SuperFE: A Scalable and Flexible Feature Extractor for ML-based Traffic Analysis Applications

Menghao Zhang[1]*, Guanyu Li[2]*, Cheng Guo[2], Renyu Yang[1], Shicheng Wang[2], Han Bao[2], Xiao Li[2], Mingwei Xu[2], Tianyu Wo[1], Chunming Hu[1]
[1]Beihang University [2]Tsinghua University

Abstract

The feature extractor component in today's ML-based traffic analysis applications is becoming a key bottleneck. While mainstream software-based approaches can support flexible feature extraction, they fail to scale to multi-100Gbps network speed easily. Meanwhile, hardware-accelerated solutions can scale to high throughput, but cannot flexibly support generic traffic analysis applications. In this paper, we propose SuperFE, a feature extraction framework that allows users to extract traffic features efficiently and flexibly. SuperFE leverages the capabilities of both new-generation programmable switches and SmartNICs, with three key designs. First, SuperFE presents a user-friendly and extensible interface to support customized feature extraction policies, shielding underlying hardware implementation details and complexities. Second, SuperFE introduces a high-performance multi-granularity key-vector cache system in the programmable switches to batch necessary feature metadata for massive amounts of packets. Third, SuperFE exploits the multi-core parallel and hierarchical memory of SoC-based SmartNICs to achieve efficient feature computation with diverse streaming algorithms. Evaluations using our prototype demonstrate that SuperFE enables various state-of-the-art traffic analysis applications to efficiently extract features from multi-100Gbps raw traffic without compromising detection accuracy, and achieves nearly two orders of magnitude higher throughput than the software-based counterparts.

CCS Concepts: • **Networks** → **Programmable networks**; **In-network processing**; **Network security**.

*Both authors contributed equally to this work.

EuroSys '25, March 30–April 3, 2025, Rotterdam, Netherlands

ACM ISBN 979-8-4007-1196-1/25/03
https://doi.org/10.1145/3689031.3696081

Keywords: Traffic Analysis, Feature Extractor, Programmable Switches, SmartNICs

ACM Reference Format:
Menghao Zhang, Guanyu Li, Cheng Guo, Renyu Yang, Shicheng Wang, Han Bao, Xiao Li, Mingwei Xu, Tianyu Wo, Chunming Hu. 2025. SuperFE: A Scalable and Flexible Feature Extractor for ML-based Traffic Analysis Applications. In *Twentieth European Conference on Computer Systems (EuroSys '25), March 30–April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3689031.3696081

1 Introduction

Traffic analysis [25, 46] refers to a class of applications that infer sensitive information to identify malicious behaviors from network communication patterns. With the prevalence of encryption and other evasion techniques that make traditional payload-based analysis of network traffic infeasible, machine learning (ML), especially deep learning, is becoming the *de facto* solution to achieve superior accuracy [1, 41, 45, 55, 61]. By using ML-based traffic analysis techniques, network administrators can identify cybercriminals (e.g., botmasters) that proxy their attack traffic via compromised machines or public relays in order to conceal their identities [29, 53, 64, 74, 79], pinpoint individuals engaged in illegal activities within privacy technologies such as Tor (The Onion Router) [24, 49, 68], detect covert channel attacks that exfiltrate confidential information from compromised machines [10, 21], and prevent malicious activities that intrude the network [41].

A common ML-based traffic analysis application [25, 46] typically comprises two components: a *feature extractor* that extracts necessary traffic features – in the form of *feature vectors* – from raw network traffic, and a *behavior detector* component that leverages ML algorithms to identify desired network behaviors. For instance, Kitsune [41], a neural network-based network intrusion detection system, first employs a feature extraction framework to obtain 115-dimension traffic feature vectors with incremental statistics over a damped window, and then uses an online detection algorithm (an ensemble of autoencoders) to take feature vectors as input to detect abnormal packets that have high root mean squared error (RMSE) values. As another example, a website fingerprinting approach on Tor [45] extracts a set of features from raw network traffic and uses a k-NN classifier to identify which website an individual accesses.

However, a closer look into today's ML-based traffic analysis applications shows that the feature extractor components are becoming a key bottleneck. While various studies (e.g., via GPUs [12], TPUs [32]) are manifested to accelerate ML-based behavior behaviors, the status quo of feature extractors has not experienced much progress. Existing mainstream feature extractors [29, 41, 46, 68] usually use port mirroring to duplicate collected network traffic, and leverage a large set of servers to store large volumes of network traffic and extract the desired traffic features. These software-based solutions incur substantial communication, storage, and computation overheads, especially for multi-100Gbps networks. Some recent works [8, 15, 28, 59, 71, 73, 81] attempt to accelerate the feature extractor with specialized network hardware, and implement the feature extractor, even the entire traffic analysis task in the data plane, which significantly reduces the requirements for extra bandwidth and servers. However, restricted by the computational model and memory resources of the network hardware, none of them are flexible enough to support generic and versatile traffic analysis applications.

In this paper, we present SuperFE, a feature extraction framework that enables the generation of desired traffic features efficiently and flexibly, and provisions feature extraction as a service to ML-based traffic analysis applications. By leveraging the capabilities of new-generation programmable switches and SmartNICs, SuperFE is able to extract feature vectors from multi-100Gbps network traffic efficiently. Besides, by providing expressive policy interfaces and designing smart data structures and algorithms in the underlying programmable switches and SmartNICs, SuperFE is able to support generic ML-based traffic analysis applications flexibly.

In particular, SuperFE's improved performance and flexibility are derived from the following three key techniques. First, SuperFE presents a user-friendly and extensible interface to help users write their own feature extraction policies, without regard to underlying hardware implementation and complexities in programmable switches and SmartNICs. Second, SuperFE introduces a high-performance multi-granularity key-vector cache system in the programmable switches to batch necessary feature metadata for massive amounts of packets, in order to significantly reduce the workload for SmartNICs. Third, SuperFE exploits the multi-core parallel and hierarchical memory of SoC-based SmartNICs to achieve efficient feature computation with streaming algorithms, in order to produce feature vectors based on batched metadata from the switches. We implement a proof-of-concept prototype of SuperFE and conduct extensive experiments. Evaluation results show that SuperFE empowers various state-of-the-art traffic analysis applications to efficiently extract features from multi-100Gbps raw traffic with nearly no detection accuracy degradation, and also exhibits nearly two orders of magnitude higher throughput than their software-based counterparts. To the best of our knowledge, although programmable switches and SmartNICs have been used together in some certain traffic analysis applications [50], SuperFE is the first research work that leverages both of them to support generic ML-based traffic analysis applications.

In summary, this paper makes the following contributions:

- We show the background of ML-based traffic analysis applications, and illustrate the design goals for an ideal feature extractor (§2).
- We propose SuperFE, a feature extraction framework that enables users to generate traffic features efficiently and flexibly (§3). SuperFE provides a set of policy interfaces for users to express the traffic feature extraction policy flexibly (§4), a high-performance multi-granularity key-vector cache system to batch packet feature metadata in programmable switches (§5), and a set of efficient streaming algorithms to compute feature vectors in SmartNICs (§6).
- We implement a prototype of SuperFE, and conduct extensive experiments to demonstrate the flexibility and performance of SuperFE (§7, §8).

Finally, we make some discussions in §9, summarize the related works in §10, and conclude this paper in §11.

2 Background and Motivation

In this section, we introduce the background of ML-based traffic analysis applications, and present the design goals and how existing systems fall short that motivates this work.

2.1 ML-based Traffic Analysis

Herein we review a few popular types of ML-based traffic analysis and discuss the pertaining applications. Traffic analysis that focuses on application-layer protocol semantics without ML algorithms [66] is out of the scope of this paper.

Botnet Detection. Botnets are severe threats to organizations, and they are becoming more difficult to be taken down with the emergence of decentralized P2P and stealthy communications. To mitigate this, researchers have proposed to identify bots through analysis of packet sizes and packet time intervals [43]. These techniques can help network administrators prevent and mitigate stealthy botnet threats effectively.

Website Fingerprinting. Privacy-enhancing technologies such as VPNs and Tor enable attackers to hide their source or destination IP addresses and the content of the visited websites, posing significant challenges for authorities in terms of accountability. Fortunately, website fingerprinting is a promising technique to identify which websites attackers access by feeding the collected traffic features to ML pipelines [24, 49, 68]. This can assist authorities in pinpointing individuals engaged in illegal activities.

Covert Channel Detection. Attackers can exfiltrate confidential information from compromised machines in organizations through timing covert channels, without being detected by classic firewalls or intrusion detection systems. Nevertheless,

network administrators can still uncover stealthy communication patterns by capturing packet time intervals and analyzing them with ML techniques [10, 21], which is invaluable for protecting the property of assets in organizations.

Intrusion Detection. Network intrusion detection systems are of great importance to monitor traffic for malicious activities. State-of-the-art intrusion detection systems extract contextual features from network traffic and use ML algorithms to differentiate normal and abnormal traffic patterns [20, 41], which is crucial to guarantee the security of the protected networks.

From these applications, we can see that feature extractor plays a crucial role in the procedure of ML-based traffic analysis. Although we have seen a trend from manual feature engineering to fully learned feature extraction by neural networks in the deep learning field, fully learned feature extraction does not mean that raw packets can be input into neural networks directly. Some typical feature extraction procedure is still required to convert raw packets into feature vectors that are ready to be used by deep learning algorithms, such as packet arrival times, sizes, directions, etc. As a result, we believe a powerful feature extractor is necessary to enable next-generation ML-based traffic analysis applications.

2.2 Design Goals

We aim to build a feature extractor solution that can achieve the following two design goals.

Scalable to multi-100Gbps with low overheads. Recently the network bandwidth at the network aggregation points in regional ISPs has already reached multi-100s of Gbps [54, 70]. Many network device providers [13, 48] and standard organizations [2] are embracing the era of 800Gbps network speed. Such a high network bandwidth necessitates the feature extractor to extract feature vectors from raw traffic whilst keeping pace with the traffic volume. However, current mainstream software-based feature extraction approaches [29, 41, 46, 68] usually use port mirroring at switches to steer network traffic, and also require a set of servers to store the intercepted traffic and extract the desired feature vector, which incurs considerable communication, computation, and storage overheads. Besides, such approaches usually make an after-the-fact analysis assumption [11] that is not real-time and cannot catch malicious network activities timely. While recent proposal [46] tries to alleviate this problem by compressing traffic features with linear projection compression algorithms, it is still difficult, and often infeasible, to scale to multi-100Gbps with this pure algorithmic solution. Hence, it is imperative for the feature extractor to be scalable to multi-100Gbps traffic processing in real-time with low overheads.

