4-5 SF-TAP: Scalable and Flexible Traffic Analysis Platform

Yuuki TAKANO

Application-level network traffic analysis is a fundamental technology, which could be widely adopted such as intrusion detection system. However, network traffic analysis algorithms consuming considerable computational resources can not be adopted by using conventional network capturing technologies. In addition, developing an application protocol analyzer is a tedious and time consuming task. Therefore, in this paper, we propose a scalable and exible traffic analysis platform (SF-TAP) that provides an efficient and exible application-level stream analysis of high-bandwidth network traffic. Because SF-TAP is horizontally scalable, such heavy algorithms can be adopted for high-bandwidth networks. Furthermore, because of the modularity of SF-TAP, developers can easily implement network traffic analyzers by using SF-TAP.

1 Introduction

Analysis of network traffic constitutes the foundation technology of network security. Many network security applications — such as intruder detection system (IDS), firewall, and network forensics — use certain techniques of network traffic analysis as their core technology. The analysis of network traffic requires a mechanism to obtain essential chunks of information (IP packet, Ethernet frame, and others) from the network flow. Candidate techniques that can fill the needs include the BSD Packet Filter (BPF) [1] used in the BSD systems, PF_PACKET [2] used in Linux, and the libpcap [3] which is a wrapper library of the former two tools. However, it has become increasingly apparent that conventional simple techniques, typically the traffic capture mechanism provided by the libpcap library, fall short of the requirements to cope with the highly sophisticated cyberattacks (e.g. targeted attack [4]) and traffic analysis of the networks with ever increasing bandwidth. To address this situation, the author is conducting research and development on the infrastructure technology for analyzing network traffic that should be flexible enough to accommodate machine-learning based advanced analysis algorithms that are capable of handling wideband networks.

In concrete terms, the research and development is focused on the development of SF-TAP (Scalable and Flexible Traffic Analysis Platform) — a software infrastructure for network traffic analysis at the application levels. The SF-TAP provides the following five features to realize easy and efficient network traffic analysis: 1) flow abstraction mechanism, 2) modularity, 3) core scale design, 4) horizontal scale architecture, and 5) operation capability on commodity devices.

The architecture and design concept of SF-TAP, as well as performance evaluation, have already been published elsewhere [5]. This report focuses on more practical aspects of SF-TAP, including the specific designs and internal implementation, as well as the description of application cases.

2 Design

This section describes the design of the SF-TAP, and how scalability and flow abstraction capabilities are implemented in it.

2.1 Design overview

Figure 1 illustrates the operation concept of the SF-TAP. The SF-TAP consists of two components — SF-TAP Cell Incubator and SF-TAP Flow Analyzer — and an entity consisting of the combination of these two is called a SF-TAP Cell. The SF-TAP Cell Incubator is a component to realize horizontal scale, which captures traffic flowing in intra-networks and externally connected networks, then divides it into chunks on a flow unit basis, followed by transfer to multiple SF-TAP Cells. Conventionally, it has been considered difficult to transfer 10 Gbps network traffic at the wire-rate speed using commodity hardware. The author has successfully realized this high-speed transfer utilizing the netmap [6]. The SF-TAP Flow Abstractor is a

