of traffic logs by up to 96.28%, when handling high-speed traffic at 100Gbps.

In summary, we make the following contributions in this paper:

- **We propose the methodology of hierarchical logging.** We observe that most applications pay attention to a small portion of each packet or a small portion of packets. Based on this observation, we propose hierarchical logging, whose basic idea is to log packets of different categories at different granularities.
- **We present TitanLog to realize hierarchical logging.** We present TitanLog as a concrete implementation of hierarchical logging. We use the emerging programmable switch and the RDMA protocol to address the challenge in time overhead.
- **We design mechanisms to make TitanLog elastic.** We integrate multiple logging granularities and modes to ensure that TitanLog remains adaptable to diverse network environments and hardware capabilities. We also develop mechanisms to detect resource insufficiency, allowing TitanLog to automatically adjust to traffic dynamics.
- **We conduct real implementation and extensive evaluations.** We fully implement TitanLog on our testbed and conduct evaluations on it. The experimental results show that TitanLog can manage traffic at 100Gbps while reducing logging volume significantly. All related code is provided open-source at GitHub [1].

II. Background and Motivation

In this section, we first illustrate the requirements in traffic logging systems. Then we explain why hierarchical logging is needed for satisfying the requirements.

A. Requirements

We outline the requirements of TitanLog as follows.

- **[R1] Efficiency.** The system should handle high-speed, high-volume network traffic efficiently, addressing bottlenecks in both time and space dimensions. In the time dimension, traditional software-based solutions struggle with the speed of today's network traffic. The packet processing speed of a single-core CPU is typically in the range of Mpps (million packets per second), whereas today's network traffic often reaches levels of 100Mpps or even Gpps (giga packets per second). To handle such high-speed traffic, a server with many CPU cores or a cluster is required, significantly increasing the cost of logging. In the space dimension, the volume of network traffic can far exceed the capacity of disk storage. For instance, a 100TB disk can only store about 2 hours of 100Gbps traffic. The system should ensure that only relevant data is stored, making it both effective and economical.
- **[R2] Elasticity.** The system should be elastic to adapt to the changing demands of engineers and the dynamics of network environments. It should quickly respond to these changes, satisfying all specified requirements while addressing variations in network traffic that occur over time. For instance, traffic typically increases during the day and decreases at night [15], [16], and unforeseen events such as DoS attacks can cause sudden bursts in traffic, significantly increasing the number of incoming packets within a short period. The ability to sense and respond to such dynamics in real-time is crucial for maintaining the relevance and effectiveness of the logging system.

[1] https://github.com/TitanLog/TitanLog

B. The Need for Hierarchical Logging

TitanLog utilizes a classification-based **hierarchical logging** approach, which classifies network packets based on the requirements of different network applications, and logs each category at its corresponding granularity. We make several observations regarding network traffic:

1) Although network attacks can significantly degrade network quality, attack traffic comprises a much smaller proportion of packets than benign traffic. Miao *et al.* study attack characteristics in cloud networks, revealing that the median attack throughput is merely about 1% of the total traffic. Furthermore, both inbound and outbound attacks typically have short durations, with a median value of less than 10 minutes [12].
2) Packet headers constitute only a minor fraction of the entire packet data. The average packet size is around 700B to 1KB [13], [14], with a TCP/IP header taking up 40B, or 4% to 6% of the total packet length.
3) Most applications pay attention to a small fraction of packet data, such as packet headers or anomalous flows. For instance, application identification and traffic replay focus on logging packet headers; IDS necessitates logging both headers and payloads of (suspected) attack packets. Assuming anomalous packets constitute 1% of all packets and packet headers account for 5% of the packet length, the combined data from anomalous packets and headers of benign packets amounts to just 5.95% of all packet data.

In light of these observations, we propose **hierarchical logging**. Based on the application demands, we divide the packets into multiple categories, and log them at differentiated levels based on their respective categories. The logging level of the packets is the intersection of the data needs of all applications. Take two applications – intrusion detection and traffic replay – as an example. In this scenario, the packets are classified into two categories: anomalous packets and benign packets. The system maintains full logs – including headers and payloads – for anomalous packets, while it only logs packet headers for benign traffic. Hierarchical logging can also be realized with other logging levels. We will discuss them in Section VI-A.

Hierarchical logging predominantly addresses space overhead. However, implementing it directly on a CPU server still faces the time bottleneck. Therefore, we opt to implement hierarchical logging with a switch-server co-design, offloading high-speed operations to the programmable switch. We choose programmable switches rather than programmable NICs or NetFPGAs as the logging location primarily due to their significantly higher throughput and lower cost [17], [18], which are achieved at the expense of some programming flexibility.

III. TitanLog Overview

In this section, we first discuss the challenges in implementing TitanLog and our design choices. Then we overview the workflow of TitanLog.

A. Challenges and Design Choices

Challenge 1: Hardware limitations of the data plane. To maintain high speed, the data plane does not support complex operations like division, floating-point arithmetic, and loops. This presents difficulties in deploying most classification models. Moreover, the memory in the switch data plane is typically limited (e.g., $\sim$10MB). Given the high rate of network traffic and the sheer volume of concurrent flows (e.g., 100k), hash collision is inevitable, leading to data loss and large error.

Design Choice 1: Deploying a random forest (RF) and sketches for in-network classification. To overcome the aforementioned bottlenecks, we choose to use an RF as the classifier. The structure of the RF well fits the pipeline architecture of the programmable switch: each layer in the RF maps to one stage in the switch data plane. For feature extraction, we observe that small error is acceptable. It is impossible to achieve full accuracy during classification: 1) Even the most sophisticated models cannot guarantee full accuracy. 2) Initial packets of a flow offer limited information. For example, determining a flow's total length based solely on its first few packets is impossible. To this end, we advocate the use of sketches, a class of compact data structures perfectly suited for deployment on programmable switches. These allow us to extract features with small error. We classify commonly-used per-flow features into 4 categories, and design different sketches based on their attributes. We also present a novel approach to maintain and estimate average features.

Challenge 2: Slow per-packet storage with CPU. Storing each arriving packet directly to the storage system via CPU presents two main limitations. First, it demands per-packet processing by the CPU. As highlighted in challenge 1, this creates a throughput bottleneck. Second, it leads to frequent disk accesses. Writing multiple small files to the disk is much slower than writing a single large file of the same size, because the former introduces excessive file operations.

Design Choice 2: Deploying RDMA protocol in the programmable switch. RDMA stands as a rising technique in the network and database fields. By adopting RDMA, we can sidestep CPU involvement when writing data to server memory. Therefore, we design the packet data storage in two steps: 1) writing the packet data to server memory using RDMA; 2) storing the data to the storage system in batches. In the first step, we choose to implement the RoCE (RDMA over Converged Ethernet) protocol [19], [20] on the switch to eliminate the need for RDMA-specific hardware. We implement only essential functions for RDMA write messages on the data plane. For other functions (e.g., establishing RDMA connections), we implement them on the switch control plane. Given the limitation that current programmable switches cannot parse variable-length data, we design a packet splitting mechanism. This splits packets into segments of appropriate length. In the second step, we design a switch-side triggering mechanism to store the data to the storage system. This design helps circumvent potential memory scanning operations that might occur with server-side triggering mechanisms. Throughout the logging process, the CPU is mainly active when writing data to disks.

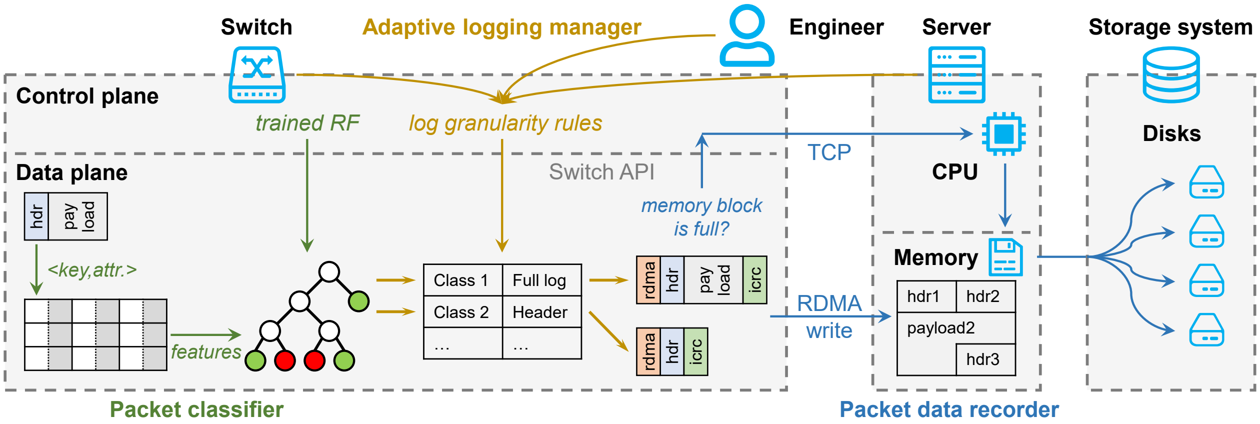


Figure 1. TitanLog overview and workflow.

Challenge 3: Complexity and risk in compile-time updates. Existing work [21] points out that deploying and updating modules at compile-time is complex and time-consuming. Compiling and refreshing often result in downtime and packet loss. To avoid this potential risk, operators must perform a series of operations, such as traffic draining and rerouting.

Design choice 3: Presetting log granularities and modes for run-time updates. TitanLog presets multiple log granularities. It uses a match-action table to map each type of packet to a granularity. The granularity mapping can be changed without shutting down the system by modifying table entries during runtime. We also analyze potential resource bottlenecks in TitanLog and propose methods to sense and address them when the network environment changes.

B. Workflow

TitanLog is designed to be deployed in a bypass configuration at the network ingress, meaning that it uses dedicated network devices and links for logging purposes, without sharing bandwidth or link capacity with production traffic. Incoming traffic is mirrored via a commercial switch, with the mirrored packets being forwarded to TitanLog. As shown in Figure 1, TitanLog is architecturally composed of three main components: a programmable switch, a lightweight relay server equipped with reduced CPU and memory resources, and a storage system equipped with multiple remote disks. TitanLog operates across three modules: the packet classifier, packet data recorder, and adaptive logging manager. TitanLog requires operators to provide two key inputs: *1) Classifier model.* A pre-trained RF loaded into the control plane to populate the data plane's packet classifier logic. *2) Logging policy.* Rules specifying traffic class-to-logging granularity mappings, managed by the adaptive logging manager and updatable at runtime.