Flexible to support generic traffic analysis applications. Recently researchers are turning to extract more ingenious features from raw network traffic and devise more powerful behavior detection algorithms to achieve satisfactory detection results. For example, in intrusion detection, researchers have extracted 115-dimension features from each packet to identify malicious activities; in website fingerprinting, researchers have turned from traditional machine learning techniques [24, 49, 68] to deep learning algorithms [1, 55, 60, 61] to achieve better classification results. However, current hardware-accelerated feature extractors lack the generality to support various traffic analysis algorithms and applications flexibly. For example, FlowLens [8] only supports the feature vector of packet length and inter time arrival distribution, N3IC [59], Mousika [71], NetBeacon [81], HorusEys [15], Leo [28] and BoS [73] merely supports one specific simplified neural network. We aim to build our feature extractor to be generic in terms of feature extraction and behavior detection. In this way, users can run custom machine learning algorithms on user-defined features to conduct their traffic analysis tasks.

3 SuperFE Overview

3.1 Design Principle

To realize the design goals above, we introduce SuperFE, a feature extraction framework that allows users to extract traffic features efficiently and flexibly. The key technology enabler is the new-generation programmable switches and SmarNICs, which offer us an unprecedented opportunity to achieve scalability and flexibility with low overheads. Programmable switches are designed to process traffic at multiple Tbps, but have a restrictive programming model and limited memory resources [36, 62, 63, 78]; SmartNICs are able to flexibly support more powerful programming models with hierarchical memory, but can only scale to tens of Gbps traffic [26, 50]. At the core of SuperFE is to leverage SmartNICs' capability of programmability in supporting more operations [47] to complement the programmable switches' scalability in terabit packet processing [27].

Based on the study of the feature extractor component in numerous traffic analysis applications, SuperFE chooses to place simple, basic, frequent feature extraction operations in programmable switches and put complex, computation/memory-intensive operations in SmartNICs. In other words, SuperFE uses programmable switches to select and aggregate the basic packet feature metadata from raw network traffic, and employs SmartNICs to process the basic feature metadata into feature vectors that are directly usable for ML algorithms. The selection and aggregation operations on switches can also reduce the traffic transferred to SmartNICs significantly.

3.2 Deployment Scenario and Workflow

SuperFE is devised to be the feature extractor of an ML-based traffic analysis application, which can be used by administrators of an ISP or cloud network to infer sensitive information from raw network traffic. Based on the application requirements, SuperFE can be deployed into one or multiple locations of the target network. At each location, as shown in Figure 1, one switch is connected with multiple

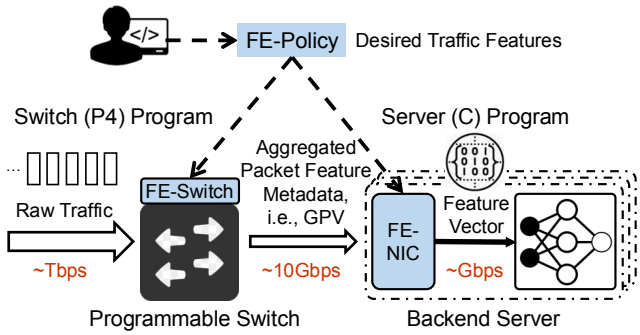

Figure 1. The overall workflow and architecture of SuperFE.

SmartNICs (in one or multiple servers) regarded as a whole, and aggregated packet feature metadata is transferred in between. The input to SuperFE is raw network traffic to the switches, and the output of SuperFE are feature vectors from the SmartNICs. SuperFE does not use port mirroring to duplicate the raw network traffic; instead, the switches in SuperFE first preserve the functionality of packet switching and then behave as a traffic feature aggregator to batch feature metadata. Note that the deployment of programmable switches and SmartNICs is not a new requirement – several ISPs/cloud networks have already deployed such programmable network devices in their networks [6, 56, 57], which we believe can also be used to support traffic analysis tasks.

Figure 1 depicts the workflow of SuperFE. The procedure is similar to the classic feature extractor component of traffic analysis applications. First, based on the property of the traffic analysis application, network administrators should specify the traffic features they want to extract and the approaches to extract these features from raw network traffic using the SuperFE policy interfaces. Then, SuperFE translates the policy into data structures and algorithms running in the underlying programmable switches and SmartNICs (named *FE-Switch* and *FE-NIC*, respectively). Finally, after the raw traffic flows through the switches in SuperFE, the feature vectors will be evicted from the SmartNIC of SuperFE, ready to be used for the behavior detector of the traffic analysis application.

During the full workflow of SuperFE, there are three key designs that help SuperFE achieve scalability and flexibility simultaneously. First, SuperFE presents a user-friendly and extensible interface to help users write their own feature extraction policies, shielding the underlying hardware implementation and complexities in programmable switches and SmartNICs (§4). Second, SuperFE introduces a high-performance multi-granularity key-vector cache system in the programmable switches to batch feature metadata for massive amounts of packets, thereby significantly reducing the workload for SmartNICs (§5). Third, SuperFE exploits the multi-core parallel and hierarchical memory of SoC-based SmartNICs to achieve efficient feature computation with streaming algorithms, which can produce feature vectors based on batched metadata from the switches (§6).

Table 1. Summary of key operators in the SuperFE interface.

Operator	Description
groupby(g)	Group input packet/group tuples by the granularity g
filter(p)	Filter input packet/group tuples satisfying predicate p
map(d, s, mf)	Apply mapping function mf to s and emit d for each member tuple
reduce(s, [rf])	Conduct a set of reducing functions [rf] to aggregated s for each input tuple
synthesize(sf)	Utilize synthesizing function sf to process features generated by reduce
collect(u)	Indicate SuperFE to produce the final feature vector with features from reduce and synthesize per unit u

4 Programming Feature Extractor

SuperFE presents a high-level interface to help users write their own feature extraction policies. Instead of exposing low-level interfaces that normally require expertise in hardware details, e.g., P4 for programmable switches or Micro-C for SmartNICs, SuperFE learns from modern stream processing to enable users to easily apply familiar dataflow operators (e.g., groupby, map, reduce) over packet streams without regard to their implementation. Users can also extend the interface to support more feature computation methods easily.

4.1 SuperFE Policy Interface

At first glance, different traffic analysis applications seem to adopt different ways to generate their feature vectors, as they require different protocol semantics and traffic characteristics, and also apply to different application scenarios. However, we observe a *common* feature extraction procedure underneath these different applications – selecting interested traffic, grouping traffic into multiple sets, mapping intermediate statistics, and producing and composing the final feature vectors. This commonality suggests an opportunity to express the feature extraction procedure with a high-level abstraction.

User-friendly streaming interface. Inspired by recent works introducing Spark-style stream processing operators [76] to network telemetry (e.g., Sonata [22], Marple [44]) and DDoS defense (e.g., Ripple [72]), we choose these functional operators as the starting point due to their high expressiveness for packet stream processing. To fit the commonality of the feature extraction in traffic analysis applications, we employ several customizations based on original dataflow operators to simplify the programming interface of SuperFE. A summary of key operators in SuperFE is shown in Table 1.

Tuple-based packet abstraction. Similar to recent works above, SuperFE also abstracts a single packet to a key-value tuple, consisting of two types of key-value pairs. One type takes the header fields of the packet as the keys, and the corresponding values come from the packet itself, e.g., source and destination IP addresses. The other type takes the metadata about the packet as keys, and the corresponding values are filled by the programmable switch, e.g., packet size and

packet arriving timestamp. SuperFE further allows users to extend this tuple abstraction to include any field that can be parsed or reached by the programmable switch. The input to a SuperFE policy is a stream of packets, i.e., a series of packet key-value tuples, represented by pktstream in the SuperFE interface. The data transferred among operators of SuperFE is also expressed in the form of key-value tuples.

Group-based stream processing. Feature extraction in traffic analysis applications is a specific kind of stream processing, indicating that SuperFE can adopt some particular optimizations on the abstraction and the interface. As traffic analysis applications observably extract and compute features for just a few common granularities, e.g., packet rate per IP address or per flow, we are motivated to propose a group-based packet stream processing model. As shown in Figure 2, the input pktstream, i.e., packet key-value tuples (aka packet tuples), can be partitioned into different groups with the operator groupby(g) by the specified granularity g. The output of groupby is also in the form of key-value tuples (aka group tuples), and each key-value tuple corresponds to a different group consisting of packet tuples with the same granularity value. Each output group tuple contains two key-value pairs: the "granularity" key-value pair, and the "member" pair whose value is a list of packet tuples included by this group. The group tuples can be further grouped with a more coarse granularity in the same way, so that traffic analysis applications can extract features like the number of TCP flows that each IP address establishes. The operator groupby(g) can be easily extended to support more group granularities g on demand. Based on the group-based processing model, users are able to intuitively describe the feature extraction logic of their traffic analysis applications.

Efficient stream operators. The operator filter(p) is used to select the packets or group tuples satisfying the predicate p from the input key-value tuples and feed them to the next operator. This can help users filter out the traffic they are interested in. Considering the unique group-based processing model of SuperFE, we confine the operation scope of other operators (map, reduce, synthesize, collect) within the group to simplify their utilization. Figure 2 presents an example application of these operators. The operator map(d, s, mf) applies the mapping function mf to the source key-value pair s and emits the new destination key-value pair d. And map conducts this mapping operation to each (packet/group) key-value tuple in the "member" list for every input group tuple. It allows users to pre-process some intermediate statistics from the original traffic for subsequent feature computation. Moreover, the feature computing is realized by the operator reduce($s, [rf]$). Specifically, it aggregates all key-value pairs s in each "member" tuple for each input group tuple and applies a set of reducing functions $[rf]$ to these key-value pairs to compute the desired features, which are added into each input group tuple as new key-value pairs. Additionally, users can utilize the operator synthesize(sf) followed by reduce to post-process the feature key-value pairs generated by reduce with the synthesizing function sf. Currently supported mapping / reducing / synthesizing functions are listed in Table 1 in Appendix A, and they can also be extended by users. SuperFE also provides the operator collect(u) to indicate how to produce the final feature vector with the obtained features earlier. If collect is called after reduce or synthesize, the obtained features will be added to the final feature vector. The parameter u can be either pkt or the "granularity" g – correspondingly, the final feature vector is produced per packet or per group. These operators are detailed with concrete examples in §4.2.

Natural support to SuperFE architecture. The policy interface provides great intuition for us to partition a SuperFE policy across the programmable switch and SmartNIC. Since groupby and filter have simple and fixed processing logic, they can be well supported by the programmable switch. More importantly, deploying these two operators on the switch can significantly reduce the traffic transferred to the SmartNIC (§5). The rest of the operators are utilized after them in the policy and are often required to use complex computing functions, which are impossible to implement with the programmable switch. Consequencely, we implement map, reduce, synthesize and collect on the SmartNIC (§6).