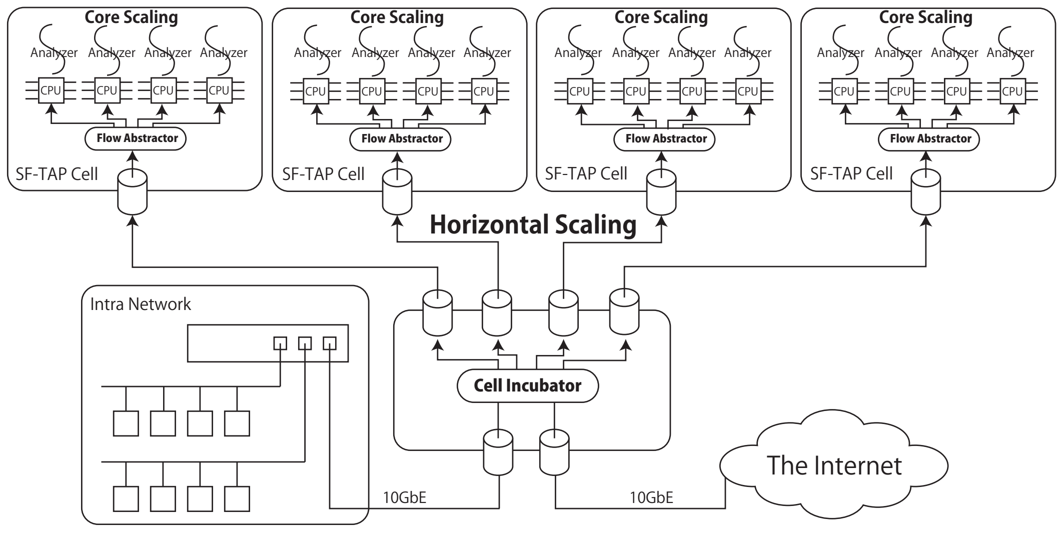


Fig. 1 SF-TP's operation scheme

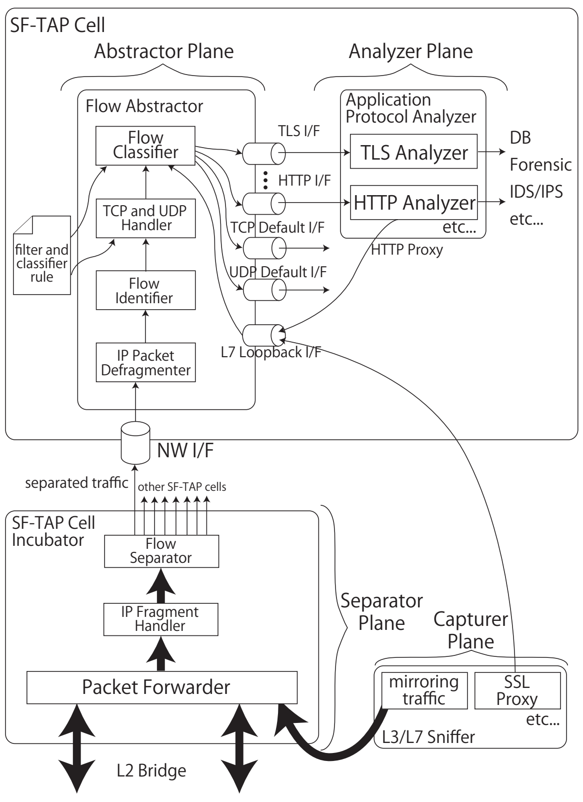


Fig. 2 Detailed architecture of the SF-TAP

component used to realize core scaling: it receives a flow from the SF-TAP Cell Incubator and reconstructs it before transferring it to multiple SF-TAP Flow Analyzers. In this way, SF-TAP is designed to guarantee multilayered scalability. This design allows flexible rescaling of hardware configuration in response to the trend of the computational load, leading to more efficient utilization of computational resources.

Figure 2 illustrates the architecture of the SF-TAP. ST-TAP's architecture consists of four Planes: Capture Plane, Separate Plane, Abstractor Plane, and Analyzer Plane. The following paragraphs describe the role played by each Plane.

The Abstractor Plane classifies and abstracts the flow. This plane performs such operations as IP fragment reassembly, flow identification, TCP stream reassembly, and identification of the application protocol using regular expression, before it sends the flow to an appropriate abstracted interface. The user of this platform can construct an analyzer using the abstract interfaces provided by the plane.

The Analyzer Plane, which includes the analyzer constructed by the platform user as its component, carries out an analysis of the application's protocol.

The Capture Plane plays the role of capturing network traffic. To be more specific, it corresponds to such mechanisms as the port mirroring of the L2/L3 switch and SSL proxy.

The Separate Plane is used to analyze wideband network traffic. This plane transfers the traffic that has been captured by its constituting component — i.e. SF-TAP Cell Incubator — to multiple SF-TAP Cells based on the flow information.

Among the four planes, this report gives only a brief mention to the Capture Plane, because it assumes the use of well-known mechanisms such as L2/L3 mirroring. Also, detailed discussion is not given to the Analyzer Plane, as it assumes implementation by the user of this platform.

Thus, the following sections place focus on the description of the SF-TAP Flow Abstractor and SF-TAP Cell Incubator, both of which are the core components of the SF-TAP.