Packet classifier. This module operates on the data plane of the programmable switch. TitanLog employs sketches to extract flow-level features and utilizes an RF for in-network packet classification. The RF is initially trained on the server and then coded into switch data plane. For each feature that the RF necessitates, a unique sketch is designed and deployed based on the attribute of the features. To ensure the accuracy of the sketches, TitanLog extracts the features within a constant time window (e.g., 10ms), i.e., the sketches are cleared after every time window. For each arriving packet, TitanLog inserts it into sketches sequentially, queries for approximate features, and records the results in metadata. Then TitanLog classifies the packet with the RF, determines its logging level, and records the outcome within the metadata.

Packet data recorder. This module involves collaboration between the programmable switch, the relay server, and the storage system. TitanLog establishes an RDMA connection between the switch control plane and the server, with only essential metadata retained on the data plane. When transmitting a packet, the switch data plane encapsulates it as an RDMA write message and sends it to the server, which writes the data to a memory buffer in sequence. Due to programmable switches' limitations in parsing variable-length payloads and appending fields at arbitrary positions, TitanLog leverages a packet splitting mechanism, fragmenting packets into fixed-size segments suitable for RDMA transmission. Additionally, TitanLog employs a switch-side triggering mechanism to manage data storage: it monitors the total length of transmitted data, and if this exceeds a preset threshold, indicating a nearly full memory buffer, it writes the buffered data to disk, clears the buffer, and resets the total length counter.

Adaptive logging manager. This module is responsive to changes in logging requirements from engineers and resource insufficiency detected automatically. TitanLog presets various log granularities and modes on switch data plane. When engineers adjust logging requirements, the control plane updates the logging rules via the switch API. TitanLog continuously monitors disk capacity, mirror bandwidth, and RDMA packet loss on both the server and switch. It automatically issues alerts when resources are insufficient. Following an alert, TitanLog adjusts the logging rules as needed and notifies these changes to the engineers.

IV. Packet Classifier

In this section, we introduce the first module of TitanLog, the packet classifier. We start by presenting our sketch-based solution for feature extraction, followed by the approach for deploying random forests (RF) in the switch data plane.

A. Features Extraction

Feature extraction is the basis of in-network classification. For each arriving packet, its features can be classified into two categories. *1) Packet-level features:* referring to the features carried by each individual single packet, e.g., `source/destination port`, `protocol`, `TCP flags`, `packet length`, etc. The packet-level features can be simply extracted by parsing the packet header. *2) Flow-level features:* referring to the statistical features of the flow to which the packet belongs, e.g., `packet number`, `average packet length`, etc. Flow-level features require maintenance on the switch.

As shown in Table I, the flow-level features can be classified into 4 categories: sum, maximum/minimum (max/min), average (avg), and variance (var). We deploy sketches on the switch to maintain flow-level features. We discuss how to perform the extraction of these features as follows.

Table I. Examples of per-flow features.

Features	Attributes
Packet number Total packet size Total IPD	Sum
Packet size max Packet size min IPD max IPD min	Maximum/Minimum
Average packet size Average IPD	Average
Packet size variance IPD variance	Variance

Sum and maximum/minimum features: To track sum and extreme values efficiently, we extend ElasticSketch [22], which originally partitions flows into heavy and light buckets for efficient tracking. However, the original ElasticSketch design does not support simultaneous tracking of multiple flow attributes and uses an approximate structure (Count-Min sketch) unsuitable for our scenario where accurate flow-level metrics are critical.

To address these limitations, TitanLog adopts only the heavy part of ElasticSketch and extends each bucket to include multiple attribute counters for different flow features, such as cumulative packet size (sum), and the maximum and minimum packet sizes observed. Upon receiving a packet, TitanLog computes its hash to identify the corresponding bucket, proceeding with the following logic:

- If the bucket is empty or matches the incoming flow ID, TitanLog directly updates all relevant attribute counters.
- If the bucket is occupied by a different flow, TitanLog decides whether to replace the existing entry based on a predefined replacement criterion.

Average and variance features: The average features can be regarded as the quotient of two features. For example, `average packet size` is the quotient of `total packet size` and `packet number`. Therefore, we maintain these two sum features and approximately compute the average feature each time.

However, we cannot directly compute the quotient on the programmable switch. Fortunately, we do not need to know the exact value of the feature, but only whether it satisfies the condition. In an RF, the condition is usually in the form of $\frac{\alpha}{\beta} < C$, or $\frac{\alpha}{\beta} > C$ (C is a constant). Take $\frac{\alpha}{\beta} < C$ as an example. The condition is equivalent to $\alpha < C \cdot \beta$. In order to compute $C \cdot \beta$, a straightforward approach is to store all possible results in an exact match-action table. During each computation, we look up the table for the result, and compare it with α. However, when the possible range of β (denoted as B) is large, we must build a large table, which may take up too much memory.

To overcome the above limitation, we propose an approximate approach with error guarantees. Instead of the exact match-action table used in the straightforward approach, we use a range match-action table. The range of the i-th entry is $[\lfloor(1+2\delta)^i\rfloor, \lfloor(1+2\delta)^{i+1}\rfloor)$, and the result is $\lfloor C \cdot (1+\delta)(1+2\delta)^i\rfloor$. This approach guarantees the relative error of the result in δ, while the number of entries is reduced from B to $\log_{1+2\delta} B$. When $B = 2^{20}$, the number of entries in approximate approach is around 700, which is much less than that in the straightforward approach (1M).

Since the key length of range match-action table is limited to 16 bits in Tofino switches, we use two match-action tables of 16-bit keys, one for small keys (named low table) and one for large keys (named high table), instead of using one match-action table of 32-bit keys. For a 32-bit integer β, when computing $C \cdot \beta$, we first compare β with 2^{16}. If $\beta < 2^{16}$, we look up the low table for result; Otherwise, we take the high 16 bits of β as the key and look up the high table for result.

Variance computation requires calculating quadratic sums (α^2), which are resource-intensive for programmable switches. Hence, TitanLog adopts NetBeacon's [23] approximate quadratic computation method, significantly reducing resource utilization while maintaining acceptable accuracy.

B. In-network Random Forest

TitanLog leverages random forests (RF) as the in-network classifier, utilizing lightweight and easily computable features with thresholds determined through offline training on real-world labeled datasets. This ensures rapid packet classification at line rate and precise assignment to appropriate logging granularities.

Training: The RF model is trained offline on a server using real-world labeled datasets. A key challenge in designing such a classifier is the inherent instability of flow-level features during a flow's initial phase. For example, a naive classifier relying on a simple threshold (e.g., `flow_size` $\leqslant 100\texttt{MTU}$) would incorrectly classify the initial packets of all long-lived flows. To overcome this fundamental challenge, our training methodology is specifically designed to handle these dynamics.

Instead of using a dataset of summarized, completed flows, TitanLog's training data is constructed from feature snapshots captured from the switch's perspective upon the arrival of each individual packet. This approach ensures the model is extensively trained on data from these early, feature-unstable stages, learning to make robust inferences by considering the holistic combination of all available features rather than over-relying on any single volatile metric. To further enhance accuracy, we assign higher weights to anomalous packets during training to minimize false negatives.

Implementation on data plane: TitanLog's implementation follows the design framework proposed by NetBeacon [23], known for its efficiency and suitability for programmable switches. This approach selects hardware-compatible features and defines criteria and thresholds via offline training. TitanLog differentiates itself from NetBeacon by primarily using flow-level RF predictions, with packet-level predictions only employed when corresponding flow-level data from sketches is unavailable (typically for small flows).

V. Packet Data Recorder

In this section, we present the second phase of TitanLog, namely the packet data recorder. We first present how TitanLog implements the RDMA protocol with the programmable switch and the server. Then we propose how TitanLog splits large packets into appropriate sizes for RDMA encapsulation. After that, we propose how the three components work together for high-throughput disk storage.

A. RDMA Encapsulation

To address the lack of inherent support for the RDMA protocol in current programmable switches, we establish an RDMA connection between the switch's data plane and the relay server before the logging process begins. Specifically, we present the following server-switch co-design.

Server design: In line with many existing efforts [24], [18], we opt to deploy the RoCE (RDMA over Converged Ethernet) protocol [19], [20] on an Ethernet programmable switch. When encapsulating a packet with an RDMA header, the switch data plane requires RDMA connection metadata from the server, including `queue pair number` (`QPN`), `virtual address` (`VA`), and `remote key` (`RK`). However, these three fields cannot be directly obtained on the data plane. To address this, we establish a reliable RDMA connection between the switch control plane and the server, allowing us to record the necessary metadata. The server creates a local queue pair (QP), uses the QP to request a memory region (MR) in local memory, and obtains `VA` and `RK` for this MR. Subsequently, the server sends `QPN`, `VA`, and `RK` to the switch control plane using an RDMA send operation. The control plane receives these fields through RDMA receive operation, and writes them to a match-action table on the data plane via switch API.

Switch design: When encapsulating a packet with an RDMA header, TitanLog fills the `QPN`, `VA`, and `RK` fields in the header with the pre-recorded data from the match-action table. Additionally, there are two more fields that need to be populated

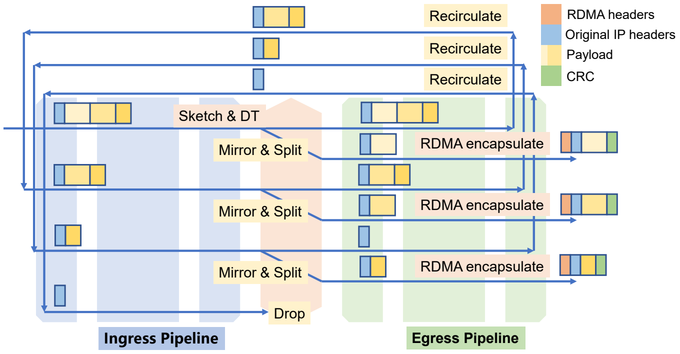


Figure 2. RDMA encapsulation and packet splitting.

based on the cumulative packet count and length: `packet sequence number` (`PSN`) and `address offset` (`AO`). These fields require dynamic maintenance on the data plane. To achieve this, we use a register to store `PSN` for the RDMA connection. For each transmitted packet, TitanLog populates this register value into the RDMA header, and then increments it by 1. Similarly, we utilize another register to maintain `AO` and update it according to the packet length.

