SF-TAP: Scalable and Flexible Traffic Analysis Platform Running on Commodity Hardware

Yuuki Takano, Ryosuke Miura, Shingo Yasuda
Kunio Akashi, Tomoya Inoue

NICT, JAIST
(Japan)

in USENIX LISA 2015
Table of Contents

1. Motivation
2. Related Work
3. Design of SF-TAP
4. Implementation of SF-TAP
5. Performance Evaluation
6. Conclusion
Motivation (1)

• Programmable application level traffic analyzer

• We want …

  • to write traffic analyzers in any languages such as Python, Ruby, C++, for many purposes (IDS/IPS, forensic, machine learning).

• **not** to write codes handling TCP stream reconstruction (quite complex).

• modularity for many application protocols.
Motivation (2)

• High speed application level traffic analyzer

• We want …
  • to handle high bandwidth traffic.
  • to handle high connections per second.
  • horizontal and CPU core scalable analyzer.
Motivation (3)

• Running on Commodity Hardware
  • We want ...
    • open source software.
  • not to use expensive appliances.
Related Work

SF-TAP

+ modularity and scalability

GASPP [USENIX ATC 2014]
SCAP [IMC 2012]
libnids
(flow oriented analyzer)

I7-filter
nDPI
libprotoident
(application traffic detector)
(low level traffic capture)

DPDK
netmap [USENIX ATC 2012]
pcap
BPF [USENIX ATC 1993]
High-level Architecture of SF-TAP

Core Scaling
SF-TAP Cell
Analyzer Analyzer Analyzer Analyzer
CPU CPU CPU CPU
Flow Abstractor

Horizontal Scaling

Intra Network

Cell Incubator

The Internet

10GbE

10GbE
Design Principle (1)

- Flow Abstraction
  - abstract flows by application level protocols
  - provide flow abstraction interfaces like /dev, /proc or BPF
  - for multiple programming languages
- Modular Architecture
  - separate analyzing and capturing logic
  - easily replace analyzing logic
Design Principle (2)

- Horizontal Scalable
  - analyzing logic tends to require many computer resources
  - volume effect should solve the problem
- CPU Core Scalable
  - both analyzing and capturing logic should be core scalable for efficiency
Design of SF-TAP (1)

defined 4 planes

Analyzer Plane
application level analyzers
Forensic, IDS/IPS, etc...  (users of SF-TAP implements here)

Abstractor Plane
flow abstraction  (we implemented)

Separator Plane
flow separation  (we implemented)

Capturer Plane
traffic capturing  (ordinary tech.)
Design of SF-TAP (2) SF-TAP Cell Incubator

Flow Separator
separate flows to multiple Ifs

IP Fragment Handler
handle fragmented packets

Packet Forwarder
layer 2 bridge
layer 2 frame capture

SF-TAP Cell Incubator
Flow Separator
IP Fragment Handler
Packet Forwarder
L2 Bridge

separated traffic
other SF-TAP cells
Design of SF-TAP (3)
SF-TAP Flow Abstractor

TCP and UDP Handler
reconstruct TCP flows
nothing to do for UDP

Flow Identifier
identify flows by IP and port

IP Packet Defragmenter
defragment IP packets if needed
Design of SF-TAP (4)
SF-TAP Flow Abstractor

Flow Classifier
classify flows by regular expressions
output to abstraction IFs
Implementation

- SF-TAP cell incubator
  - C++11
  - it uses netmap, available on FreeBSD
- SF-TAP flow abstractor
  - C++11
  - it uses pcap or netmap (updated from the paper)
  - available on Linux, *BSD, and MacOS
- Source Code
  - https://github.com/SF-TAP
- License
  - 3-clauses BSD
Performance Evaluation (1)

Figure 7: CPU Load of HTTP Analyzer and Flow Abstractor

Figure 8: Total Memory Usage of HTTP Analyzer

Figure 9: Packet Drop against CPS

5.3 Performance Evaluation of the Cell Incubator

In the experiments involving the cell incubator, we used a PC with DDR3 16 GB Memory and an Intel Xeon E5-2470 v2 processor (10 cores, 2.4 GHz, 25 MB cache) and FreeBSD 10.1. The computer was equipped with four Intel quad-port 1 GbE NICs and an Intel dual-port 10 GbE NIC. We generated network traffic consisting of short packets (i.e., 64-byte L2 frames) on the 10 GbE lines for our evaluations. The cell incubator separated traffic based on the flows, with the separated flows forwarded to the twelve 1 GbE lines. Figure 13 shows our experimental network.