2.2 SF-TAP Cell Incubator

The SF-TAP Cell Incubator performs L2 bridging, mirroring of wideband traffic, and load balancing on a flow unit basis. As shown in Fig. 2, the SF-TAP Cell Incubator architecture consists of the Packet Forwarder, IP Fragment Handler, and Flow Separator. The Packet Forwarder receives a L2 frame and transfers it to other NICs or IP Fragment Handlers. The IP Fragment Handler is a component used to divide a flow, whereby an IP fragment is taken into consideration. It appropriately performs flow identification (both fragmented and unfragmented packets) and transfer the L2 frame to a Flow Separator. Based on the flow information, the Flow Separator, in turn, transfers it to plural SF-TAP Cells.

Open vSwitch [7] [8] is one of the software tools capable of controlling traffic on a flow unit basis, but its ovf-ctrl is unable to handle an IP packet properly if it is fragmented. There are other software tools, such as iptables [9] and pf [10], which can perform traffic control using a L4 header, but they are also weak in terms of fragment handling. In addition, these software tools described above tend to exhibit performance problems when dealing with wideband traffic.

2.3 SF-TAP Flow Abstractor

The SF-TAP Flow Abstractor is a component used to perform such operations as reassembly of IP fragment packets, allocation of flow ID and classification of flow protocols, and it provides abstract flow interfaces to traffic analysis applications. As shown in Fig. 2, the SF-TAP Flow Abstractor is a multicomponent tool consisting of an IP Packet Defragmenter, Flow Identifier, TCP and UDP Hander, and Flow Classifier. The IP Packet Defragmenter reassembles IP fragment packets, and the TCP and UDP Handler reassembles TCP packets. Note that the TCP and UDP Handler transfers UDP packets to the Flow Classifier intact (UDP packets do not need any reassembly). The Flow Identifier identifies a flow based on the IP address, port number and Hop count (the number of re-injections into SF-TAP Flow Abstractor), and assign an ID to the flow. The flow ID generated by the Flow Identifier is used in subsequent operations such as reassembling TCP and in multi-thread processing of the flow.

The SF-TAP performs file based abstraction: a common approach with Plan 9 [11], BPF, and /dev of UNIX systems. Figure 3 illustrates the directory structure of the abstracted flow interface provided by the SF-TAP Abstractor. The flows, after being reassembled and having their IDs identified, are classified based on protocols and finally output to one of the files (UNIX Domain Socket) as shown in the figure. Analysis applications have to connect to these files to read in the flow to be analyzed. Note that the loopback7 is a specialized interface for analyzing tunneling protocols, such as Proxy, and used solely for re-injection. The default interface is used as the common output destination for all the flows that defy identification by the Flow Classifier.

As shown in Fig. 3, more than one interface is available to the HTTP protocol (http [0–3]). The purpose of this configuration is to enable load balancing of the HTTP flows and running analyzers on multiple CPUs. For example, CPU resources can be more efficiently utilized if four processes are simultaneously activated on a HTTP analyzer and each of them is connected to a different HTTP interface.

3 Implementation

Figure 4 shows the main classes and functions contained in the SF-TAP Flow Abstractor, as well as mutual relations among the threads. Based on the figure, this

```
$ ls -R /tmp/sf-tap
loopback7=         tcp/               udp/

/tmp/sf-tap/tcp:
default=           http0=             ssh=               dns=
smtp=              http1=             ftp=               http_proxy=
torrent_tracker=   http2=             irc=               websocket=
ssl=               http3=

/tmp/sf-tap/udp:
default=           dns=               torrent_dht=
```

Fig. 3 The directory structure of abstract flow interface

section describes how the SF-TAP Flow Abstractor is implemented.

3.1 Main classes of SF-TAP Flow Abstractor

The SF-TAP Flow Abstractor is constructed following the object-oriented concept, and implemented using C^{++} language. Thus, all the functions and variables are implemented as an object through the use of the classes. The following are the main classes contained in the ST-TAP Flow Abstractor.

fabs_pcap Captures Ethernet frames using pcap.
fabs_netmap Captures Ethernet frames using netmp.
fabs_ether Transfers the captured Ethernet frame to an appropriate function.
fabs_fragment Reassembles IPv4's IP fragment packets.
fabs_udp Processes UDP packet related jobs.
fabs_tcp Processes TCP packet related jobs.
fabs_appif Processes UNIX Domain Socket related jobs.
fabs_id Manages flow IDs.
fabs_id_dir A fabs_id class with additional flow direction information.