B. Packet Splitting

There are two hardware limitations in encapsulating the packet data into RDMA message. First, current programmable switches do not support adding fields to the end of the packet. Therefore, it becomes challenging to include the CRC checksum, which is typically located at the end of the entire RDMA packet. Second, current switches can only parse constant-length fields with a limitation of about 200B, while packets can be large (e.g., 1500B). To overcome these limitations, our main approach is to split the payload of the packets into multiple segments, each with a size less than 128B, and encapsulate each segment as a separate RDMA write message.

For each transmitted packet, in the ingress pipeline, TitanLog mirrors it, and marks one of the mirrored packets with a `mirror` flag. Then, in the egress pipeline, TitanLog parses the IP header of the packet and breaks down the `total_len` field into 8 sub-fields: $\texttt{total\_len}[15:0] = \texttt{total\_len}[15:7] : \texttt{total\_len}[6:6] : \cdots : \texttt{total\_len}[0:0]$. Based on these 8 sub-fields, TitanLog determines the length of the segment that needs to be split. If $\texttt{total\_len}[15:7] \neq 0$, indicating that the payload length exceeds 128B, TitanLog splits the first 128B of the payload as the payload of the split segment (in the form of a packet header). Specifically, for packets without the `mirror` flag, TitanLog sets the `total_len` field to 128B, encapsulates the headers and the split payload as an RDMA packet, adds the CRC checksum field to the end of the packet, and then forwards the encapsulated packet to the server. For packets with the `mirror` flag, TitanLog removes the first 128B from the payload, subtracts `total_len` by 128B, deparses the packet, and recirculates it through the recirculation port to the ingress pipeline for further splitting process. Note that the recirculated packets bypass the classifier (i.e., sketches and the RF).

If `total_len`$[15:7]=0$, indicating that the payload length is less than 128B, TitanLog proceeds to check whether the `total_len`$[6:6]$ field is 0. If it is not 0 (indicating that the payload length exceeds 64B), TitanLog splits the first 64B of the payload as the payload of the segment. The splitting process is similar to that of splitting a 128B segment. Otherwise, TitanLog continues to check the `total_len` fields and split the packet, until `total_len`$[0:0]=0$. For example, consider a packet with a 1480B payload. With this approach, the packet will be split into 13 segments: 11 128B segments, a 64B segment, and an 8B segment. This approach also addresses the problem of the encapsulated packet exceeding the MTU limitation.

Overhead analysis: Under typical conditions with an MTU of 1500B, a packet may be split into at most 16 segments: up to 11 128B segments, a 64B segment, a 32B segment, a 16B segment, an 8B segment, and a 4B segment. Since packets are typically aligned to 4B, segments smaller than 4B are usually unnecessary. Consequently, each packet requiring full logging could incur additional overhead for up to 15 segment headers, resulting in an overhead of approximately 600B for TCP packets. We experimentally measured this overhead (see Figure 4). Results demonstrate that the logging volume approximately doubles for anomalous traffic. Considering typical packet sizes range from 700 to 1000B, this finding aligns well with our analytical expectations.

Fortunately, in most scenarios, anomalous traffic constitutes a small proportion of total traffic, and thus this overhead remains small compared to the storage savings achieved for benign traffic. However, in extreme scenarios where anomalous traffic exceeds 50% of the total volume, logging full-packet becomes challenging. To address this, we propose mechanisms in Section VI to detect when the log volume exceeds manageable limits and offer various logging granularity adjustments to handle such situations effectively.

C. Data Storage

To store packet data from the server to disks, we use a commercial storage system, and design a mechanism to store data in the form of batches. The storage system comprises multiple smartNICs and remote disks. Each smartNIC is connected to the relay server and several disks. Leveraging the NVMe-oF protocol, the storage system enables writing data from the server to remote disks as if they were local disks, ensuring efficient and fast data storage.

We employ a switch-side triggering mechanism for data storage in TitanLog. As a reminder, TitanLog directly writes packet data to the server memory using RDMA write messages. The server's CPU is not involved in this process, which means the server cannot directly sense the volume of data being logged in its memory.

The switch-side triggering mechanism works as follows. We virtually divide the memory buffer (memory region in the RDMA context) on the server into two blocks. Let $\mathcal{M}$ represent the total buffer size (e.g., 16GB). The offsets of the two blocks are $\left[0, \frac{\mathcal{M}}{2}\right)$ and $\left[\frac{\mathcal{M}}{2}, \mathcal{M}\right)$, respectively. On the switch data plane, when `address offset` (`AO`) approaches the end of one block (i.e., `AO` $\geqslant \frac{\mathcal{M}}{2} - \text{MTU}$, or `AO` $\geqslant \mathcal{M} - \text{MTU}$), TitanLog recognizes that this memory block is nearly full, and switches to writing to the other memory block. Specifically, the data plane sends a `block-full` message to the control plane through the switch API, and then sets `AO` to the beginning of the other block (i.e., $\frac{\mathcal{M}}{2}$ or 0). The control plane receives the `block-full` message and forwards it to the server via a TCP connection. Upon receiving the message, the server writes the full block to the storage system (using multiple threads), and then clears the block. This mechanism ensures that very little space is wasted (less than one MTU per block).

VI. Adaptive Logging Manager

In this section, we present our design that provides elasticity. We first present the logging granularities and modes of TitanLog. Then we show how TitanLog senses resource insufficiency to switch between logging modes during network dynamics.

A. Logging Granularities and Modes

As mentioned earlier, TitanLog logs packet data at different levels. We call these log levels *log granularities*. In the previous sections, we mainly focus on two log granularities, i.e., logging both headers and payloads, and logging only headers to reduce disk overhead. Through our implementation, we find that besides disk capacity, two additional resources can potentially become bottlenecks: mirror bandwidth and RDMA throughput. Consequently, we propose two more granularities to mitigate these bottlenecks. The complete list of log granularities is provided below.

- *Full logging:* Logging all headers and payload of packets. In this granularity, TitanLog splits the packet, encapsulates it as RDMA messages following the method described in Section V-B, and sends them to the server for storage.
- *Header-only logging:* Discarding the payload and logging only the packet headers. In this granularity, TitanLog encapsulates the packet headers as an RDMA message and sends it to the server. With header-only logging, we can reduce the log volume to match the disk capacity.
- *Partial-load logging:* Logging all packet headers and partial payload (e.g., 128B or 256B), discarding the remaining payload. Partial-load logging can preserve some important information in the packet, for example, the application layer protocol. Partial-load logging can be implemented by modifying the process of packet splitting. Take 128B splitting as an example. If `total_len`$[15:7]$ is not 0 (i.e., the packet length is larger than 128B), TitanLog still splits the first 128B from the payload, encapsulates it as an RDMA write message, and transmits it. The main difference is that TitanLog does not mirror the rest of the payload, but simply discards it. If `total_len`$[15:7]$ is 0 (i.e., the packet length is smaller than 128B), TitanLog performs the same process as Section V-B. With partial-load logging, we can reduce the number of mirrors to accommodate the mirror bandwidth.
- *Sampled logging:* On the basis of the above logging granularities, we can further sample the packets for logging. In

Table II. RECOMMENDED LOGGING MODES.

Mode	Log resolution		Description
	Anomalous flow	Benign flow	
Default mode	Full logging	Header-only	Default mode to realize hierarchical logging.
Log-all mode	Full logging	Full logging	Used to log all data when serious failure occurs.
Log-partial mode	Partial-load	Header-only	Used when mirror bandwidth is insufficient.
Log-sampled mode	Full logging	Sampled header-only	Used when RDMA throughput is insufficient, or to reduce log volume when disk capacity is insufficient.
Log-minimum mode	Partial-load	Sampled header-only	Used to minimize log volume.

this granularity, TitanLog only logs the data of the sampled packets. For other packets, TitanLog discards them directly. Users can also configure the sampling ratio through switch API according to the traffic volume. Sampled logging can preserve the traffic distribution. This is also the minimum granularity that preserves the per-flow information. With sampled logging, we can reduce the number of the packets transmitted to the server to match the RDMA throughput. Sampled logging can also reduce log volume to match the disk capacity.

The four logging granularities are widely used in network logging applications. Notably, both the sampled logging and partial-load logging modes offer flexible configuration options. By adjusting the sampling ratio in the sampled logging mode and the truncation length in the partial-load logging mode, users can effectively fine-tune the granularity of the logs. This flexibility enables the system to record more detailed logs when finer granularity is needed, or to reduce the logging overhead in resource-constrained scenarios. Such dynamic adjustment capabilities not only accommodate the primary use cases discussed in this work but also extend the applicability of TitanLog to a broader range of network monitoring and analysis tasks.

Based on the above log granularities, we provide several recommended logging modes (see Table II) for different network environments and hardware support. Network operators can also configure other modes as needed. To realize the logging modes, we add a table between the two phases to match each packet category to a logging granularity. TitanLog can switch modes fast and flexibly by reconfiguring the table through switch API rather than re-compiling.

B. Sense and Respond to Resource Insufficiency

Insufficiency in disk capacity: TitanLog senses remaining disk size on the server. Specifically, we maintain a global counter on the relay server, which records the cumulative size of data written to disks. Each time the relay server writes buffered logs to the storage system, it increases this counter accordingly and then checks whether the remaining available disk space (total disk capacity minus the counter) is below a predefined threshold (e.g., 5% of the total disk capacity). If the remaining disk size falls below a pre-defined threshold, TitanLog warns that it is suffering from the bottleneck of disk capacity. There are two ways to handle this situation. First, the operators can switch TitanLog to log-sampled mode to reduce log volume. Second, if the disk insufficiency is due to the arrival of a traffic burst, we can overwrite unimportant data. This may cause a slight data loss, which is acceptable.

Insufficiency in mirror bandwidth: One reason for the insufficiency in mirror bandwidth is that in large packet splitting, TitanLog mirrors the large packets multiple times, resulting in data amplification. This insufficiency may cause packet loss in the switch pipeline.

In order to detect the insufficiency in mirror bandwidth, we maintain a counter on the switch data plane to record the amount of mirror data. When splitting a packet, we increase the counter by the length of the mirror packet. Every second, the switch control plane reads and resets the counter. If it exceeds the threshold, TitanLog warns that it is suffering from insufficiency in mirror bandwidth, and switches to log-partial mode.

Insufficiency in RDMA throughput: Packet loss may occur when the server receives too many RDMA messages. We use the reliable RDMA protocol to transmit packet data between the programmable switch and the server. When the server receives an RDMA message, it sends an RDMA ACK message to the programmable data plane. If a packet loss occurs, the server sends an RDMA NACK message and waits for retransmission. The server will drop all other packets until receiving the first lost packet. Therefore, it is important to sense and recover RDMA packet loss in time.