We conducted our experiments using three patterns:

1. The cell incubator worked in the mirroring mode using port mirroring on the L2 switch; in other words, it captured packets at $\alpha$ and forwarded packets to $\gamma$;
2. The cell incubator worked in the inline mode but did not forward packets to 1 GbE NICs, instead only $\alpha$ to $\beta$; and
3. The cell incubator worked in the inline mode, capturing packets at $\alpha$ and forwarding to both $\beta$ and $\gamma$.

Table 14 shows the performance of the cell incubator. For pattern (1), i.e., the mirroring mode, the cell incubator could manage packets up to 12.49 Mpps. For pattern (2), i.e., the cell incubator working as an L2 bridge, it could forward packets up to 11.60 Mpps. For pattern (3), i.e., forwarding packets to $\beta$ and $\gamma$, the cell incubator could forward packets to $\beta$ and $\gamma$ up to 11.44 Mpps. The performance of the inline mode was poorer than that of the mirroring mode because packets were forwarded to two NICs when using the inline mode. However, the inline mode is more suitable for specific purposes such as IDS/IPS because the same packets are dropped at $\beta$ and $\gamma$. In other words, all transmitted packets can be captured when using the inline mode.

Table 15 shows the CPU load averages of the cell incubator when in the inline mode and forwarding 64-byte frames. At 5.95 and 10.42 Mpps, packets were not dropped when forwarding. At approximately 10.42 Mpps, the upper limit of dropless forwarding was reached. This indicates that several CPUs were used for forwarding, but the 15th CPU's resources were especially consumed.

( pcap) packet drop against connections per second
Ling include using cryptographic protocols such as SS-TLS instead of traditional protocols such as HTTP and SSL. Countermeasures against pervasive monitoring [5, 37, 41]. The implementation of regular expressions for high-performance regular expressions should be expensive, it potentially requires high computational resources. Thus, high-performance regular expressions should be abstracted, but at present, this is not critical. Nonetheless, it is simply an implementation issue and is benchmarked for the cell incubator. The current implementation of the cell incubator requires the memory allocation strategy of netmap. The replacement of malloc by a slab allocator therefore constitutes an aspect of GPGPU for high-performance regular expressions. We plan to improve the performance of the flow abstractor. Although the replacement of malloc by a slab allocator is trivial, it is not suitable for mobile devices, which are widely used in today’s society, because of the lack of machine resources. Therefore, new approaches such as IDS and intrusion detection systems (IDS) are needed. Here, malloc can be replaced by another lightweight mechanism such as a slab allocator. The replacement of malloc by a slab allocator constitutes an aspect of GPGPU for high-performance regular expressions.

5.95 Mpps, the load average was approximately 50%, but over, at 14.88 Mpps, CPU resources were completely consumed. This limitation in forwarding performance was probably caused by the bias, which in turn was due to software interrupts occurring among the NIC, kernel, and user space. Thus, new mechanisms for high-performance message passing and packet-capturing applications, and libnids [15] is used to support the future of the Internet. The L7 loopback interface of the flow director [10] of Intel’s NIC and its driver. The flow director cannot currently be controlled by user programs. Support for packet-capturing applications is also a challenge, because of the lack of machine resources. Therefore, new approaches such as IDS and intrusion detection systems (IDS) are needed. Here, malloc can be replaced by another lightweight mechanism such as a slab allocator. The replacement of malloc by a slab allocator constitutes an aspect of GPGPU for high-performance regular expressions.

Thus, new mechanisms for high-performance message passing and packet-capturing applications, and libnids [15] is used to support the future of the Internet. The L7 loopback interface of the flow director [10] of Intel’s NIC and its driver. The flow director cannot currently be controlled by user programs. Support for packet-capturing applications is also a challenge, because of the lack of machine resources. Therefore, new approaches such as IDS and intrusion detection systems (IDS) are needed. Here, malloc can be replaced by another lightweight mechanism such as a slab allocator. The replacement of malloc by a slab allocator constitutes an aspect of GPGPU for high-performance regular expressions.