4.2 SuperFE Policy Examples

To illustrate the simplicity and expressiveness of SuperFE policy interface, we now describe three sample feature extracting policies that are utilized by state-of-the-art traffic analysis applications.

Basic statistical features are the most commonly used in traffic analysis applications. Figure 3 describes an example policy extracting basic statistical features for each TCP flow, which is a part of the feature extractor of a covert channel detection application [7]. This policy first filters out TCP packets and groups them by flow to get *tcp_flow* which is composed of group tuples (lines 1-3) and reused in the following codes. Then it counts the number of packets of each flow by mapping "one" to each packet and summing them for every flow (lines 5-7). Finally, it computes the mean, variance, minimum, and maximum of packet size (line 10-11) and inter-packet time (line 14-16) for each flow respectively, where the inter-packet time is figured out according to the arriving timestamp of packets. Besides, this policy instructs SuperFE to produce a feature vector composed of these features for each TCP flow (lines 8, 12, 17).

Packet frequency distributions have been proved to be critical for traffic analysis applications [8]. SuperFE supports this feature extraction by providing a specific reducing function *ft_hist*, which captures the histogram of the given data to represent its distribution. Compared with other generic reducing functions, *ft_hist* requires users to provide two parameters to specify the width and the number of bins for the histogram.

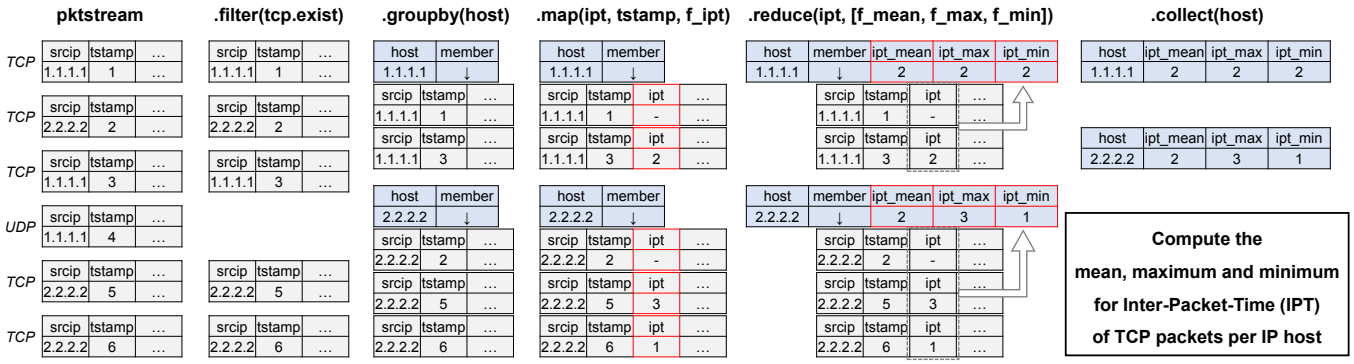

Figure 2. Example application of operators in SuperFE.

```
1  tcp_flow = pktstream
2  .filter(tcp.exist)
3  .groupby(flow)
4
5  tcp_flow
6  .map(one, _, f_one)
7  .reduce(one, [f_sum])
8  .collect(flow)
9
10 tcp_flow
11 .reduce(size, [f_mean, f_var, f_min, f_max])
12 .collect(flow)
13
14 tcp_flow
15 .map(ipt, tstamp, f_ipt)
16 .reduce(ipt, [f_mean, f_var, f_min, f_max])
17 .collect(flow)
```

Figure 3. Basic statistical features with SuperFE.

```
1  pktstream
2  .groupby(flow)
3  .map(ipt, tstamp, f_ipt)
4  .reduce(ipt, [ft_hist{10000, 100}])
5  .reduce(size, [ft_hist{100, 16}])
6  .collect(flow)
```

Figure 4. Packet frequency distributions with SuperFE.

As shown in Figure 4, this example policy selects the distribution of packet size and inter-packet time for each flow as the feature vector. It uses a histogram with 100 bins of width 10000 (ns) for inter-packet time (line 4) and a histogram with 16 bins of width 100 (bytes) for packet size (line 5).

```
1  pktstream
2  .filter(tcp.exist)
3  .groupby(flow)
4  .map(one, _, f_one)
5  .map(direction, one, f_direction)
6  .reduce(direction, [f_array])
7  .collect(flow)
```

Figure 5. Packet direction sequences with SuperFE.

Packet direction sequences are usually used as the feature representation for deep learning-based website fingerprinting [55, 60, 61]. Figure 5 presents a policy showing how to extract directional features in SuperFE. The *flow* granularity is used to preserve the directional information when conducting the group operation. Moreover, users can adopt directional

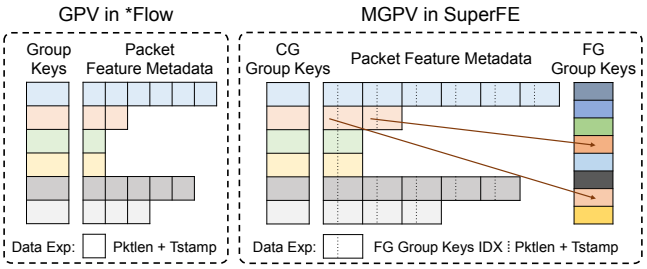

Figure 6. GPV in *Flow and MGPV in SuperFE.

mapping functions to apply the directional information. For example, this policy utilizes *f_direction* to generate "1" for ingress packets and "-1" for egress packets by multiplying "one" and the direction factor (line 5). To generate the direction sequence, this policy employs the reducing function *f_array* to just pack the given data into an array (line 6).

5 Batching Feature Metadata on Switches

At the core of SuperFE is to identify groupby and filter that appeared in the policy, generate the corresponding P4 program, and deploy the program on the switch to batch necessary feature metadata for the input traffic. Technically, the switch first parses the header fields used in the policy from the received packet, then determines whether this packet is filtered out by the policy, and finally groups and caches the required information and metadata, which are soon evicted to the SmartNIC in order. The filtering is realized with a single match-action table matching a set of header fields and converting the predicate to the rule. To achieve grouping and batching, SuperFE introduces a key-vector cache system that supports multi-granularity traffic features and achieves efficient resource utilization.

5.1 Enable Multi-Granularity Traffic Features

Recent telemetry system (*Flow [63]) proposes the *Grouped Packet Vector* (GPV) format for the storage and transmission of streamed packet metadata. As depicted in Figure 6, a GPV is a hybrid between a packet record and a flow record, and it contains a flow key (e.g., 5-tuples) and a variable-length list of packet feature metadata (e.g., packet length, timestamps).

It is demonstrated that GPV is flexible and efficient for the programmable switch to group and batch packet feature metadata for the back-end processing. However, GPV does not natively support multi-granularity packet feature metadata, which is commonly required by feature extractors of traffic analysis applications [38, 41, 80]. For example, Kitsune [41] and HEALD [80] are all designed to compute the same packet features for at least three granularities, including per *host* (i.e., the source IP), *channel* (i.e., the source IP and destination IP pair), and *socket* (i.e., the 5 tuples).

We observe that there are usually dependencies among the granularities used in these feature extractors, and multiple granularities employed in a traffic analysis application can be often modeled into a dependency chain. The existence of dependency chains is common in traffic analysis applications [38, 41, 80], as traffic analysis applications prefer to use features that are correlated with each other. For the dependency chain of granularities extracted from the feature extractor, we indicate its head and tail with the coarsest granularity (CG) and the finest granularity (FG). Since GPV only supports single granularity packet feature metadata, one naïve approach is to allocate memory for each granularity respectively, which wastes a tremendous amount of switch memory.

To address this, we propose *Multi-granularity GPV* (MGPV) to support multi-granularity packet feature metadata with efficient memory utilization, which stores only one copy of feature metadata for each packet. To achieve a certain granularity's packet grouping results, one can aggregate the corresponding groups at the finer granularity into one group. For instance, we can merge all packets from group *socket* into group *channel*. However, when each socket runs out of its buffer space, the packet metadata is evicted from the switch to the SmartNIC. Such cache eviction is driven by the traffic within each socket, so the packet metadata evicted to the NIC may not be ordered. Traffic analysis tasks accessing across these streams would then face inaccuracy caused by out-of-order packet arrival. And introducing the timestamp and complex re-ordering mechanisms like TCP to solve this disorder issue would impose extra computational burdens and communication latency on the SmartNIC.

To resolve the issues above, MGPV turns to another way, achieving packet grouping for a finer granularity based on the results from a coarser granularity. To do so, as shown in Figure 6, on the switch, packets are grouped at the CG level, while each packet's feature record points to its corresponding FG group key. These FG group keys are stored only once, from which the SmartNIC can recover grouping at intermediate granularity levels. Retaking *host*, *channel* and *socket* as an example, as shown in Figure 7, SuperFE regards *host* as the CG and *socket* as the FG, groups packets by the source IP, and stores the 5 tuples of each packet on the switch. Then on the SmartNICs, the 5 tuples of all the packets (i.e., FG group keys) will be used to split the packet feature metadata of the *host* group into the *channel* and *socket* groups respectively.

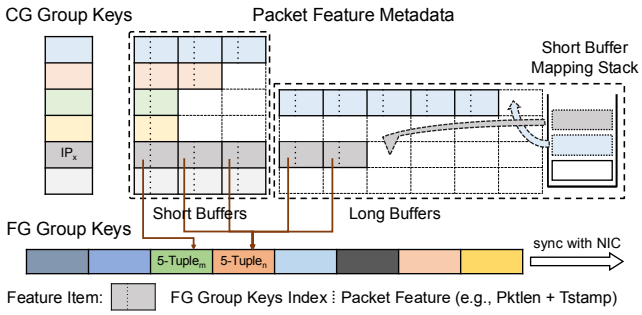

Figure 7. Memory allocation of MGPV on switch. Keys and Metadata with the same color belong to one CG group (i.e., one source IP). The stack is used to store the top pointer that tracks the long buffer that a short buffer owns. Each packet feature metadata has an index to the FG Group Key table (only show 5 brown arrows here for simplicity), and many metadata items share the same FG Group Key.

Furthermore, MGPV adopts a memory-efficient way to store these FG group keys. MGPV introduces a separate hash table to store these FG group keys for all packets, which are synchronized between the switch and SmartNIC, i.e., all changes to this table on the switch are notified to the SmartNIC for synchronous updates. The feature metadata of each packet records an additional index pointing to the FG Group Keys in the hash table, and many packet feature metadata items share the same FG Group Key. For example, in Figure 7, for the CG group key IP_x, the feature metadata of the first packet points to the FG group key 5-Tuple$_m$, and the feature metadata of the next four packets points to the FG group key 5-Tuple$_n$. Since the SmartNIC maintains an FG Group Keys table that is synchronized with the switch, the SmartNIC can easily access the FG group keys for each packet with the corresponding index and conduct group splitting and subsequent operators, when the packet vector of a group is evicted from the switch and arrives at the SmartNIC. With all the techniques above, SuperFE is able to offload groupby to the programmable switch in a memory-efficient and order-preserving manner.