Fortunately, in log systems, slight packet loss in a very short period is acceptable, because it has almost no effect on analysis performance. In order to detect and handle packet loss caused by RDMA throughput bottleneck in time, TitanLog monitors RDMA NACK messages on the programmable switch. Specifically, TitanLog parses RDMA responses on the switch data plane and checks whether there is a NACK message. TitanLog uses a bit on the data plane to record the receipt of the NACK packet. At the same time, the control plane reads this bit periodically (e.g., 1ms). When the bit is set to 1, the control plane reconfigures relevant counters (`PSN` and `AO`) to resume RDMA transmissions if it finds that it has received NACK packets. Meanwhile, TitanLog warns that it is suffering from insufficiency in RDMA throughput, and switches to log-sampled mode. Note that with this mechanism, there is still packet loss in the 1ms.

Insufficiency in switch queue buffer: Under bursty traffic, the switch buffer may become full, resulting in packet loss. To handle this issue, TitanLog periodically monitors queue congestion status using a congestion flag maintained by the switch data plane. Specifically, when the packet queue length on the programmable switch exceeds a predefined threshold, TitanLog sets this congestion flag to indicate potential buffer overflow. The switch control plane periodically reads and resets this flag. If congestion is detected, TitanLog triggers an alert indicating the switch queue buffer is experiencing high occupancy. In response, TitanLog automatically switches to log-sampled mode, thereby proactively mitigating the risk of packet loss due to queue overflow.

VII. Implementation

Switch data plane: We implement TitanLog's data plane design in P4 [25] and compile it to the Tofino switch [26]. The data plane implementation consists of three parts. For sketches and the RF, we implement them in the ingress pipeline. For packet splitting, we implement the mirror operation in the ingress pipeline, and the split operation in the egress. The RF consists of 3 7-layer decision trees. The sketch consists of two layers, each consisting of 2^{16} buckets. We use 32-bit registers to extract the sum of packet length and IPD. For other features, we use 16-bit registers to conserve SRAM resources.

The resource usage on the switch data plane is shown in Table III. TitanLog consumes SALUs the most, at 85.42%. This is because TitanLog deploys multiple sketches for feature extraction, all of which require SALUs to access memory. SRAM is another heavily consumed resource. This is because we want the sketches to be as large as possible in order to accurately extract features and classify flows. We consider this acceptable: TitanLog is dedicated to logging network traffic, so it can monopolize all resources.

Table III. Resource usage on the Tofino switch.

Resource	Usage	Percentage
Exact Match Input xbar	303	19.73%
Ternary Match Input xbar	4	0.51%
Hash Bit	797	15.97%
SRAM	233	24.27%
Map RAM	214	37.15%
TCAM	1	0.35%
VLIW Instr	57	14.84%
Stats ALU	41	85.42%

Switch control plane and server: We implement the switch control plane in Python and C++. It manages the switch data plane (e.g., clearing and reading registers) through a run-time API generated by the P4 compiler. We use C++ to implement the server design. We establish two connections between the server and the control plane: an RDMA reliable connection for acquiring RDMA metadata, and a TCP connection for transmitting the `block-full` message.

VIII. Experimental Results

We fully implement TitanLog on our testbed and conduct extensive experiments to investigate the following key questions:

- **What is the accuracy of our in-network classifier in classifying packets?** We validate the effectiveness of the classifier in packet classification, verify the accuracy of the sketch algorithm in obtaining per-packet features, and demonstrate that the in-network classifier can classify packets with extremely high accuracy (§VIII-B).
- **How much traffic can we reduce through our data plane TitanLog?** We validate that for five datasets with different traffic distributions, by using TitanLog in the data plane, we can significantly reduce the bandwidth and packet rate of traffic passing through the switch. We also assess the impact of these mechanisms on the bandwidth and packet rate of anomalous traffic both inside and outside the data plane, and verify that the additional cost is entirely acceptable (§VIII-C).
- **Can we easily record traffic using relay servers and storage systems?** We demonstrate the real-time CPU workload of the relay server during the entire operation of TitanLog and the time taken for various tasks such as writing to disk, creating the new file, etc., and verify that even a server with weak CPU is completely sufficient for processing (§VIII-D).

A. Experiment Setup

Testbed setup: The testbed consists of a Tofino programmable switch, two servers (each with 24 CPUs and 32GB of memory), and a storage system with 4 1TB disks. The servers and the switch control plane are equipped with Mellanox ConnectX-5 NIC to support RDMA protocol. We use one server, with the switch and the storage system, to build a TitanLog prototype. All three components are connected through 100Gbps cables. We use the other server as a traffic sender. The sender is connected to the programmable switch data plane through 100Gbps cables.

Traces: We use the CIC-IDS2017 datasets [27]. The dataset contains benign and the most up-to-date common attacks, which resembles the true real-world data. There are 5 traces in the dataset. The trace from Monday consists exclusively of benign traffic, while the other traces comprise a mixture of benign traffic and various types of attacks, each exhibiting distinct traffic distributions. We list the detailed information of the traces under 100Gbps in Table IV. For other input bandwidths, the ratio of anomalous bandwidth to packet rate is the same, though the absolute values vary.

We use the DPDK [28] driver to replay the traces and vary the bandwidth. For bandwidth less than 70Gbps, we send the packets in sequence with a single thread. For bandwidth more than 80Gbps, we divide each trace into 8 sub-traces with equal length, and send the 8 sub-traces with 8 threads/queues simultaneously. The packets in each sub-trace are sent in sequence. We repeatedly replay the datasets to generate continuous high-speed traffic, ensuring that TitanLog's capability to handle sustained high-volume network traffic is effectively evaluated.

B. Evaluation on In-network Classification

In this section, we evaluate the accuracy of the sketch as well as the performance of the in-network classifier of TitanLog.

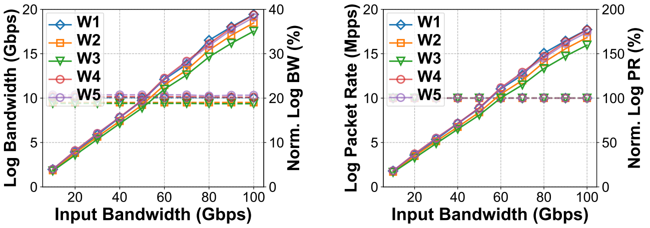
The accuracy of the sketch (Table V): We evaluate the accuracy of the sketches in obtaining 5 different flow-level features. The experimental results show that the sketches achieve ARE (average relative error) of 1e-2, 2e-4, 4e-3, 4e-4, and 6.83. We observe that the sketch achieves high accuracy when estimating the feature `size avg`. This is because the errors from estimating `size sum` and `packet count` accumulate simultaneously, thus effectively canceling each other out and reducing the overall estimation error of `size avg`. Conversely, the sketch has the lowest accuracy

Table IV. STATISTICAL INFORMATION OF THE DATASET UNDER 100GBPS.

Trace	max # active flows (k)	Packet rate (Mpps)			Bandwidth (Gbps)	
		Total	Anomalous	Ratio (%)	Anomalous	Ratio (%)
Monday (W1)	33.356	15.214	0	0	0	0
Tuesday (W2)	30.865	14.694	0.344	2.344	0.153	0.153
Wednesday (W3)	32.711	14.418	2.551	17.695	16.621	16.621
Thursday (W4)	58.638	15.856	0.163	1.029	0.348	0.348
Friday (W5)	295.509	15.986	2.135	13.353	12.039	12.039

Table V. ARE OF SKETCH ALGORITHMS.

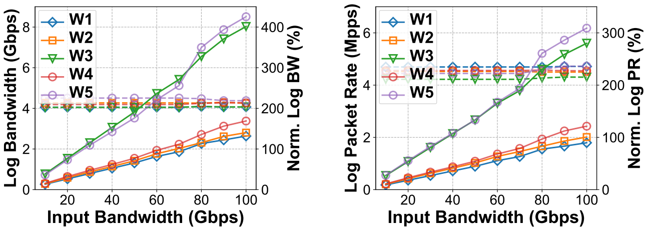
	W1	W2	W3	W4	W5
Size sum	2e-2	3e-2	1e-2	6e-3	7e-3
Size max	6e-5	4e-5	6e-5	4e-5	1e-3
Size min	9e-4	4e-4	8e-4	1e-2	1e-2
Size avg	7e-5	2e-5	3e-5	4e-4	1e-3
Size var	1.36	1.64	1.19	1.88	28.1



(a) Bandwidth.

(b) Packet rate.

Figure 3. Bandwidth (BW) and packet rate (PR) of benign traffic logs output from the switch. Dashed line are the results normalized relative to the original benign traffic.



(a) Bandwidth.

(b) Packet rate.

Figure 4. Bandwidth (BW) and packet rate (PR) of anomalous traffic logs output from the switch. Dashed line are the results normalized relative to the original anomalous traffic.

when estimating the feature `size var`. This is primarily due to the considerable errors introduced by approximate multiplication operations. Nevertheless, the sketch can still accurately estimate all 5 features, thus achieving high-accuracy packet classification in the data plane.

The performance of the classifier (Table VI): We evaluate the precision, recall, and accuracy of TitanLog's classifier. TitanLog achieves an average precision, recall, and accuracy of 0.486, 0.846, and 0.949. Compared to NetBeacon [23], TitanLog's classifier achieves higher performance on all workloads. We hypothesize that this improvement arises because TitanLog uses a more accurate flow-level RF to predict large flows as well as a subset of small flows, while NetBeacon uses flow-level RF only for flows identified as large.

C. Evaluation on End-to-end Performance

In this section, we first investigate how hierarchical logging affects the bandwidth and packet rate of traffic logs for benign and anomalous traffic. We also evaluate how much we can reduce network load in the log-minimum mode.

The network load of benign traffic logs (Figure 3): We demonstrate the bandwidth and packet rate of logs generated by benign traffic under different traffic distributions. The experimental results show that the bandwidth of benign traffic logs output by the switch varies between 1.80Gbps and 19.43Gbps, while the packet rate varies between 1.64Mpps and 17.72Mpps. Compared with the original benign traffic, the policy of only retaining the packet header can reduce the average traffic bandwidth by 80.23%.