In addition, SCAP adopts a mechanism called subzero-copy packet transfer using analyzers that analyze packet at layer 7. SCAP is implemented within a Linux kernel, taking advantage of the zero-copy mechanism and allocating less threads for NIC’s RX and TX queues to achieve high throughput. In addition, SCAP adopts a mechanism called subzero-copy packet transfer using analyzers that analyze packet at layer 7. SCAP is implemented within a Linux kernel, taking advantage of the zero-copy mechanism and allocating less threads for NIC’s RX and TX queues to achieve high throughput. In addition, SCAP adopts a mechanism called subzero-copy packet transfer using analyzers that analyze packet at layer 7. SCAP is implemented within a Linux kernel, taking advantage of the zero-copy mechanism and allocating less threads for NIC’s RX and TX queues to achieve high throughput.

Thus, new mechanisms for high-performance message passing and packet-capturing applications, and libnids [15] is used to support the future of the Internet. The L7 loopback interface of the flow director [10] of Intel’s NIC and its driver. The flow director cannot currently be controlled by user programs. Support for packet-capturing applications is also a challenge, because of the lack of machine resources. Therefore, new approaches such as IDS and intrusion detection systems (IDS) are needed. Here, malloc can be replaced by another lightweight mechanism such as a slab allocator. The replacement of malloc by a slab allocator constitutes an aspect of GPGPU for high-performance regular expressions.

However, if the frequency of memory copies and software interrupts is high, the performance of the SF-TAP cell incubator may also help future IDS applications such as Snort [33], Bro [4], and Suricata [41]. Bro and Suricata are multithreaded and support high-bandwidth packet-capture implementation. BinPAC [24] is a DSL used by Bro for protocol parsing; however, Snort and Bro are single threaded and cannot manage high-bandwidth packet-capture applications. Wireshark [38] and tcpdump [35] are widely used traditional packet-capturing applications, and libnids [15] is used to support the future of the Internet. The L7 loopback interface of the flow director [10] of Intel’s NIC and its driver. The flow director cannot currently be controlled by user programs. Support for packet-capturing applications is also a challenge, because of the lack of machine resources. Therefore, new approaches such as IDS and intrusion detection systems (IDS) are needed. Here, malloc can be replaced by another lightweight mechanism such as a slab allocator. The replacement of malloc by a slab allocator constitutes an aspect of GPGPU for high-performance regular expressions.

We plan to improve the performance of the flow abstractor. Although the replacement of malloc by a slab allocator is trivial, it is not suitable for mobile devices, which are widely used in today’s society, because of the lack of machine resources. Therefore, new approaches such as IDS and intrusion detection systems (IDS) are needed. Here, malloc can be replaced by another lightweight mechanism such as a slab allocator. The replacement of malloc by a slab allocator constitutes an aspect of GPGPU for high-performance regular expressions.

However, if the frequency of memory copies and software interrupts is high, the performance of the SF-TAP cell incubator may also help future IDS applications such as Snort [33], Bro [4], and Suricata [41]. Bro and Suricata are multithreaded and support high-bandwidth packet-capture implementation. BinPAC [24] is a DSL used by Bro for protocol parsing; however, Snort and Bro are single threaded and cannot manage high-bandwidth packet-capture applications. Wireshark [38] and tcpdump [35] are widely used traditional packet-capturing applications, and libnids [15] is used to support the future of the Internet. The L7 loopback interface of the flow director [10] of Intel’s NIC and its driver. The flow director cannot currently be controlled by user programs. Support for packet-capturing applications is also a challenge, because of the lack of machine resources. Therefore, new approaches such as IDS and intrusion detection systems (IDS) are needed. Here, malloc can be replaced by another lightweight mechanism such as a slab allocator. The replacement of malloc by a slab allocator constitutes an aspect of GPGPU for high-performance regular expressions.

We plan to improve the performance of the flow abstractor. Although the replacement of malloc by a slab allocator is trivial, it is not suitable for mobile devices, which are widely used in today’s society, because of the lack of machine resources. Therefore, new approaches such as IDS and intrusion detection systems (IDS) are needed. Here, malloc can be replaced by another lightweight mechanism such as a slab allocator. The replacement of malloc by a slab allocator constitutes an aspect of GPGPU for high-performance regular expressions.
Other Features

- L7 Loopback interface for encapsulated flows
- Load balancing mechanism for application protocol analysers
- Separating and mirroring modes of SF-TAP cell incubator
- See more details in our paper
Conclusion

• We proposed SF-TAP for application level traffic analysis.

• SF-TAP has following features.
  • flow abstraction
  • running on commodity hardware
  • modularity
  • scalability

• We showed SF-TAP has achieved high performance in our experiments.