5.2 Optimize Resource Utilization

We propose a series of novel mechanisms to optimize MGPV resource utilization in the programmable switch.

Memory allocation. Considering the long-tail nature of flow length distribution, inspired by *Flow [63], SuperFE adopts a hierarchical hash-based cache system to store MGPVs as shown in Figure 7. Since most flows are short flows consisting of a few packets, SuperFE allocates a short buffer for each coarsest group to store their vector feature metadata. For those long flows, SuperFE prepares a stack of long buffers to cache more packet feature metadata with MGPV, which can effectively reduce the overall rate of messages to SmartNICs. To adapt to flow length distributions, the size of long buffers is

set to be much larger than that of short buffers, while the number of long buffers is set to be much smaller than that of short buffers. When a flow fills its short buffer for the first time, it is likely to be a long flow and SuperFE attempts to pop a pointer from the stack to keep caching its packet feature metadata on the long buffer. The stack is implemented with a pointer register and a register array, which needs `resubmit` to complete its allocation and release semantic [63]. In addition to the feature metadata, SuperFE needs to allocate memory to store FG Group Keys, which is realized with a hash table. Whenever the FG Group Keys table is updated, the switch sends a notification to the SmartNIC for synchronization.

MGPV eviction. There are three cases where feature metadata in MGPV on the switch will be evicted to SmartNICs. The first and the most common case is caused by hash collision. When a packet from a group not recorded by MGPV arrives at the switch, the data will be inserted into the cache directly if the corresponding slot is empty; otherwise it will be inserted after the older group entry is evicted to the SmartNIC, including its feature metadata in the short buffer and long buffer if it owns. It is a simple yet effective approach in practice, since the collision eviction policy highly matches the procedure of LRU updating [44, 63].

The second case is when the short or long buffer is filled up. When a packet from a group recorded by MGPV arrives at the switch and happens to fill up the corresponding short buffer, a long buffer is allocated to this group if there is still one in the stack; otherwise, the feature metadata of this group in its short buffer is evicted to the SmartNIC. When a packet happens to fill up the corresponding long buffer, the feature metadata of its group in the short buffer and the long buffer is both evicted to the SmartNIC. Such an eviction policy ensures these buffers on the switch can be reused. The third case is caused by the aging mechanism designed for MGPV (see details below).

Aging mechanism. We introduce an aging mechanism for MGPV to recycle expired entries in the cache, which should be released and reused after the corresponding flow becomes inactive. To achieve this, SuperFE maintains a timestamp for each cache entry and evicts the group entry from the cache if it has not been accessed for more than a period T. The aging mechanism helps the switch keep track of real active flows, and further improves memory utilization. In particular, the occupied long buffers can be released in a timely manner and reused by other long flows, which enables SuperFE to handle more long flows and increase its memory efficiency.

However, monitoring the accessing states of all cache entries periodically on the control plane is CPU-intensive and consumes most of the control channel. To resolve this challenge, SuperFE leverages some "internal" packets which are constantly recirculated within the switch pipeline to conduct such monitoring entirely in the data plane. These packets stay in the switch through the recirculation port, check the cache entry in turn at a high frequency to determine whether it is timed out, and evict the outdated entry with its packet feature metadata in the short buffer and long buffer if it owns.

6 Computing Features on SmartNICs

SuperFE extracts the rest of the operators from the policy, i.e., map, reduce, synthesize and collect, and offloads them to SmartNICs by generating the corresponding Micro-C program. The SmartNIC receives the aggregated MGPV evicted from the switch, traverses each MGPV cell storing the separate packet feature metadata, updates the statistics of corresponding groups by applying mapping / reducing / synthesizing functions, and finally collects feature vectors as indicated and sends them to the behavior detector component. During this process, mapping, synthesizing, and collecting usually do not involve too complex arithmetic operations, which are intuitive to implement on the SmartNIC. However, reducing especially reducing functions, which are responsible for feature computing, are generally computationally complex to adapt to the architecture of SmartNIC. To achieve efficient reduction, SuperFE applies powerful streaming algorithms and optimizes their resource utilization on the SmartNIC.

6.1 Enforce Streaming Algorithms

SoC-based SmartNICs enclose programmable general-purpose SoC cores, which are specifically optimized for packet processing tasks. Compared with x86 CPU cores, these SoC cores usually have lower performance characteristics and constrained programming models. To implement reduce with limited hardware resources, we, therefore, employ techniques adapted from the streaming algorithms literature within the context of SmartNICs. With data arriving one at a time in a "stream", streaming algorithms just require restricted states and make only one pass over the stream to approximate various statistics. In this subsection, we facilitate some of the most commonly used reducing functions in SuperFE with the help of streaming algorithms.

Sum & Maximum & Minimum. *f_sum*, *f_max* and *f_min* represent the simplest reducing functions in SuperFE, and it is unnecessary to use streaming algorithms for them. They only need to save one state for each group and perform one add or compare operation for each packet feature metadata.

Mean & Variance. *f_mean* and *f_var* compute the mean and variance of a data stream $(x_1, ..., x_n)$ within the group respectively. A naïve method may involve a two-pass algorithm: the first pass applies *f_sum* to the data stream and divides the sum by the weight n to acquire the mean $\bar{x}$; the second pass iterates over the data stream again to calculate $\sum (x_i - \bar{x})^2$. Conducting two passes requires a significant amount of storage to store all entire data streams, which may exceed the capacity offered by common SmartNICs. Therefore, we turn to the streaming algorithm from Welford [69], which aims to compute the mean and estimated variance in a single pass in

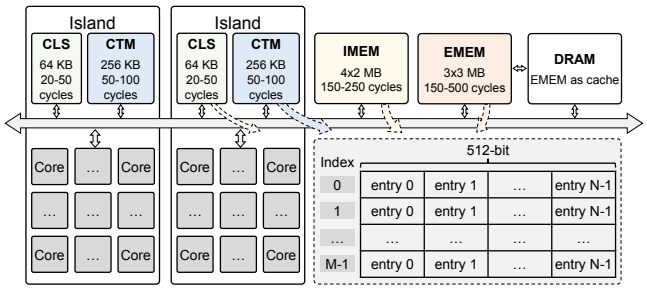

Figure 8. Netronome NFP architecture.

terms of the following formulas:

$$\bar{x}_n = \frac{(n-1)\bar{x}_{n-1} + x_n}{n} = \bar{x}_{n-1} + \frac{x_n - \bar{x}_{n-1}}{n} \tag{1}$$

$$\begin{aligned} \sigma_n^2 &= \frac{(n-1)\sigma_{n-1}^2 + (x_n - \bar{x}_{n-1})(x_n - \bar{x}_n)}{n} \\ &= \sigma_{n-1}^2 + \frac{(x_n - \bar{x}_{n-1})(x_n - \bar{x}_n) - \sigma_{n-1}^2}{n} \end{aligned} \tag{2}$$

Compared with the two-pass algorithm, Welford's online algorithm requires only a small amount of storage to record n, $\bar{x}_n$ and σ^2 for the data stream of each group at the expense of slightly increased computation costs.

Cardinality. *f_card* is used to count the number of unique elements in the given group, which is often used to figure out the number of flows or connections established by each IP. SuperFE applies the streaming algorithm adapted from HyperLogLog [18] to implement *f_card*. The basic idea is to calculate the maximum number of leading zeros in the binary hash representation of each element in the group. If the observed maximum number of leading zeros is n, an estimate for the number of distinct elements is 2^n. This approach is memory-efficient but suffers from high variance for the approximation. The variance can be minimized by splitting the incoming stream into numerous buckets, estimating the cardinality of each bucket by calculating the maximum number of leading zeros separately, and combining them by taking the mean to derive the cardinality of the whole group. In practice, SuperFE computes a 32-bit hash for each packet data, where the first k bits are used to index the bucket and the last $32 - k$ bits are used to count leading zeros. Hence, *f_card* only maintains 2^k states to record the maximum number of leading zeros of each bucket, and the exponential and division operations it performs can both be replaced with much faster shift operations.

Distribution-related features. *ft_hist*, *ft_percent* and *f_cdf* are provided to compute distribution-related features, where *ft_hist* is considered as the basis implementation for the rest of the functions. This is because the mission of *ft_hist* is to capture the histogram of the given data to approximate its distribution, and other distribution-related features can be computed from the histogram. In particular, *f_cdf* can be achieved by executing the cumulative sum to all bins of its histogram and then conducting normalization to the result, and *ft_percent* can be calculated by adding up those bins lower than that data. To realize *ft_hist*, SuperFE allocates an array of states for each group to store their bins whose width and number are specified by the parameters. Furthermore, SuperFE also conducts variable bin width [14] to improve the accuracy of features computed through the histogram.

6.2 Optimize Resource Utilization

Considering the unique hardware architecture of SoC-based Netronome NFP SmartNICs, as shown in Figure 8, we introduce several optimizations to exploit architectural features of SmartNICs to improve offloading performance.

Computational cycle optimization. Netronome NFP SmartNICs contain tens of RISC cores for parallel packet processing, and each core is capable of executing 8 hardware threads at 800 MHz. The number of cycles the processing core executes for each packet directly determines the overall performance of SmartNICs. As a consequence, we propose three optimizations dedicated to reducing the processing cycles of SuperFE. First, we reuse the hash value computed by the switch to eliminate the hash computation overhead on SmartNICs. The 32-bit hash index value used by the switch cache system is sent to the SmartNIC along with the evicted MGPV. Second, we exploit the threading mechanism to hide the memory access latency. Concretely, the cores are configured to run the same SuperFE program for all threads and switch to run another thread to process other packets when the current thread is waiting for memory access. Since context switching only takes 2 cycles, it can efficiently save cycles caused by memory access latency.

Third, we optimize the usage of complex computational operations, especially division. Considering the cores are designed for packet processing, they lack support for floating-point data types and complex operations like division. Though the compiler provides division support through algorithmic implementation, it takes 1500 cycles to perform such computation on SmartNICs [58], which significantly slows down the performance. To resolve this problem, we simplify our dependence on division with the deep analysis of streaming algorithms. Take computing mean as an example, Equation 1 involves a division operation per packet. Since $(x_n - \bar{x}_{n-1})$ is often not very large and n gets larger with more packets received, the result of this division is usually 0 or 1. So we replace this division by comparing n and $(x_n - \bar{x}_{n-1})$ to achieve the result when n is large.