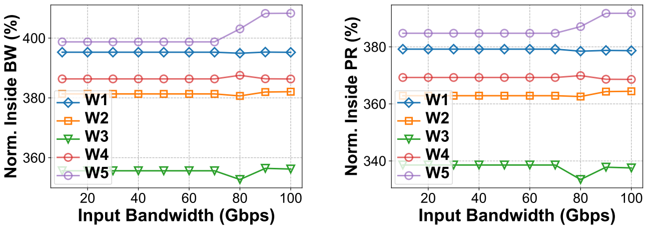
The network load of anomalous traffic logs (Figure 4-6): We first demonstrate the bandwidth and packet rate of logs generated by anomalous traffic under different traffic distributions. As shown in Figure 4, we find that the bandwidth of anomalous traffic logs output by the switch varies between 0.27Gbps and 8.51Gbps, while the packet rate varies between 0.18Mpps and 6.18Mpps. In other words, the output bandwidth increases by 1.03 to 1.25 times, and the packet rate increases by 1.11 to 1.36 times, compared to the original anomalous traffic. This implies that each anomalous traffic packet is split into an average of 2.25 log packets, and its size is expanded by 1.11 times on average. This increase is expected due to the presence of packet splitting and RDMA encapsulation mechanisms.

We also demonstrate the bandwidth and packet rate of log-related traffic generated within the switch. As shown in Figure 5, we find that the bandwidth of log-related traffic generated within the switch increases by 2.56 to 3.08 times, and the packet rate increases by 2.38 to 2.92 times, compared to the original anomalous traffic. This implies that for each anomalous traffic packet, the data plane generates an average of 3.84 log-related packets, with their total size being 3.67 times that of the original packet.

As mentioned in Section V-B, in the worst-case scenario, packet splitting mechanisms can split a single anomalous packet into more than a dozen log packets and perform RDMA encapsulation, which may significantly increase the log traffic output from the switch. Moreover, packet splitting is implemented on the data plane through loopbacks, making the increase in internal switch traffic even more pronounced. Surprisingly, the analysis of traffic belonging to various distributions reveals that packets belonging to attack traffic generally have a smaller payload, which are often powers of two. As a

Table VI. THE PERFORMANCE OF IN-NETWORK CLASSIFIER.

	W1	W2			W3			W4			W5		
	ACC	PRE	REC	ACC	PRE	REC	ACC	PRE	REC	ACC	PRE	REC	ACC
Ours	0.951	0.453	0.999	0.976	0.817	0.997	0.942	0.073	0.394	0.917	0.835	0.991	0.976
NetBeacon	0.944	0.435	0.999	0.974	0.834	0.947	0.935	0.051	0.348	0.898	0.816	0.994	0.973



(a) Bandwidth. (b) Packet rate.

Figure 5. Bandwidth (BW) and packet rate (PR) of the packets related to anomalous traffic logs generated inside the switch. The results are normalized relative to the original anomalous traffic.

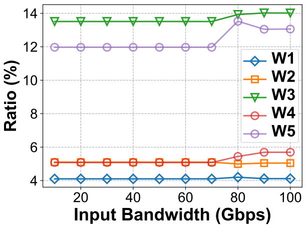


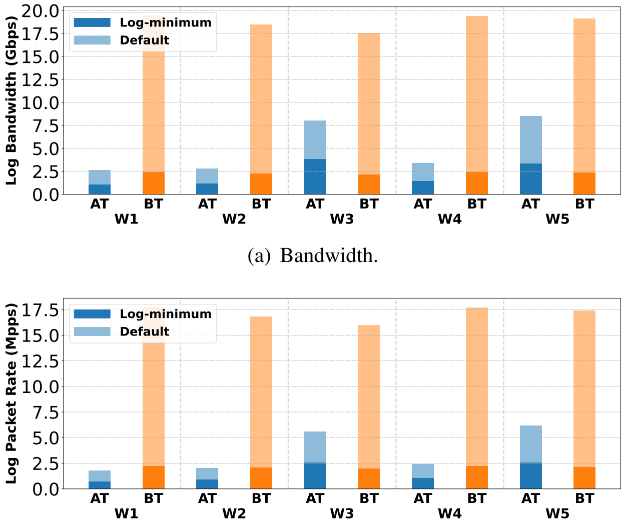
Figure 6. Ratio of detected anomalous packets to total packets.

result, the actual traffic amplification for anomalous flows is much lower than the theoretical worst-case scenario.

Figure 6 illustrates the ratio of detected anomalous packets. The experimental results show that when a network or system attack actually occurs, TitanLog detects 4.10% to 14.02% of the overall traffic as anomalous packets. Consequently, the impact of logging overhead on the overall switch throughput is modest. This degree of additional internal and output traffic generally does not create bottlenecks in mirroring and loopbacks.

The network load under log-minimum mode (Figure 7): We demonstrate the bandwidth and packet rate of the logs generated in log-minimum mode for different traffic distributions. Experimental results show that the bandwidth of benign traffic logs output by the switch varies between 2.19Gbps and 2.43Gbps, while the packet rate varies between 2.00Mpps and 2.22Mpps; the bandwidth of anomalous traffic logs output by the switch varies between 1.07Gbps and 3.84Gbps, while the packet rate varies between 0.76Mpps and 2.61Mpps. Compared to original traffic, log-minimum mode on average reduces the overall traffic bandwidth by 96.28% and the overall packet rate by 80.14%.

The frequency of RDMA errors (Figure 8): When traffic arrives in a burst form, it may cause the packet rate of the log traffic output by the switch to exceed the processing capacity limit of the receiving end RDMA NIC of the relay server, thereby triggering RDMA errors. RDMA errors can be detected through NACK messages sent from the receiving end NIC to the switch. We count the frequency of RDMA errors under different traffic distributions. Experimental results show that no RDMA errors occur under traffic load of 100Gbps. However, when using log-all mode, RDMA errors occur up to 9 times under traffic load of 40Gbps, causing some log data packets to not be written to the file normally. When traffic load exceeds 50Gbps, packet loss increases significantly. We speculate that this is because the internal switch traffic exceeds the bandwidth of the recirculate port. We compare the log capacities of TitanLog to tcpdump (see Table VII). We find that when logging 10Gbps of traffic, tcpdump has a packet loss rate of about 60%. In contrast, the default and log-all modes of TitanLog do not result in any packet loss. We also



(a) Bandwidth.

(b) Packet rate.

Figure 7. Bandwidth and packet rate of traffic logs output from the switch in log-minimum mode. “AT” and “BT” refer to anomalous traffic and benign traffic, respectively.

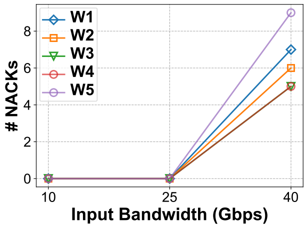
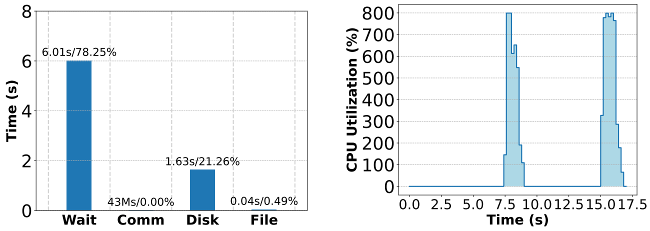


Figure 8. Number of RDMA anomalies (NACK messages) occurrences under log-all mode.

Table VII. PACKET LOSS RATE OF TCPDUMP UNDER 10GBPS.

	W1	W2	W3	W4	W5
Loss rate (%)	64.7	64.4	66.6	57.7	62.3



(a) Task Processing Time.

(b) Real-time CPU Load.

Figure 9. CPU usage statistics on the relay server.

compare the log capacities of TitanLog to xdpcap [8]. We find that the packet loss rate of xdpcap is similar to that of tcpdump. We suspect this is because xdpcap only optimizes the speed of packet processing, but does not improve the performance of storage, making storage the bottleneck of the whole system.

Response time: We evaluate the time interval from the generation of an anomaly signal to the completion of the logging mode switch. The experimental results show that the response time is less than 1 ms, indicating that TitanLog can quickly and efficiently respond to resource insufficiency and network dynamics.

D. Evaluation on Relay Server

In this section, we analyze the CPU load on the relay server under a high-speed network load of 100Gbps.

CPU load analysis (Figure 9): We use 16GB memory on the relay server as a buffer for RDMA writes and employ 16 threads to write logs from memory to disks. These threads are divided into two groups, with 8 threads per group, and each thread is responsible for managing a 1GB memory buffer. Whenever the switch outputs traffic logs that fill up 8GB of memory, the switch notifies the relay server using a network message, and the relay server writes to the disks using 8 threads. Following this, these threads clear the memory, create new files, and wait for the next message from the switch. As shown in Figure 9(a), during a processing cycle of approximately 7.67s, the CPU idle waiting time is about 6.01s, the disk writing time is about 1.63s, the new file creation time is about 0.04s, and the communication time with the switch can be considered negligible. As shown in Figure 9(b), within a cycle, only approximately 1.6s of disk writing requires CPU usage, which accounts for 20% of the entire cycle. This implies that even if the network traffic increases fivefold, the current relay server can still handle disk writes.

E. Evaluation on Scalability

In this subsection, we demonstrate the scalability potential of TitanLog for future networks operating at higher bandwidths. Due to hardware limitations, we simulate and extrapolate the performance of the sketch and classifier under increasing bandwidth scenarios. To this end, we implement a high-level logical simulator in Python that accurately reproduces the core processing logic of the switch data plane, including sketch-based feature extraction, the RF classifier, and the rules for processing packets according to different logging granularities.

Table VIII. ARE OF SKETCHES UNDER HIGHER BANDWIDTH.

		W1	W2	W3	W4	W5
Size sum	200Gbps	0.03	0.03	0.01	0.01	0.01
	400Gbps	0.06	0.05	0.04	0.03	0.04
Size max	200Gbps	5e-3	3e-3	3e-3	4e-3	4e-3
	400Gbps	0.02	0.02	0.02	0.02	0.02
Size min	200Gbps	0.08	0.05	0.05	0.08	0.08
	400Gbps	0.42	0.34	0.32	0.36	0.41
Size avg	200Gbps	5e-3	3e-3	3e-3	5e-3	5e-3
	400Gbps	0.03	0.02	0.02	0.02	0.02
Size var	200Gbps	1.32	1.66	1.18	1.68	27.5
	400Gbps	1.36	1.68	1.16	1.67	16.0

To generate higher-bandwidth traffic, we adopted a proportional scaling methodology. Specifically, for each target bandwidth N Gbps (e.g., 200Gbps, 400Gbps), we simultaneously replay $\frac{N}{100}$ parallel streams of the dataset, assigning distinct flow identifiers to each parallel stream. This approach ensures that the number of concurrent flows proportionally increases with bandwidth, while keeping the distribution of flow size, duration, and the proportion of anomalous traffic consistent across scenarios.