The classes fabs_pcap and fabs_netmap capture Ethernet frames using pcap and netmap, respectively. The Ethernet frame captured by either of these classes is handed over to the ether_input function that belongs to the fabs_ether class. When the Ethernet frame is transferred to the fabs_ether class its IP packet is extracted from the frame, followed by examination if it is an instance of a fragmented IPv4 packet. If it is, the packet is transferred to input_ip function (fabs_fragment class). If it is not an instance of an IPv4 packet, and if it is either a UDP or TCP packet, it is transferred to the fabs_udp or fabs_tcp class through the operator( ) function (fabs_callback class).

The fabs_udp class transfers the packet to the fabs_appif class without performing any modification. The fabs_tcp class performs the same operation, but with TCP flow reassembly before the transfer. The fabs_appif class identifies the application protocol used in the received packet, and controls (listen/accept/close) the UNIX Domain Socket including data input/output. Regular expressions — regular expression library re2 [12] — are used to identify the application protocol, and matching determination was made using the fabs_appif:appif_consumer class (i.e. in_datagram or send_tcp_data function). Following the identification procedure of the application protocol, the data is output to an appropriate Unix Domain Socket. Data output to UNIX Domain Socket starts with the header, followed by the payload. Header output. The write_head function (fabs_appif class) carries out the header output. Payload output is carried out by two functions: the in_datagram function for UDP, and the send_tcp_data function for TCP.

Two classes, fabs_id and fabs_id_dir, are provided to control flow Ids. The fabs_id class is used to uniquely identify the stream, and the fabs_id_dir class holds, in addition to the fabs_id information, the directional information (upstream/downstream) in the stream. The SF-TAP Flow Abstractor executes flow control based on the information provided by these flow ID control classes.

3.2 Threads of SF-TAP Flow Abstractor

This section describes thread handlings performed by the SF-TAP Flow Abstractor. The basic approach taken by the SF-TAP Flow Abstractor to utilize threads relies on the producer-consumer pattern. Bottlenecks in multithread programming often come from synchronization processing: if inadequately implemented, it can cause drastically reduced system performance and plague the system with hordes of multithread-related bugs (e.g. deadlock). However, use of producer-consumer patterns can reduce the amount of data shared among threads, simplifying synchronization processing.

The SF-TAP Flow Abstractor contains a low-level, spinlock based, synchronization mechanism independently which makes use of atomic operations. The reason for this implementation is to address the generation of function-call overhead (typically such function as pthread_mutex), as well as the possibility of transfer of the thread processing to the OS's scheduler. Handover of thread processing to the OS's scheduler gives rise to a context switching, which is highly likely to generate a large delay in synchronizing among the threads. To avoid this, an independent synchronization mechanism is implemented. Low-level synchronization processing is realized by two classes: fabs_spin_lock class and fabs_spin_rwlock class. The fabs_spin_lock class contains simple spin-lock functions, and the fabs_spin_rwlock class contains readers-writer lock functions.

The SF-TAP Flow Abstractor uses, as shown in Fig. 4, the following five types of threads: Capture thread, IP Defragment thread, TCP/UDP thread, UX thread, and Classification thread. The Capture thread captures Ethernet frames using pcap or netmap. The IP Defragment thread reassembles fragmented packets of IPv4. The TCP/UDP