Group table implementation. map, reduce and synthesize need to maintain states for each group to execute corresponding functions, so we design an efficient hash table structure for group state management on the SmartNIC. We observe that the width of the data bus between processing cores and the memory sub-system is 512-bit (64-byte), which is usually large enough to cover simultaneous state access of several groups. For instance, if the given policy requires 12-byte states for each IP host, then the entry size of its host-based

group table is 16-byte, including a 4-byte IP address and its states, which means the core can load states of 4 hosts from the memory at once. To leverage this hardware feature, we employ the hash table with fixed-length chaining to organize group states, as illustrated in Figure 8, which enables both fast parallel lookup and low collision rate [77]. The width of the group table, i.e., the number of entries at each index, is determined by users based on their requirements. To resolve hash collisions, we utilize the external DRAM to extend the group table to save entries not accepted by hashing. When there is a hash collision in entry insertion, SuperFE stores the new entry into DRAM; when there is a hash collision in table lookup, SuperFE turns to DRAM to search the target entry. Though accessing DRAM is slow, there is no significant performance drop as long as the collision rate is low.

Hierarchical memory allocation. Netronome SmartNICs own a complex hierarchical memory, which is composed of CLS, CTM, IMEM, and EMEM, with increasing sizes but higher access latencies, as shown in Figure 8. These memories are shared by processing cores, e.g., CLS and CTM are only shared by cores on the same island while IMEM and EMEM are shared by all cores. To avoid memory contention where different cores access the same address of the same memory, we manipulate the ingress Network Block Interface (NBI) of NFP to distribute packets to cores on a per-IP basis.

Furthermore, SuperFE optimally maps group states across different memory hierarchies according to their sizes and access patterns to enhance memory access performance. We formulate the group table placement problem as follows. S denotes the set of states required by the given feature extracting policy, and SuperFE analyzes each state $s \in S$ to obtain its sizes b_s and access times t_s per packet. The hierarchical memory architecture can be represented by M, and each level of memory $m \in M$ has the access latency l_m and the maximum access data-bus width w_m. And the width of the group table in the memory m is n_m, which can be configured by users to balance the hash collision rate and lookup performance. We define the binary variable $p_{s,m} = 1$ if and only if the state s is placed into the group table in the memory m, otherwise $p_{s,m} = 0$. Then the group table placement problem can be solved based on Integer Linear Programming (ILP):

$$min \quad \sum_{s \in S} \sum_{m \in M} p_{s,m} t_s l_m \tag{3}$$

$$s.t. \quad \forall s \in S: \quad \sum_{m \in M} p_{s,m} = 1 \tag{4}$$

$$\forall m \in M: \; n_m \sum_{s \in S} p_{s,m} b_s \leq w_m \tag{5}$$

Equation (3) indicates that the objective is to minimize the total access latency to all states. Equation (4) specifies that each state must be placed into one memory hierarchy, and Equation (5) specifies that the data bus of each memory must hold access to the group table. Note that this ILP problem happens only when a new SuperFE policy is declared, so the time to solve the ILP problem is not a concern. Additionally, as the scale of ILP constraints is not large, the ILP problem can be quickly solved with existing optimization toolboxes like Gurobi [23]. Given the output $\{p_{s,m}\}$, we can determine the optimal group table placement strategy.

7 Implementation

We develop a SuperFE proof-of-concept prototype, including a policy enforcement engine, an MGPV batching engine (*FE-Switch*), and a feature computing engine (*FE-NIC*).

We implement the policy engine on the x86 platform with ~1K lines of Python code. It is a simple translator, which analyzes the input feature extraction policy described in §4, extracting operators groupby and filter to configure the program of *FE-Switch*, and assemble the program of *FE-NIC* by translating the rest of the operators.

The MGPV batching engine is implemented with ~2K lines of P4-16 for the Intel Tofino ASIC and ~4K lines of C for the control plane. We set the size and number of short buffers as 4 and 16384, and that of long buffers as 20 and 4096. We set the size of the FG table as the same size of short buffers, i.e., 16384. The aging threshold T is set according to traffic patterns.

The feature computing engine is implemented with ~3K lines of Micro-C for Netronome NFP-4000. It realizes basic frameworks for operators map, reduce, synthesize and collect, and provides a set of corresponding functions for Table 1 in Appendix A.

8 Evaluation

In this section, we evaluate SuperFE by answering the following key questions: (1) How expressive is the SuperFE policy language in supporting diverse feature extractors (§8.2)? (2) How well can our system work with state-of-the-art traffic analysis applications (§8.3)? (3) How effective is *FE-Switch* in batching packet feature metadata (§8.4)? (4) How efficient is *FE-NIC* in computing various feature vectors (§8.5)?

8.1 Experimental Setup

We deploy a real-world testbed and conduct trace-driven experiments to evaluate SuperFE. Our testbed consists of one 3.3 Tb/s Intel Tofino switch and two Dell R740 servers. Each server is equipped with Intel(R) Xeon(R) Gold 6230R CPUs and 64 GB memory. In particular, one server runs as the backend server, installed with two 40Gbps Netronome NFP-4000 SmartNICs. The other server runs as the traffic generator, connected to the switch via a 40 GbE Intel XL710 NIC. The prototype of SuperFE consists of the Tofino switch and the backend server. Limited by available facilities in our lab, we do not have access to GPUs currently, so we run the behavior detector on CPUs of the backend server.

Our workload traffic traces have different flow lengths and packet size distributions (Table 2), which are all collected

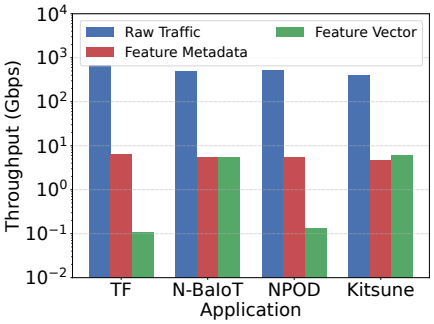

Figure 9. Overall system performance.

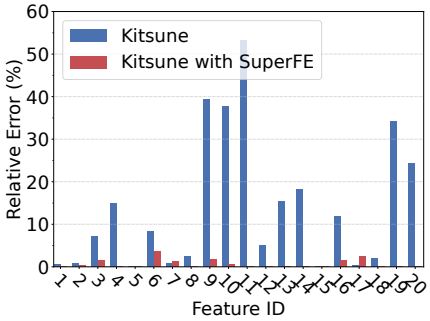

Figure 10. Error of feature extraction.

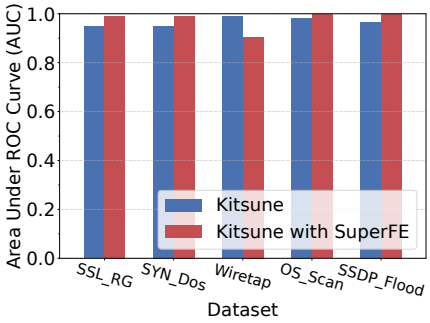

Figure 11. Accuracy of detection.

Table 2. Workload traffic traces.

Traffic Trace	Average Flow Length	Average Packet Size
MAWI-IXP	104 packets/flow	1246 Bytes/packet
ENTERPRISE	9.2 packets/flow	739 B/packet
CAMPUS	58 packets/flow	135 Bytes/packet

Table 3. Lines of code to implement different feature extractors with SuperFE.

Application	Objective of Traffic Analysis	Feature Dimension	LOC in SuperFE
CUMUL [49]	Website fingerprinting	104	29
AWF [55]	Website fingerprinting	5000	9
DF [60]	Website fingerprinting	5000	9
TF [61]	Website fingerprinting	5000	9
PeerShark [43]	Botnet detection	4	22
N-BaIoT [38]	Botnet detection	65	34
MPTD [7]	Covert channel detection	166	101
NPOD [67]	Covert channel detection	37	24
HELAD [80]	Intrusion detection	100	49
Kitsune [41]	Intrusion detection	115	49

from the real world to cover three different scenarios: (1) Trace *MAWI* is collected from a main Internet Exchange (IX) link, which is available on the online dataset [51]. (2) Trace *ENTERPRISE* is collected from a cloud gateway server of a cloud provider. (3) Trace *CAMPUS* is collected from the core router of our department, which records half-hour network activities of our colleagues. We also use four public application-specific traces to train and test our system, i.e., trace in [61] for website fingerprinting, trace in [38] for botnet detection, trace in [67] for covert channel detection and trace in [41] for intrusion detection. In our experiments, we replay these traces with MoonGen [16] to generate experimental traffic up to 40 Gbps. For experiments requiring a larger traffic volume, we employ techniques in [35, 82] to amplify the traffic by replicating and modifying packets with the programmable switch.

8.2 Policy Expressiveness

To demonstrate the expressiveness of the SuperFE policy interface, we re-implement feature extractors of 10 state-of-the-art traffic analysis applications using primitives provided by SuperFE. Table 3 lists the lines of code to implement them with SuperFE. As we can see, SuperFE allows concise specifications of the feature extraction policies and shields the complexities of the underlying hardware. Here we present a brief description of each example.

Website Fingerprinting. Early works rely on machine learning and per-flow statistical features of packet size and packet number, e.g., CUMUL [49]. Recent works attempt to apply deep learning and simplify input features to a fixed-length (5000) sequence of [-1, 1] representing packet directions of flows, e.g., AWF [55], DF [60], TF [61]. SuperFE supports both types of feature extractors (§4.2).

Botnet Detection. Researchers usually monitor the conversational pattern of each IP-pair to identify stealthy P2P communications. For example, PeerShark [43] and N-BaIoT [38] both compute per-IP-connection statistical features based on packet size and inter-packet time. In addition, N-BaIoT also utilizes similar per-host statistics to identify bot hosts. These basic statistical features are all available in SuperFE.

Covert Channel Detection. Existing works only focus on per-flow features to locate suspicious flows hiding covert channels. MPTD [7] depends on various statistical features, while NPOD [67] employs the distribution of packet size and inter-packet time of each flow. We discuss packet frequency distributions in detail in §4.2.

Intrusion Detection. State-of-the-art works, e.g., Kitsune [41] and HEALD [80], extract mixed statistical features from multiple granularities, i.e., host, channel, and socket. SuperFE proposes the group-based model to enable multi-granularities feature extraction and provides a set of reducing functions to compute these statistical features.

From the discussions above, we can see SuperFE policy interface is expressive and flexible to support generic traffic analysis applications. Such usability and flexibility are missed in recent hardware-accelerated feature extraction works. For instance, FlowLens [8] only supports the feature vector of packet distributions, which only satisfies NPOD [67].