To accurately analyze the system's performance bottlenecks, our simulation is based on the following constraints derived from hardware specifications and our prior experiments:

- *Switch internal bandwidth:* The packet splitting and recirculation operations required for "full logging" consume internal switch bandwidth. We set this bandwidth limit to 100Gbps, consistent with the capacity of the physical switch's recirculation port.
- *RDMA throughput:* Based on our experimental results in Section VIII-C, RDMA errors (NACK messages) begin to appear when the output traffic in log-all mode reaches 40Gbps. Therefore, we set the effective RDMA throughput limit of the relay server to approximately 60Gbps, beyond which significant data packet loss is expected.
- *Relay server processing capability:* The server is configured with a 16GB memory buffer for RDMA writes. According to our measurements in Section VIII-D, it takes the server CPU approximately 1.67s to persist a full 8GB memory block of data to the disk system.
- *Disk capacity:* We assume that disk capacity can be provisioned as needed. Therefore, our analysis focuses on throughput-related bottlenecks rather than storage capacity.

The accuracy of the sketch (Table VIII): The experimental results show that although the sketch accuracy decreases with higher bandwidth, it maintains high accuracy (ARE $<$ 0.1) for sum, maximum, minimum, and average features under 200Gbps, which corresponds to about 590k active flows. At 400Gbps, the ARE of `size min` rises to over 0.4, which significantly exceeds the empirical threshold (e.g., ARE $<$ 0.1) typically expected for high-fidelity flow measurement tasks. This indicates that the sketch can support up to 590k

Table IX. The performance of in-network classifier under higher bandwidth.

	W1	W2			W3			W4			W5		
	ACC	PRE	REC	ACC	PRE	REC	ACC	PRE	REC	ACC	PRE	REC	ACC
200Gbps	0.943	0.435	0.999	0.974	0.805	0.997	0.938	0.066	0.394	0.910	0.823	0.991	0.974
400Gbps	0.918	0.362	0.998	0.965	0.765	0.998	0.921	0.055	0.340	0.892	0.783	0.991	0.967

Table X. Bandwidth of traffic logs output from the switch under higher bandwidth (inGbps).

	W1	W2	W3	W4	W5
200Gbps	43.003	41.642	77.459	44.917	70.808
400Gbps	93.096	88.153	159.756	94.662	146.580

Table XI. Packet rate of traffic logs output from the switch under higher bandwidth (in Mpps).

	W1	W2	W3	W4	W5
200Gbps	39.379	38.156	54.610	41.130	55.511
400Gbps	82.196	78.618	111.554	84.650	113.422

Table XII. Bandwidth of the packets generated inside the switch under higher bandwidth (inGbps).

	W1	W2	W3	W4	W5
200Gbps	15.791	14.474	65.949	15.924	50.087
400Gbps	40.518	35.365	138.004	27.855	106.523

active flows with reasonable errors. It is worth noting that the classifier's performance degradation is much smaller compared to that of the sketches. We believe this is because, for the vast majority of flows (especially large flows), the sketch accuracy remains very high, while accuracy significantly drops for only a small portion of flows.

The performance of the classifier (Table IX): We evaluate the precision, recall, and accuracy of TitanLog's classifier. We find that TitanLog's classifier achieves high performance even under 400Gbps. The precision, recall, and accuracy of TitanLog's classifier degrade by 0.053, 0.014, and 0.023 after the bandwidth expands to 400Gbps, but still with a recall of 0.832 and an accuracy of 0.933.

Network load (Table X-XII): The results show that as the input bandwidth increases, the output bandwidth, packet rates, and internally generated traffic also increase. These metrics are direct observations from the simulation under the specified constraints. At 200Gbps across the five workloads, both the output bandwidth and internal switch traffic remain below the 100Gbps port limit. However, for workloads W3 and W5, the log bandwidth exceeds the 60Gbps RDMA throughput threshold. This is expected to generate numerous RDMA errors and substantial packet loss, suggesting that RDMA emerges as the primary bottleneck in these scenarios. At 400Gbps, these metrics commonly approach or surpass the respective limits, indicating multiple potential bottlenecks and similar risks of significant packet loss. These outcomes are derived from simulation observations based on defined thresholds, indicating potential bottlenecks when metrics exceed hardware limits, rather than actual programmed packet drops. To mitigate such issues, strategies like increasing recirculation ports or enhancing RDMA throughput capacity could be employed.

IX. Discussion and Limitations

Handling encrypted traffic: We acknowledge that the proliferation of encrypted traffic presents a significant challenge to network monitoring systems, including TitanLog. Since performing line-rate decryption within the data plane of a programmable switch is computationally infeasible, TitanLog cannot inspect encrypted packet payloads, even if the switch has the private key. This inherently limits its ability to detect threats that rely on deep packet inspection (DPI), such as those with specific payload signatures. Consequently, the classification accuracy for certain application-layer attacks may be affected. However, TitanLog remains effective for a broad range of traffic classification tasks, as it does not solely depend on payload content. A substantial body of research has demonstrated that encrypted traffic can be effectively classified by analyzing statistical features extracted from unencrypted packet headers and flow metadata [29], [30]. TitanLog's architecture leverages this principle by using its sketch-based module to capture the behavioral fingerprints of flows, such as packet size distributions, inter-arrival times, and overall flow volume. These characteristics often reveal distinct patterns for different applications and can effectively distinguish anomalous activities, like DDoS attacks or C&C communication, from benign traffic, even when the payload is opaque. Although TitanLog's packet classifier cannot inspect encrypted payloads, server-based operations can still utilize the logged encrypted payloads—for instance, through offline decryption if keys are available, or for analyses based on traffic replay.

Reliance on pre-trained models: Another limitation is TitanLog's reliance on an offline, pre-trained classifier to identify traffic categories. For applications like intrusion detection and traffic replay, which rely on categorizing traffic based on known patterns, this reliance means the system cannot proactively identify and log zero-day threats whose patterns were not in the training data. However, this does not impede its function of efficiently logging all traffic that it has been trained to recognize. The impact of this limitation can be mitigated by performing timely updates to the classifier as soon as new threat signatures become available. For the distinct challenge of actively discovering novel patterns, a different operational strategy is necessary, as we discuss below.

Hierarchical logging for other applications: While this paper primarily illustrates the hierarchical logging concept using intrusion detection and traffic replay as representative examples, the methodology is inherently extensible to various other network scenarios. Below, we briefly discuss potential applications beyond the two scenarios discussed above:

- *Offline congestion analysis [31].* TitanLog facilitates offline congestion analysis by capturing high-fidelity packets from potentially congested flows, providing raw data instead of pre-processed metadata. The classifier identifies these packets using features from two sources: direct signals like Explicit Congestion Notification (ECN) CE codepoints parsed directly from packet headers, and flow-level features maintained by the sketch (e.g., per-flow packet counts and IPD). Once categorized as potentially congested, packets are handled according to the pre-configured logging granularity (e.g., full logging). To record packet arrival timestamps, we add a metadata field during the RDMA encapsulation process on the switch, which captures the ingress timestamp and includes it in the logged data. This enables engineers to diagnose the root causes of congestion via metrics like RTT, latency, jitter, and packet rates—for example, extracting per-flow RTT by comparing timestamps of data packets and corresponding ACK packets to reconstruct events.

- *New intrusion pattern discovery [32].* Although TitanLog is a traffic logging system and not an intrusion detection system, it can be configured to produce curated datasets that facilitate the discovery of new intrusion patterns. This process is enabled by a conservative classification strategy. In this configuration, the classifier is not trained to find specific attacks, but rather to identify a baseline of known, benign traffic with very high confidence. Any packet that does not match this high-confidence benign profile is categorized separately as unknown or potentially anomalous. Following this classification, the system's active logging mode applies a differentiated logging granularity: the high-confidence benign traffic is logged minimally (e.g., header-only or sampled logging), while all other packets are subjected to full-payload logging. By systematically filtering out the high-volume, predictable background traffic, TitanLog provides security analysts with a high-fidelity data stream containing all potentially interesting events. This curated dataset is invaluable for sophisticated offline analysis and the ultimate discovery of novel threats.

X. Related Work

Traffic log: Tcpdump is the most classic tool for logging network packets. Some work improves the per-packet processing speed by reducing copying and using dedicated data structures [7], [8], [9]. Other work reduces the log volume by truncating and/or sampling [10], [11]. The biggest difference between TitanLog and existing work is that we propose the hierarchical logging methodology, and are the first to use switch-server co-design for traffic logging. Some existing work attempts to log differently. However, systems based on CPUs and GPUs face challenges with insufficient packet processing performance. Thanks to the high speed and high programmability of the programmable switch, TitanLog successfully uses complex classifiers while not reducing throughput.

Sketches: With the rise of data plane programmability, more and more sketches are being deployed on programmable switches [33], [34], [35], [36], [22], [37], [38], [39], [40], [41], [42], [43], [44], [45], [46], [47], [48]. Many sketches focus on estimating sum features and their variations (e.g., heavy hitters, heavy changes, flow size distribution, etc.). Typical sketches include the CM sketch [33], Count sketch [34], UnivMon [35], ElasticSketch [22], NZE sketches [40], and BitSense [43]. Zhao *et al.* [39] propose the SuMax sketch to record both sum features and maximum features on the data plane. FlyMon [49] abstracts four frequently used attributes, and supports on-the-fly reconfiguration between these attributes. However, none of the above solutions support capturing average features on the data plane.

In-network classification: Many researchers focus on in-network classification. Among them, the decision tree and its variants well fit the switch architecture, and have been widely studied [50], [51], [52], [53], [54], [55], [23], [56]. The state-of-the-art solution NetBeacon [23] translates the model into multiple feature tables for implementation. It differentiates short and long flows to reduce storage overhead of flow-level features. Specifically, NetBeacon only records flow-level features for long flows, and uses purely per-packet features for short flows. Besides, some researchers attempt to design non-tree-based in-network classifiers, such as neural networks, on the data plane [57], [58], [59], [60], [61], [62]. However, deploying these models faces hardware limitations of the programmable switches, so there is a trade-off between accuracy and resource footprint. The main difference in our work is the introduction of sketches for feature extraction, which improves the accuracy of flow-level features and reduces the impact of error caused by hash conflicts.