8.3 Application Study

To illustrate that SuperFE is a powerful component to construct scalable traffic analysis applications, we seek representative works from four common types of applications

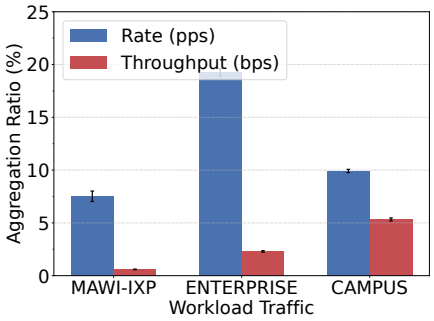

Figure 12. Aggregation ratio of MGPV.

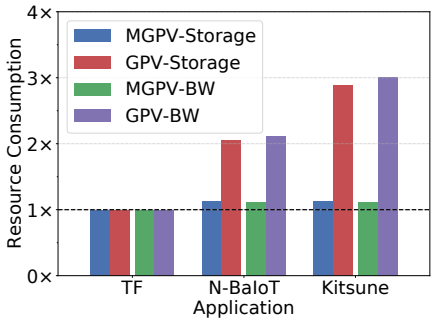

Figure 13. Resource efficiency of MGPV.

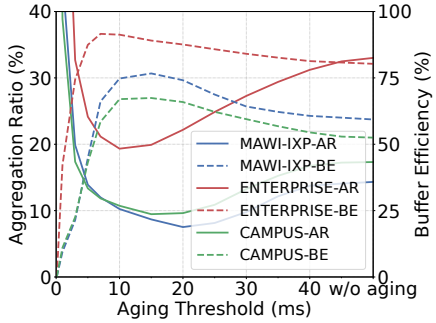

Figure 14. Optimization of aging design.

and reconstruct them with SuperFE to accelerate their feature extractors. Specially, we select TF [61], N-BaIoT [38], NPOD [67] and Kitsune [41] from Table 3 as state-of-the-art works. We rewrite their open-source codes to replace the software-based feature extractors with SuperFE, and reuse the original behavior detectors, i.e., triplet networks for TF, deep autoencoders for N-BaIoT, decision trees for NPOD, and autoencoders for Kitsune. We then conduct experiments to evaluate the scalability of SuperFE integrated with real-world traffic analysis applications.

Multi-100Gbps performance. To give a general insight into the performance of SuperFE, Figure 9 depicts the throughput of our system when it accelerates traffic analysis applications. This figure shows that SuperFE empowers these applications to handle multi-100Gbps raw traffic and generate feature vectors at rates of ~Gbps, which is nearly two orders of magnitude higher than their original implementations. The throughput and scalability gains can be attributed to the architecture of SuperFE, which offloads batching on the switch and computing on the SmartNIC separately. Although solutions [8, 15, 28, 59, 71, 73, 81] that leverage programmable switches or SmartNICs can achieve similar performance, they lack the flexibility to support generic and versatile traffic analysis applications. SuperFE takes both advantages of programmable switches and SmartNICs to realize scalable and flexible feature extractors.

Detection accuracy. The feature vectors produced by SuperFE must promise fidelity to the user-defined feature set. To illustrate this, we take Kitsune, which has the most complex feature computation, as the representative. We first calculate the relative error of feature vectors produced by SuperFE and the original Kitsune implementation, comparing them with the standard feature definitions. Results in Figure 10 indicate that the extraction error of SuperFE is below 4%, much better than that of the original Kitsune applying approximate algorithms. We further examine the detection accuracy of Kitsune models trained with these feature vectors, and Figure 11 shows Kitsune can achieve accurate detection under different scenarios with feature vectors generated by SuperFE.

Resource overhead. To evaluate the resource overhead of SuperFE, we profile its resource usage on our test switch and SmartNICs, when SuperFE is deployed with different applications. Table 4 displays several key hardware resources in our system. Utilization for most resources on the switch is low, except for stateful ALUs, which are heavily used by *FE-Switch* to implement the aggregation mechanism. But it still leaves enough resources for the concurrent execution of common forwarding behaviors, which do not require much stateful processing [63]. The utilization of hierarchical memory on SmartNICs is not a concern, as the SmartNIC is dedicated to feature computation intrinsically.

Table 4. Hardware resource utilization.

	Switch			SmartNIC
	Tables	**sALUs**	**SRAM**	**Memory**
TF	26.04%	68.75%	16.56%	49.17%
N-BaIoT	30.73%	72.92%	18.23%	57.30%
NPOD	26.04%	68.75%	16.56%	74.46%
Kitsune	31.77%	77.08%	18.75%	60.81%

8.4 Batching on Switch

Effective batching. To illustrate that our MGPV batching mechanism of *FE-Switch* significantly reduces the workload of SmartNICs, we measure the ratio between the MGPV traffic sent from the switch to SmartNICs and the original traffic received by the switch, which is referred to as the aggregation ratio. We run TF, N-BaIoT, NPOD, and Kitsune with *FE-Switch* and replay three traffic traces to conduct the same measurement, and the results are shown in Figure 12. We can see that our design on the switch enables an over 80% reduction both in receiving rate and receiving throughput for SmartNICs, which effectively protects SmartNICs from processing massive original traffic directly.

Powerful MGPV. To demonstrate the advantage of MGPV in supporting multi-granularity packet feature metadata, we compare the memory occupation and switch-SmartNIC bandwidth consumption of MGPV and GPV when they support applications having different grouping requirements. In particular, we select three traffic analysis applications, TF, N-BaIoT, and Kitsune, which group packets by one, two, and three granularities, respectively, and take the *FE-Switch* resource utilization of k-fingerprinting as the baseline. As shown in Figure 13,

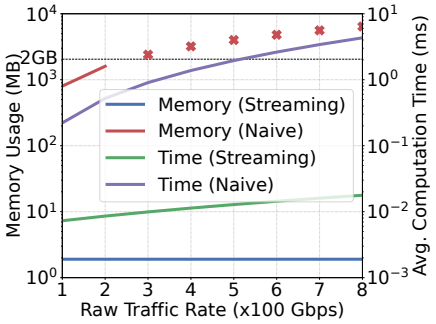

Figure 15. Efficient streaming algorithms.

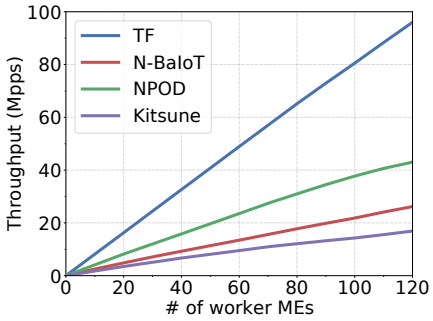

Figure 16. Scalability of SmartNICs.

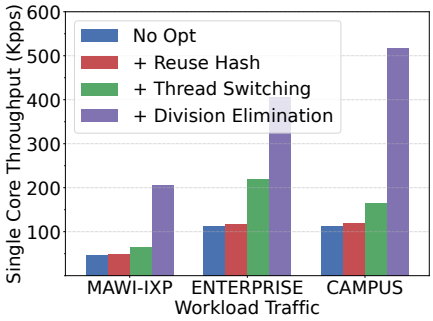

Figure 17. Optimizations on SmartNICs.

MGPV can maintain an approximately constant resource footprint while the resources desired by GPV increase linearly with the grouping granularities utilized by applications. This is because the dependency chain of granularities helps MGPV store only one copy of feature metadata for each packet.

Aging mechanism. The aging mechanism further optimizes the resource utilization of MGPV, whose efficiency is decided by the parameter T, i.e., the timeout period. To find a suitable value for T, we deploy TF on *FE-Switch* and configure different values for T to seek how T affects the aggregation ratio and buffer efficiency, i.e., the ratio of active flows in MGPV buffers. As shown in Figure 14, the aging mechanism can reduce the aggregation ratio and increase the buffer efficiency, which manifests *FE-Switch* can handle more active flows. Furthermore, too small T would cause entries of MGPV to be frequently evicted by timeout, and too large T would cause the aging mechanism to fail. Figure 14 also indicates the appropriate value of T depends on the flow distribution of the traffic. For instance, the average flow length of ENTERPRISE is short so a small T is applicable. Besides, the aging mechanism also imposes an upper limit on the batching delay, which does not exceed O(10) milliseconds.

8.5 Computing on SmartNIC

Efficient computing. To illustrate that *FE-NIC* efficiently computes features by performing streaming algorithms, we measure the total memory consumption and the average feature computation time of the feature extractor with and without streaming algorithms. Since Kitsune applies some representative features, we experiment by re-implementing its feature extractor with naïve algorithms and accelerating it using streaming algorithms. Figure 15 presents *FE-NIC* sustains rapid feature computing with a small memory footprint on SmartNICs even when dealing with heavy traffic, which benefits from the adoption of streaming algorithms. On the contrary, naïve algorithms ask for a large amount of on-chip memory, which exceeds the capacity of our SmartNICs.

Scalable performance. To determine whether *FE-NIC* takes full advantage of the multi-core scalability of SoC-based architecture, we profile the performance gain of *FE-NIC* by allocating more SoC cores to the system, from a single core to 120 cores of two SmartNICs. Figure 16 shows the scalability of our design when SmartNICs execute different feature computations, where WFP owns the simplest feature extractor so it achieves the highest throughput. From these results, we can see *FE-NIC* is able to scale almost linearly as an increasing number of SoC cores are involved, because *FE-NIC* nearly eliminates all the resource contention between cores. We can also add more SmartNICs to scale up *FE-NIC* further, with a simple load-balance mechanism implemented on the switch to distribute the MGPV traffic across them evenly.

Advanced optimizations. To testify the effect of optimizations that *FE-NIC* adopts for performance improvement, we estimate the throughput of SoC cores when these optimizations are enabled incrementally and the results are displayed in Figure 17. When all optimizations are enabled, the total throughput of *FE-NIC* rises to as much as 4 times when compared to the baseline setup without any optimizations applied. The elimination of division shows the most significant performance improvement, as the division takes up to hundreds of times more cycles than other arithmetic operations.

9 Discussion

More complex granularity dependency relationships. Future traffic analysis applications may require features of more granularity to locate malicious behaviors more accurately. This means the used granularity may be abstracted into a dependency graph instead of a dependency chain. A possible solution is to split the dependency graph into a minimum number of dependency chains and allocate resources for each granularity chain to apply MGPV separately. Such a dependency graph cutting algorithm is left for our future work.

Traffic analysis v.s. network monitor. Although seeming similar, traffic analysis applications have quite different semantics from network monitor applications [22, 44, 50, 62, 63]. Traffic analysis pays more attention to host behaviors, e.g., whether a host is compromised, or accesses a malicious website. By contrast, network monitoring emphasizes more on packet/network behaviors, e.g., whether there is congestion, packet loss, or a deep switch queue. Because of such differences, traffic analysis applications are transitioning to

use machine learning, especially deep learning, to achieve better detection accuracy, while traffic monitor applications are not so interested in emerging deep learning algorithms [65].