RDMA on programmable switches: Many efforts deploy RDMA on programmable switches to accelerate various network applications. Ribosome [18] aims to overcome the processing bottleneck of network function (NF) servers. They use a programmable switch to split the packet headers from the payloads, delegate the payloads on RDMA servers, and only forward the headers to NF servers. DART [63] and DTA [64] propose to encapsulate telemetry data as RDMA messages on programmable switches to relieve CPU from processing incoming telemetry reports. TEA [24] implements RDMA in the data plane to enable the programmable switches to access external DRAM without involving CPUs. SwitchML [65] designs protocols to perform in-network aggregation on programmable switches at line rate. It implements a subset of RDMA in the switch, and uses RDMA write immediate messages for all communication to allow data to move directly between the switch and GPUs. The scenarios and the design goals of these solutions are completely different from ours. The difference between our work and other works is that we introduce a packet splitting mechanism. This allows us to add checksum to the back of each variable-length RDMA write message and detect if packet loss has occurred.

XI. Conclusion

In this paper, we propose TitanLog to log high-speed, high-volume, and dynamic network traffic. We propose hierarchical logging to reduce log volume, and we propose a switch-server co-design to achieve fast logging. Additionally, we present mechanisms to make TitanLog adapt to changes in logging demands and the network environments at runtime. TitanLog supports logging 100Gbps traffic with zero packet loss rate, and significantly reduces the log volume by up to 96.28%.

Acknowledgment

The authors would like to thank their editor and the anonymous reviewers for their thoughtful feedback. The authors would also like to thank Prof. Zhuotao Liu for his thoughtful suggestions. This work was supported by the National Key Research and Development Program of China under Grant No. 2024YFB2906602, and in part by the National Natural Science Foundation of China (NSFC) (No. 62372009, 623B2005).

References

[1] Sarang Dharmapurikar, Praveen Krishnamurthy, Todd Sproull, and John Lockwood. Deep packet inspection using parallel bloom filters. In *11th Symposium on High Performance Interconnects, 2003. Proceedings.*, pages 44–51. IEEE, 2003.

[2] Caleb C Noble and Diane J Cook. Graph-based anomaly detection. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636, 2003.

[3] Ravi Netravali, Anirudh Sivaraman, Somak Das, Ameesh Goyal, Keith Winstein, James Mickens, and Hari Balakrishnan. Mahimahi: Accurate record-and-replay for http. In *Usenix annual technical conference*, pages 417–429, 2015.

[4] Sultan Mahmud Sajal, Rubaba Hasan, Timothy Zhu, Bhuvan Urgaonkar, and Siddhartha Sen. Tracesplitter: a new paradigm for downscaling traces. In *EuroSys*, pages 606–619, 2021.

[5] Manish Joshi and Theyazn Hassn Hadi. A review of network traffic analysis and prediction techniques. *arXiv preprint arXiv:1507.05722*, 2015.

[6] R Vinayakumar, KP Soman, and Prabaharan Poornachandran. Applying deep learning approaches for network traffic prediction. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 2353–2358. IEEE, 2017.

[7] Shane Alcock, Perry Lorier, and Richard Nelson. Libtrace: A packet capture and analysis library. *ACM SIGCOMM Computer Communication Review*, 42(2):42–48, 2012.

[8] Tcpdump like xdp packet capture. https://github.com/cloudflare/xdpcap.

[9] Paul Emmerich, Maximilian Pudelko, Sebastian Gallenmüller, and Georg Carle. Flowscope: Efficient packet capture and storage in 100 gbit/s networks. In *2017 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 1–9. IEEE, 2017.

[10] Antonis Papadogiannakis, Michalis Polychronakis, and Evangelos P Markatos. Scap: Stream-oriented network traffic capture and analysis for high-speed networks. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 441–454, 2013.

[11] Víctor Uceda, Miguel Rodríguez, Javier Ramos, José Luis García-Dorado, and Javier Aracil. Selective capping of packet payloads at multi-gb/s rates. *IEEE Journal on Selected Areas in Communications*, 34(6):1807–1818, 2016.

[12] Rui Miao, Rahul Potharaju, Minlan Yu, and Navendu Jain. The dark menace: Characterizing network-based attacks in the cloud. In *Proceedings of the 2015 Internet Measurement Conference*, pages 169–182, 2015.

[13] Yu Zhou, Chen Sun, Hongqiang Harry Liu, Rui Miao, Shi Bai, Bo Li, Zhilong Zheng, Lingjun Zhu, Zhen Shen, Yongqing Xi, et al. Flow event telemetry on programmable data plane. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 76–89, 2020.

[14] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. *ACM SIGCOMM Computer Communication Review*, 40(1):92–99, 2010.

[15] Vijay Kumar Adhikari, Sourabh Jain, and Zhi-Li Zhang. Youtube traffic dynamics and its interplay with a tier-1 isp: An isp perspective. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 431–443, 2010.

[16] Yingying Chen, Sourabh Jain, Vijay Kumar Adhikari, Zhi-Li Zhang, and Kuai Xu. A first look at inter-data center traffic characteristics via yahoo! datasets. In *2011 Proceedings IEEE INFOCOM*, pages 1620–1628. IEEE, 2011.

[17] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. Netcache: Balancing key-value stores with fast in-network caching. In *Proceedings of the 26th symposium on operating systems principles*, pages 121–136, 2017.

[18] Mariano Scazzariello, Tommaso Caiazzi, Hamid Ghasemirahni, Tom Barbette, Dejan Kostic, and Marco Chiesa. A high-speed stateful packet processing approach for tbps programmable switches. In *Proceedings of the 20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023.

[19] InfiniBand Trade Association et al. Supplement to infiniband architecture specification volume 1 release 1.2. 1 annex a16: Rdma over converged ethernet (roce), 2010.

[20] InfiniBand Trade Association et al. Supplement to infiniband architecture specification volume 1 release 1.2. 1 annex a17: Rocev2, 2014.

[21] Jiarong Xing, Kuo-Feng Hsu, Matty Kadosh, Alan Lo, Yonatan Piasetzky, Arvind Krishnamurthy, and Ang Chen. Runtime programmable switches. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 651–665, 2022.

[22] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic sketch: Adaptive and fast network-wide measurements. In *Proceedings of the 2018 Conference of the ACM SIGCOMM*, pages 561–575. ACM, 2018.

[23] Guangmeng Zhou, Zhuotao Liu, Chuanpu Fu, Qi Li, and Ke Xu. An efficient design of intelligent network data plane. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 6203–6220, 2023.

[24] Daehyeok Kim, Zaoxing Liu, Yibo Zhu, Changhoon Kim, Jeongkeun Lee, Vyas Sekar, and Srinivasan Seshan. Tea: Enabling state-intensive network functions on programmable switches. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 90–106, 2020.

[25] P4-16 Language Specification. https://p4.org/p4-spec/docs/P4-16-v1.2.4.html.

[26] Barefoot tofino: World's fastest p4-programmable ethernet switch asics. https://barefootnetworks.com/products/brief-tofino/.

[27] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1:108–116, 2018.

[28] Data plane development kit. http://doc.dpdk.org/guides-18.02/.

[29] Tal Shapira and Yuval Shavitt. Flowpic: Encrypted internet traffic classification is as easy as image recognition. In *IEEE INFOCOM 2019-IEEE conference on computer communications workshops (INFOCOM WKSHPS)*, pages 680–687. IEEE, 2019.

[30] Aristide Tanyi-Jong Akem, Guillaume Fraysse, and Marco Fiore. Encrypted traffic classification at line rate in programmable switches with machine learning. In *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, pages 1–9. IEEE, 2024.

[31] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. Hpcc: High precision congestion control. In *Proceedings of the ACM special interest group on data communication*, pages 44–58. 2019.

[32] Yang Guo. A review of machine learning-based zero-day attack detection: Challenges and future directions. *Computer communications*, 198:175–185, 2023.

[33] Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1), 2005.

[34] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. *Automata, languages and programming*, 2002.

[35] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 101–114, 2016.

[36] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. Flowradar: A better netflow for data centers. In *NSDI*, 2016.

[37] Qun Huang, Patrick PC Lee, and Yungang Bao. Sketchlearn: relieving user burdens in approximate measurement with automated statistical inference. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 576–590. ACM, 2018.

[38] Xiaoqi Chen, Shir Landau-Feibish, Mark Braverman, and Jennifer Rexford. Beaucoup: Answering many network traffic queries, one memory update at a time. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 226–239, 2020.

[39] Yikai Zhao, Kaicheng Yang, Zirui Liu, Tong Yang, Li Chen, Shiyi Liu, Naiqian Zheng, Ruixin Wang, Hanbo Wu, Yi Wang, et al. {LightGuardian}: A {Full-Visibility}, lightweight, in-band telemetry system using sketchlets. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 991–1010, 2021.

[40] Qun Huang, Siyuan Sheng, Xiang Chen, Yungang Bao, Rui Zhang, Yanwei Xu, and Gong Zhang. Toward nearly-zero-error sketching via compressive sensing. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, April 2021.

[41] Ruijie Miao, Yinda Zhang, Zihao Zheng, Ruixin Wang, Ruwen Zhang, Tong Yang, Zaoxing Liu, and Junchen Jiang. Cocosketch: High-performance sketch-based measurement over arbitrary partial key query. *IEEE/ACM Transactions on Networking*, 31(6):2653–2668, 2023.

[42] Kaicheng Yang, Sheng Long, Qilong Shi, Yuanpeng Li, Zirui Liu, Yuhan Wu, Tong Yang, and Zhengyi Jia. Sketchint: Empowering int with towersketch for per-flow per-switch measurement. *IEEE Transactions on Parallel and Distributed Systems*, 2023.

[43] Rui Ding, Shibo Yang, Xiang Chen, and Qun Huang. Bitsense: Universal and nearly zero-error optimization for sketch counters with compressive sensing. In *Proceedings of the ACM SIGCOMM 2023 Conference*, pages 220–238, 2023.

[44] Kaicheng Yang, Yuhan Wu, Ruijie Miao, Tong Yang, Zirui Liu, Zicang Xu, Rui Qiu, Yikai Zhao, Hanglong Lv, Zhigang Ji, and Gaogang Xie. Chamelemon: Shifting measurement attention as network state changes. In *Proceedings of the 2023 ACM SIGCOMM Conference*. ACM, 2023.

[45] Zhuochen Fan, Zhoujing Hu, Yuhan Wu, Jiarui Guo, Sha Wang, Wenrui Liu, Tong Yang, Yaofeng Tu, and Steve Uhlig. Pisketch: Finding persistent and infrequent flows. *IEEE/ACM Transactions on Networking*, 31(6):3191–3206, 2023.

[46] Zhuochen Fan, Xiangyuan Wang, Xiaodong Li, Jiarui Guo, Wenrui Liu, Haoyu Li, Sheng Long, Zheng Zhong, Tong Yang, Xuebin Chen, et al. Steadysketch: A high-performance algorithm for finding steady flows in data streams. *IEEE/ACM Transactions on Networking*, 32(6):5004–5019, 2024.

[47] Jiarui Guo, Boxuan Chen, Kaicheng Yang, Tong Yang, Zirui Liu, Qiuheng Yin, Sha Wang, Yuhan Wu, Xiaolin Wang, Bin Cui, et al. Hourglasssketch: An efficient and scalable framework for graph stream summarization. In *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, pages 114–127. IEEE Computer Society, 2025.

[48] Hun Namkung, Zaoxing Liu, Daehyeok Kim, Vyas Sekar, and Peter Steenkiste. {SketchLib}: Enabling efficient sketch-based monitoring on programmable switches. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 743–759, 2022.

[49] Hao Zheng, Chen Tian, Tong Yang, Huiping Lin, Chang Liu, Zhaochen Zhang, Wanchun Dou, and Guihai Chen. Flymon: enabling on-the-fly task reconfiguration for network measurement. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 486–502, 2022.

[50] Bruno Missi Xavier, Rafael Silva Guimarães, Giovanni Comarela, and Magnos Martinello. Programmable switches for in-networking classification. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2021.

[51] Coralie Busse-Grawitz, Roland Meier, Alexander Dietmüller, Tobias Bühler, and Laurent Vanbever. pforest: In-network inference with random forests. *arXiv preprint arXiv:1909.05680*, 2019.

[52] Jong-Hyouk Lee and Kamal Singh. Switchtree: in-network computing and traffic analyses with random forests. *Neural Computing and Applications*, pages 1–12, 2020.

[53] Zhaoqi Xiong and Noa Zilberman. Do switches dream of machine learning? toward in-network classification. In *Proceedings of the 18th ACM workshop on hot topics in networks*, pages 25–33, 2019.

[54] Changgang Zheng, Zhaoqi Xiong, Thanh T Bui, Siim Kaupmees, Riyad Bensoussane, Antoine Bernabeu, Shay Vargaftik, Yaniv Ben-Itzhak, and Noa Zilberman. Iisy: Hybrid in-network classification using programmable switches. *IEEE/ACM Transactions on Networking*, 2024.

[55] Guorui Xie, Qing Li, Yutao Dong, Guanglin Duan, Yong Jiang, and Jingpu Duan. Mousika: Enable general in-network intelligence in programmable switches by knowledge distillation. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pages 1938–1947. IEEE, 2022.

[56] Syed Usman Jafri, Sanjay Rao, Vishal Shrivastav, and Mohit Tawarmalani. Leo: Online {ML-based} traffic classification at {Multi-Terabit} line rate. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1573–1591, 2024.

[57] Giuseppe Siracusano and Roberto Bifulco. In-network neural networks. *arXiv preprint arXiv:1801.05731*, 2018.

[58] Davide Sanvito, Giuseppe Siracusano, and Roberto Bifulco. Can the network be the ai accelerator? In *Proceedings of the 2018 Morning Workshop on In-Network Computing*, pages 20–25, 2018.

[59] Thi-Nga Dao, Van-Phuc Hoang, Chi Hieu Ta, et al. Development of lightweight and accurate intrusion detection on programmable data plane. In *2021 International Conference on Advanced Technologies for Communications (ATC)*, pages 99–103. IEEE, 2021.

[60] Tushar Swamy, Alexander Rucker, Muhammad Shahbaz, Ishan Gaur, and Kunle Olukotun. Taurus: a data plane architecture for per-packet ml. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1099–1114, 2022.

[61] Jinzhu Yan, Haotian Xu, Zhuotao Liu, Qi Li, Ke Xu, Mingwei Xu, and Jianping Wu. Brain-on-switch: Towards advanced intelligent network data plane via nn-driven traffic analysis at line-speed. *arXiv preprint arXiv:2403.11090*, 2024.

[62] Yinchao Zhang, Su Yao, Yong Feng, Kang Chen, Tong Li, Zhuotao Liu, Yi Zhao, Lexuan Zhang, Xiangyu Gao, Feng Xiong, et al. Pegasus: A universal framework for scalable deep learning inference on the dataplane. In *Proceedings of the ACM SIGCOMM 2025 Conference*, pages 692–706, 2025.

[63] Jonatan Langlet, Ran Ben-Basat, Sivaramakrishnan Ramanathan, Gabriele Oliaro, Michael Mitzenmacher, Minlan Yu, and Gianni Antichi. Zero-cpu collection with direct telemetry access. In *Proceedings of the Twentieth ACM Workshop on Hot Topics in Networks*, pages 108–115, 2021.

[64] Jonatan Langlet, Ran Ben Basat, Sivaramakrishnan Ramanathan, Gabriele Oliaro, Michael Mitzenmacher, Minlan Yu, and Gianni Antichi. Direct telemetry access. In *Proceedings of the ACM SIGCOMM 2023 Conference*, 2023.

[65] Amedeo Sapio, Marco Canini, Chen-Yu Ho, Jacob Nelson, Panos Kalnis, Changhoon Kim, Arvind Krishnamurthy, Masoud Moshref, Dan RK Ports, and Peter Richtárik. Scaling distributed machine learning with in-network aggregation. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 785–808, 2021.

Yuanpeng Li received the B.S. degree in Data Science from Peking University in 2022. He is currently pursuing the Ph.D. degree in the School of Computer Science, Peking University, advised by Prof. Tong Yang. His research interests include AI for networking, network measurements, and sketches.

Xian Niu is a third-year Ph.D. student in the School of Computer Science at Beijing University of Posts and Telecommunications, Beijing, China, advised by Assoc. Prof. Yuchao Zhang. His research interests include programmable networking and large-scale data center networks in computer networks, as well as hash tables and streaming algorithms in data structures and algorithms. He received the B.S. degree in Computer Science from Peking University, Beijing, China in 2023.

Yikai Zhao received the B.S. degree in computer science from Peking University in 2020. He is currently pursuing the Ph.D. degree with the School of Computer Science, Peking University, advised by Tong Yang. His research interest lies in the intersection of distributed systems and probabilistic algorithms. He published papers in SIGMOD, SIGCOMM, NSDI, etc.

Tong Yang is an associate professor in the School of Computer Science, Peking University. His research interests focus on data structures in networking, LLM, and data bases. He published 24 papers in SIGCOMM, SIGKDD, SIGMOD, and published 22 Trans on papers. He also served as the chair of international conferences and the editor of journals for several times. Many of his researches have been deployed in industry field, including Redis databases and ByteDance, etc.

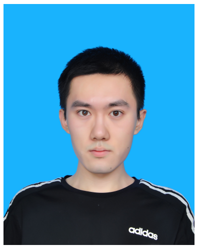
Yannan Hu received the B.S. degree from Harbin Institute of Technology, Harbin, China in 2008 and Ph.D. degree from Beijing University of Posts and Telecommunications, Beijing, China in 2015. He is currently an Associate Researcher at Beijing Zhongguancun Laboratory. His research interests include network security, network architecture and machine learning.

Yuchao Zhang (Member, IEEE) received the B.S. degree in computer science and technology from Jilin University in 2012 and the Ph.D. degree from the Department of Computer Science, Tsinghua University, in 2017. She is currently an Associate Professor with the Beijing University of Posts and Telecommunications, Beijing, China, and a Visiting Scholar with the University of Cambridge, where she is also a Research Associate at the Wolfson College. Her research interests include large scale datacenter networks, federated learning, data-driven networks, and edge computing. She is a member of ACM.

Xiangwei Deng received his B.S. in Computer Science from Beijing University of Posts and Telecommunications in 2023. He is now pursuing his Master's degree at the School of Computer Science, Peking University, under the supervision of Professor Tong Yang. His research interests include network measurement and large language models (LLMs).

Qiuheng Yin received the B.S. degree in computer science from Peking University in 2025. He is currently a graduate student at Peking University specializing in computer networks. His research interests include network measurement, network security, and in-network computing.

Ziyun Zhang received the B.S. degree in computer science in 2024 from the school of EECS in Peking University. He is currently a first-year Ph.D. student in the School of Computer Science of Peking University, advised by Associate Professor (with tenure) Guojie Luo. His research interest includes probabilistic data structures, parallel computing, and heterogeneous computing.



Ruwen Zhang received his B.S. degree in Mathematics from Peking University in 2021 and completed his M.S. degree in Computer Science from Peking University in 2024. His research focuses on network algorithms and data stream processing. He has contributed to multiple publications in the fields of networking and data measurement.

Yisen Hong is currently pursuing a Ph.D. at the Department of Computer Science and Technology, Tsinghua University. His research focuses on sketch algorithms and network measurement. He places strong emphasis on academic writing and publication, and is committed to translating research findings into practical, real-world applications.

Kaicheng Yang received his B.S. degree in Computer Science from Peking University in 2021. He is currently pursuing the Ph.D. degree in the School of Computer Science, Peking University, advised by Tong Yang. His research interests include network measurement and programmable data plane.

Ruijie Miao is currently pursuing the Ph.D. degree with the School of Computer Science, Peking University, under the supervision of Tong Yang. His research interests include network measurement and streaming algorithms.

Kun Meng received the Ph.D. degree in communication and information systems from the University of Science and Technology Beijing (USTB), Beijing, China, in 2012. He conducted postdoctoral research at the Department of Computer Science and Technology, Tsinghua University, Beijing, China, from 2012 to 2014. He is currently an Associate Professor with the College of Computer Science, Beijing Information Science and Technology University (BISTU), Beijing, China. He also serves as chief scientist researcher of LUISUAN Tech Company, Guangdong, China, since 2022. His research interests include network security and performance evaluation, identity authentication, data organization and access optimization, and decentralized networking applications.

Dahui Wang is currently the Technical Director with Guangdong Green Computing Technology Co., Ltd., Guangzhou, China. His research interests include Ethernet Bunch of Flash (EBOF) architecture, BMC management software, and FPGA-based storage acceleration. He has played a key role in the development of high-performance commercial EBOF systems and 100G FPGA target cards.

Yong Cui received the B.E. and Ph.D. degrees from Tsinghua University, China, in 1999 and 2004, respectively. He is currently a Full Professor with the Department of Computer Science and Technology, Tsinghua University. He has published more than 100 academic articles in refereed journals and conferences. He has authored a number of RFCs. His major research interests include mobile/wireless internet and network architecture. He received two best paper awards. He also serves as the Co-Chair for IETF IPv6 Transition Software WG.