10 Related Work

Besides the most relevant works discussed in the main text, our work is also inspired by the following topics.

Policy languages. There are many domain-specific policy languages in the networking community [3, 4, 19, 22] and the security community [9, 72, 75, 78] to simplify policy expression. Although our key idea of using programmable switches and SmartNICs to accelerate the feature extractor is not tied to any specific policy language, to hide the underlying hardware complexity, we extend SuperFE policy interface based on Spark-style stream processing operators [76], which is tailored for feature extraction in ML-based traffic analysis applications.

Programmable switches. SuperFE builds on the recent trends that leverage programmable switches to accelerate various applications in networking [22, 40, 44], distributed systems [30, 31] and security [36, 39, 78], but focuses on a different problem: accelerating feature extractor in ML-based traffic analysis applications. To achieve this, we design the multi-granularity Group Packet Vector (MGPV) abstraction to batch packet feature metadata in programmable switches.

SmartNICs. Recent research has increasingly demonstrated the benefits of SmartNICs in offloading and accelerating network functions [17, 34, 52], key-value storage [33, 37], transport protocol [5, 42], and packet inspection [26, 50]. SuperFE is along with these lines and explores the utility of SmartNICs in enhancing feature extractors for ML-based traffic analysis applications. Specially, we propose a design that coordinates the multi-core parallel and hierarchical memory of SoC-based SmartNICs to achieve efficient streaming algorithms.

11 Conclusion

In this paper, we identify the current feature extractor is becoming the key bottleneck of ML-based traffic analysis applications, and introduce SuperFE, a scalable and flexible feature extraction framework leveraging the capabilities of programmable switches and SmartNICs. SuperFE presents a user-friendly and extensible interface to support custom feature extraction policies without considering underlying hardware complexities, introduces a high-performance multi-granularity key-vector cache system in the programmable switches to batch necessary feature metadata, and exploits the multi-core parallel and hierarchical memory of SoC-based SmartNICs to achieve efficient feature computation with streaming algorithms. Our implementation and evaluation demonstrate that SuperFE allows various state-of-the-art traffic analysis applications to efficiently extract features from multi-100Gbps raw traffic without compromise in detection accuracy, and provides nearly two orders of magnitude higher throughput than their software-based counterparts. We hope SuperFE can serve as the foundation of the next-generation feature extractor for ML-based traffic analysis applications.

Acknowledgment

We thank our shepherd, Xiaosong Ma, and the anonymous EuroSys reviewers for their valuable comments. This work is supported in part by the National Natural Science Foundation of China (No. 62402025, No. 62402024, No. 62221003), the Fundamental Research Funds for the Central Universities and gifts from Beihang-Huawei Key Software Joint Laboratory. Renyu Yang and Shicheng Wang are the corresponding authors.

References

[1] Kota Abe and Shigeki Goto. 2016. Fingerprinting attack on Tor anonymity using deep learning. *Proceedings of the Asia-Pacific Advanced Network* 42 (2016), 15–20.

[2] Accton. 2024. Towards 800G and 1600G Ethernet. https://www.accton.com/Technology-Brief/towards-800g-and-1600g-ethernet/.

[3] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. 2014. NetKAT: Semantic foundations for networks. *Acm sigplan notices* 49, 1 (2014), 113–126.

[4] Mina Tahmasbi Arashloo, Yaron Koral, Michael Greenberg, Jennifer Rexford, and David Walker. 2016. SNAP: Stateful network-wide abstractions for packet processing. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 29–43.

[5] Mina Tahmasbi Arashloo, Alexey Lavrov, Manya Ghobadi, Jennifer Rexford, David Walker, and David Wentzlaff. 2020. Enabling Programmable Transport Protocols in {High-Speed}{NICs}. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 93–109.

[6] Asterfuison. 2022. The Most Comprehensive DPU/SmartNIC Vendors with its Product Line Summary. https://cloudswit.ch/blogs/the-most-complete-dpu-smartnic-vendors-with-its-product-line-summary/.

[7] Diogo Barradas, Nuno Santos, and Luís Rodrigues. 2018. Effective detection of multimedia protocol tunneling using machine learning. In *27th USENIX Security Symposium (USENIX Security 18)*. 169–185.

[8] Diogo Barradas, Nuno Santos, Luís Rodrigues, Salvatore Signorello, Fernando M. V. Ramos, and André Madeira. 2021. FlowLens: Enabling Efficient Flow Classification for ML-based Network Security Applications. In *NDSS*.

[9] Kevin Borders, Jonathan Springer, and Matthew Burnside. 2012. Chimera: A declarative language for streaming network traffic analysis. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*. 365–379.

[10] Serdar Cabuk, Carla E Brodley, and Clay Shields. 2004. IP covert timing channels: design and detection. In *Proceedings of the 11th ACM conference on Computer and communications security*. 178–187.

[11] CAIDA. 2022. Passive Monitor: equinix-chicago. https://www.caida.org/catalog/datasets/monitors/passive-equinix-chicago/.

[12] Jack Choquette, Olivier Giroux, and Denis Foley. 2018. Volta: performance and programmability. *IEEE Micro* 38, 2 (2018), 42–52.

[13] Cisco. 2023. New Cisco 800G Innovations Help to Supercharge the Internet for the Future by Improving Networking Economics and Sustainability for Service Providers and Cloud Providers. https://newsroom.cisco.com/c/r/newsroom/en/us/a/y2023/m03/new-cisco-800g-innovations-help-to-supercharge-the-internet-for-the-future-by-improving-networking-economics-and-

sustainability-for-service-providers-and-cloud-providers.html.

[14] Ralph B D'Agostino and Michael A Stephens. 1986. Goodness-of-fit techniques.

[15] Yutao Dong, Qing Li, Kaidong Wu, Ruoyu Li, Dan Zhao, Gareth Tyson, Junkun Peng, Yong Jiang, Shutao Xia, and Mingwei Xu. 2023. {HorusEye}: A Realtime {IoT} Malicious Traffic Detection Framework using Programmable Switches. In *32nd USENIX Security Symposium (USENIX Security 23)*. 571–588.

[16] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2015. Moongen: A scriptable high-speed packet generator. In *Proceedings of the 2015 Internet Measurement Conference*. 275–287.

[17] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. 2018. Azure Accelerated Networking:{SmartNICs} in the Public Cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 51–66.

[18] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. 2007. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Discrete Mathematics and Theoretical Computer Science*. Discrete Mathematics and Theoretical Computer Science, 137–156.

[19] Nate Foster, Rob Harrison, Michael J Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker. 2011. Frenetic: A network programming language. *ACM Sigplan Notices* 46, 9 (2011), 279–291.

[20] Chuanpu Fu, Qi Li, and Ke Xu. 2023. Detecting unknown encrypted malicious traffic in real time via flow interaction graph analysis. In *NDSS*.

[21] Steven Gianvecchio and Haining Wang. 2007. Detecting covert timing channels: an entropy-based approach. In *Proceedings of the 14th ACM conference on Computer and communications security*. 307–316.

[22] Arpit Gupta, Rob Harrison, Marco Canini, Nick Feamster, Jennifer Rexford, and Walter Willinger. 2018. Sonata: query-driven streaming network telemetry. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. ACM, 357–371.

[23] Gurobi. 2024. Gurobi Optimization. https://www.gurobi.com/.

[24] Jamie Hayes and George Danezis. 2016. k-fingerprinting: A robust scalable website fingerprinting technique. In *25th USENIX Security Symposium (USENIX Security 16)*. 1187–1203.

[25] Jordan Holland, Paul Schmitt, Nick Feamster, and Prateek Mittal. 2021. New directions in automated traffic analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 3366–3383.

[26] Joel Hypolite, John Sonchack, Shlomo Hershkop, Nathan Dautenhahn, André DeHon, and Jonathan M Smith. 2020. DeepMatch: practical deep packet inspection in the data plane using network processors. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*. 336–350.

[27] Intel. 2022. Intel Tofino: P4-programmable Ethernet switch ASIC that delivers better performance at lower power. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series.html.

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

[29] Steve TK Jan, Qingying Hao, Tianrui Hu, Jiameng Pu, Sonal Oswal, Gang Wang, and Bimal Viswanath. 2020. Throwing darts in the dark? detecting bots with limited data using neural data augmentation. In *2020 IEEE symposium on security and privacy (SP)*. IEEE, 1190–1206.

[30] Xin Jin, Xiaozhou Li, Haoyu Zhang, Nate Foster, Jeongkeun Lee, Robert Soulé, Changhoon Kim, and Ion Stoica. 2018. NetChain: Scale-Free Sub-RTT Coordination. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association.

[31] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. 2017. NetCache: Balancing Key-Value Stores with Fast In-Network Caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 121–136.

[32] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. 1–12.

[33] Bojie Li, Zhenyuan Ruan, Wencong Xiao, Yuanwei Lu, Yongqiang Xiong, Andrew Putnam, Enhong Chen, and Lintao Zhang. 2017. Kv-direct: High-performance in-memory key-value store with programmable nic. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 137–152.

[34] Bojie Li, Kun Tan, Layong Luo, Yanqing Peng, Renqian Luo, Ningyi Xu, Yongqiang Xiong, Peng Cheng, and Enhong Chen. 2016. Clicknp: Highly flexible and high performance network processing with reconfigurable hardware. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 1–14.

[35] Guanyu Li, Menghao Zhang, Cheng Guo, Han Bao, Mingwei Xu, Hongxin Hu, and Fenghua Li. 2022. {IMap}: Fast and Scalable {In-Network} Scanning with Programmable Switches. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 667–681.

[36] Guanyu Li, Menghao Zhang, Chang Liu, Xiao Kong, Ang Chen, Guofei Gu, and Haixin Duan. 2019. Nethcf: Enabling line-rate and adaptive spoofed ip traffic filtering. In *2019 IEEE 27th international conference on network protocols (ICNP)*. IEEE, 1–12.

[37] Ming Liu, Liang Luo, Jacob Nelson, Luis Ceze, Arvind Krishnamurthy, and Kishore Atreya. 2017. Incbricks: Toward in-network computation with an in-network cache. *ACM SIGOPS Operating Systems Review* 51, 2 (2017), 795–809.

[38] Yair Meidan, Michael Bohadana, Yael Mathov, Yisroel Mirsky, Asaf Shabtai, Dominik Breitenbacher, and Yuval Elovici. 2018. N-baiot—network-based detection of iot botnet attacks using deep autoencoders. *IEEE Pervasive Computing* 17, 3 (2018), 12–22.

[39] Roland Meier, Petar Tsankov, Vincent Lenders, Laurent Vanbever, and Martin Vechev. 2018. NetHide: Secure and practical network topology obfuscation. In *27th USENIX Security Symposium (USENIX Security 18)*. 693–709.

[40] Rui Miao, Hongyi Zeng, Changhoon Kim, Jeongkeun Lee, and Minlan Yu. 2017. SilkRoad: Making Stateful Layer-4 Load Balancing Fast and Cheap Using Switching ASICs. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 15–28.

[41] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: an ensemble of autoencoders for online network intrusion detection. In *Proceedings of the 25th Network and Distributed System Security Symposium (NDSS)*.

[42] YoungGyoun Moon, SeungEon Lee, Muhammad Asim Jamshed, and KyoungSoo Park. 2020. {AccelTCP}: Accelerating Network Applications with Stateful {TCP} Offloading. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. 77–92.

[43] Pratik Narang, Subhajit Ray, Chittaranjan Hota, and Venkat Venkatakrishnan. 2014. Peershark: detecting peer-to-peer botnets by tracking conversations. In *2014 IEEE Security and Privacy Workshops*. IEEE, 108–115.

[44] Srinivas Narayana, Anirudh Sivaraman, Vikram Nathan, Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimalkumar Jeyakumar, and Changhoon Kim. 2017. Language-directed hardware design for network performance monitoring. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 85–98.

[45] Milad Nasr, Alireza Bahramali, and Amir Houmansadr. 2018. Deepcorr: Strong flow correlation attacks on tor using deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1962–1976.

[46] Milad Nasr, Amir Houmansadr, and Arya Mazumdar. 2017. Compressive traffic analysis: A new paradigm for scalable traffic analysis. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2053–2069.

[47] Netronome. 2022. Netronome MicroC. https://cdn.open-nfp.org/media/documents/the-joy-of-micro-c_fcjSfra.pdf.

[48] NVIDA. 2024. NVIDIA BlueField Data Processing Units. https://www.nvidia.com/en-us/networking/products/data-processing-unit/.

[49] Andriy Panchenko, Fabian Lanze, Jan Pennekamp, Thomas Engel, Andreas Zinnen, Martin Henze, and Klaus Wehrle. 2016. Website Fingerprinting at Internet Scale.. In *NDSS*.

[50] Sourav Panda, Yixiao Feng, Sameer G Kulkarni, KK Ramakrishnan, Nick Duffield, and Laxmi N Bhuyan. 2021. SmartWatch: accurate traffic analysis and flow-state tracking for intrusion prevention using SmartNICs. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies*. 60–75.

[51] WIDE Project. [n. d.]. MAWI Working Group Traffic Archive. https://mawi.wide.ad.jp/mawi/.

[52] Yiming Qiu, Jiarong Xing, Kuo-Feng Hsu, Qiao Kang, Ming Liu, Srinivas Narayana, and Ang Chen. 2021. Automated smartnic offloading insights for network functions. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 772–787.

[53] Daniel Ramsbrock, Xinyuan Wang, and Xuxian Jiang. 2008. A first step towards live botmaster traceback. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 59–77.

[54] ISP review. 2023. Neos Networks Bring National 400Gbps Services to UK Businesses. https://www.ispreview.co.uk/index.php/2023/08/neos-networks-bring-national-400gbps-services-to-uk-businesses.html.

[55] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. 2018. Automated website fingerprinting through deep learning. In *NDSS*.

[56] SDXcentral. 2022. AT&T Picks Barefoot Networks for Programmable Switches. https://www.sdxcentral.com/articles/news/att-picks-barefoot-networks-programmable-switches/2017/04/.

[57] SDXcentral. 2022. Barefoot Scores Tofino Deals with Alibaba, Baidu, and Tencent. https://www.sdxcentral.com/articles/news/barefoot-scores-tofino-deals-with-alibaba-baidu-and-tencent/2017/05/.

[58] Rajath Shashidhara, Tim Stamler, Antoine Kaufmann, and Simon Peter. 2022. {FlexTOE}: Flexible {TCP} Offload with {Fine-Grained} Parallelism. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 87–102.

[59] Giuseppe Siracusano, Salvator Galea, Davide Sanvito, Mohammad Malekzadeh, Gianni Antichi, Paolo Costa, Hamed Haddadi, and Roberto Bifulco. 2022. Re-architecting Traffic Analysis with Neural Network Interface Cards. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 513–533.

[60] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. 2018. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 1928–1943.

[61] Payap Sirinam, Nate Mathews, Mohammad Saidur Rahman, and Matthew Wright. 2019. Triplet fingerprinting: More practical and portable website fingerprinting with n-shot learning. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1131–1148.

[62] John Sonchack, Adam J Aviv, Eric Keller, and Jonathan M Smith. 2018. Turboflow: information rich flow record generation on commodity switches. In *Proceedings of the Thirteenth EuroSys Conference*. ACM, 11.

[63] John Sonchack, Oliver Michel, Adam J Aviv, Eric Keller, and Jonathan M Smith. 2018. Scaling Hardware Accelerated Network Monitoring to Concurrent and Dynamic Queries With* Flow. In *2018 USENIX Annual Technical Conference USENIX ATC 18)*. USENIX Association.

[64] Stuart Staniford-Chen and L Todd Heberlein. 1995. Holding intruders accountable on the internet. In *Proceedings 1995 IEEE Symposium on Security and Privacy*. IEEE, 39–49.

[65] Lizhuang Tan, Wei Su, Wei Zhang, Jianhui Lv, Zhenyi Zhang, Jingying Miao, Xiaoxi Liu, and Na Li. 2021. In-band network telemetry: A survey. *Computer Networks* 186 (2021), 107763.

[66] Gerry Wan, Fengchen Gong, Tom Barbette, and Zakir Durumeric. 2022. Retina: analyzing 100GbE traffic on commodity hardware. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 530–544.

[67] Liang Wang, Kevin P Dyer, Aditya Akella, Thomas Ristenpart, and Thomas Shrimpton. 2015. Seeing through network-protocol obfuscation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 57–69.

[68] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective attacks and provable defenses for website fingerprinting. In *23rd USENIX Security Symposium (USENIX Security 14)*. 143–157.

[69] BP Welford. 1962. Note on a method for calculating corrected sums of squares and products. *Technometrics* 4, 3 (1962), 419–420.

[70] George Winslow. 2022. Comcast Reports 400 Gbps Internet Speeds in Hollowcore Fiber Test. https://www.tvtechnology.com/news/comcast-reports-400-gbps-internet-speeds-in-hollowcore-fiber-test.

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

[72] Jiarong Xing, Wenqing Wu, and Ang Chen. 2021. Ripple: A Programmable, Decentralized Link-Flooding Defense Against Adaptive Adversaries. In *30th USENIX Security Symposium (USENIX Security 21)*. 3865–3881.

[73] Jinzhu Yan, Haotian Xu, Zhuotao Liu, Qi Li, Ke Xu, Mingwei Xu, and Jianping Wu. 2024. {Brain-on-Switch}: Towards Advanced Intelligent Network Data Plane via {NN-Driven} Traffic Analysis at {Line-Speed}. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 419–440.

[74] Kunikazu Yoda and Hiroaki Etoh. 2000. Finding a connection chain for tracing intruders. In *European Symposium on Research in Computer Security*. Springer, 191–205.

[75] Tianlong Yu, Seyed Kaveh Fayaz, Michael P Collins, Vyas Sekar, and Srinivasan Seshan. 2017. PSI: Precise Security Instrumentation for Enterprise Networks.. In *NDSS*.

[76] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. 15–28.

[77] Chaoliang Zeng, Layong Luo, Teng Zhang, Zilong Wang, Luyang Li, Wenchen Han, Nan Chen, Lebing Wan, Lichao Liu, Zhipeng Ding, et al. 2022. Tiara: A scalable and efficient hardware acceleration architecture for stateful layer-4 load balancing. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 1345–1358.

[78] Menghao Zhang, Guanyu Li, Shicheng Wang, Chang Liu, Ang Chen, Hongxin Hu, Guofei Gu, Qianqian Li, Mingwei Xu, and Jianping Wu. 2020. Poseidon: Mitigating volumetric ddos attacks with programmable switches. In *the 27th Network and Distributed System Security Symposium (NDSS 2020)*.

[79] Yin Zhang and Vern Paxson. 2000. Detecting stepping stones.. In *USENIX Security Symposium*, Vol. 171. 184.

[80] Ying Zhong, Wenqi Chen, Zhiliang Wang, Yifan Chen, Kai Wang, Yahui Li, Xia Yin, Xingang Shi, Jiahai Yang, and Keqin Li. 2020. HELAD: A novel network anomaly detection model based on heterogeneous ensemble learning. *Computer Networks* 169 (2020), 107049.

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

[82] Yu Zhou, Zhaowei Xi, Dai Zhang, Yangyang Wang, Jinqiu Wang, Mingwei Xu, and Jianping Wu. 2019. Hypertester: high-performance network testing driven by programmable switches. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*. 30–43.

Appendix

A Parameters supported by SuperFE

The parameters supported by SuperFE policy interfaces are listed in Table 5. The *Granularity* is used as the argument of the *groupby* operator and is customized beyond the above ones. The *Mapping Function*, the *Reducing Function*, and the *Synthesizing Function* are used in the operator *map*, *reduce* and *synthesize* respectively, which specify the concrete operations.

Table 5. Description of parameters in SuperFE interface.

Parameter	Description
Granularity (g)	
flow	Group packets by the 5-tuple.
host	Group packets by the source IP and record direction information of each packet.
channel	Group packets by the IP-pair and record direction information of each packet.
socket	Group packets by the 5-tuple and record direction information of each packet.
Mapping Function (mf)	
f_one	Add a key-value pair whose value is "1".
f_ipt	Add a key-value pair to record the inter-packet time.
f_speed	Add a key-value pair to compute the speed.
f_burst	Add structured key-value pairs to identify bursts.
f_direction	Add a key-value pair to record the direction of the packet.
Reducing Function (rf)	
f_sum	Calculate the sum.
f_mean	Calculate the mean.
f_var	Calculate the variance.
f_std	Calculate the stddev.
f_max	Calculate the max.
f_min	Calculate the min.
f_kur	Calculate the kurtosis.
f_skew	Calculate the skew.
f_mag	Calculate the magnitude of bidirectional sequences.
f_radius	Calculate the radius of bidirectional sequences.
f_cov	Calculate the covariance between bidirectional sequences.
f_pcc	Calculate the correlation coefficient of bidirectional sequences.
f_card	Calculate the cardinality.
f_array	Pack fields as an array.
f_pdf	Estimate the probability density function.
f_cdf	Estimate the cumulative distribution function.
ft_hist{}	Obtain the histogram.
ft_percent{}	Estimate the quantile.
Synthesizing Function (sf)	
f_marker	Add a structure at each direction change to reflect the bytes/packet numbers previously sent.
f_norm	Normalize the sequence.
ft_sample	Sample from a sequence.