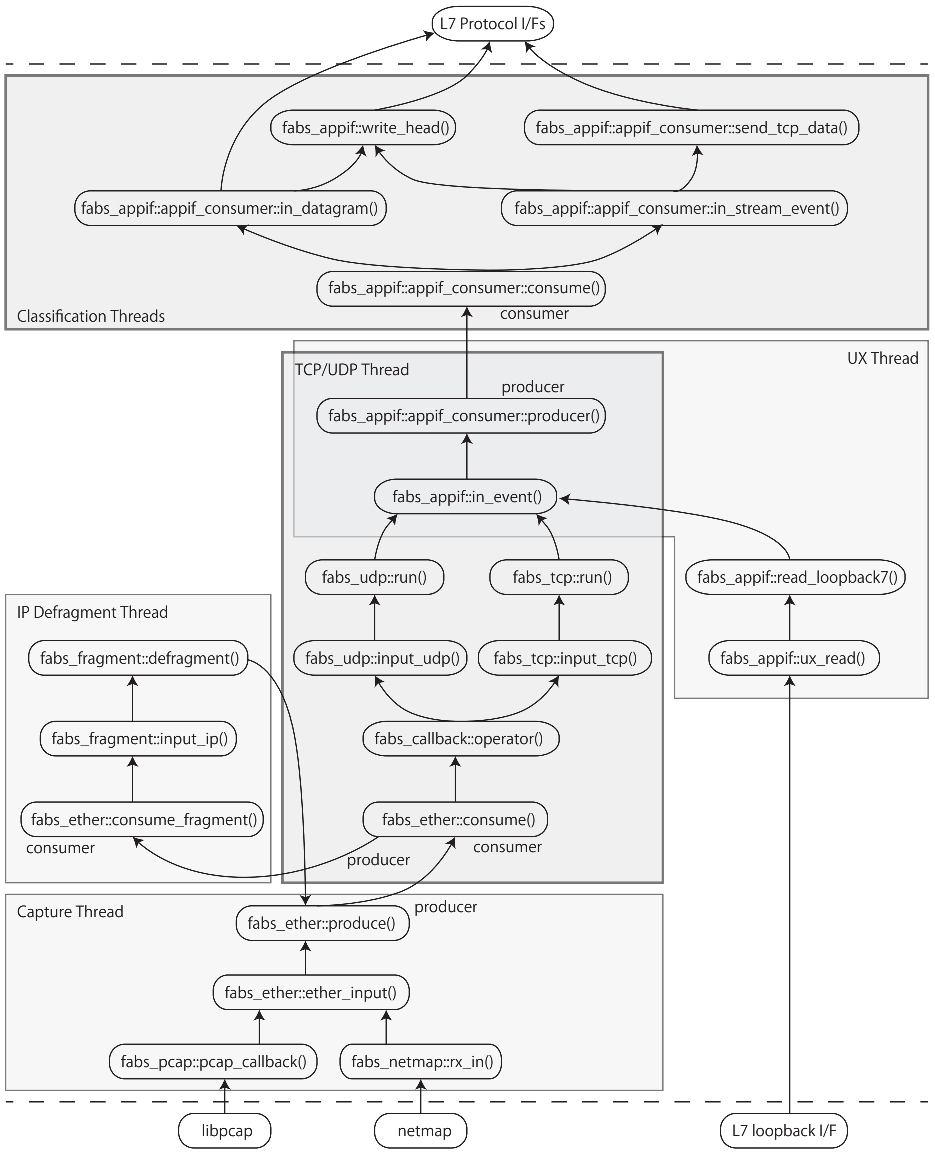

L7 Protocol I/Fs
fabs_appif::write_head()
fabs_appif::appif_consumer::send_tcp_data()
fabs_appif::appif_consumer::in_datagram()
fabs_appif::appif_consumer::in_stream_event()
fabs_appif::appif_consumer::consume()
Classification Threads
consumer
UX Thread
TCP/UDP Thread
producer
fabs_appif::appif_consumer::producer()
fabs_appif::in_event()
fabs_udp::run()
fabs_tcp::run()
fabs_appif::read_loopback7()
IP Defragment Thread
fabs_fragment::defragment()
fabs_udp::input_udp()
fabs_tcp::input_tcp()
fabs_appif::ux_read()
fabs_fragment::input_ip()
fabs_callback::operator()
fabs_ether::consume_fragment()
fabs_ether::consume()
consumer
producer
consumer
Capture Thread
fabs_ether::produce()
producer
fabs_ether::ether_input()
fabs_pcap::pcap_callback()
fabs_netmap::rx_in()
libpcap
netmap
L7 loopback I/F

Fig. 4 Implementations of SF-TAP Flow Abstractor

thread performs various processing related to TCP and UDP. The UX thread performs UNIX Domain Socket related processing such as listen, accept, and close. The Classification thread identifies the protocol used in applications and outputs data to UNIX Domain Socket.

4 Application cases

Up till now, the SF-TAP has been applied to investigative researches on DNS Open Resolver and third-party web tracking. This section gives a description of these application cases.

4.1 Investigative survey on actual situations in which DNS open resolver is used

The DNS amp attack [13] [14] is a mode of DDoS attack strategies frequently reported from the year around 2013, which is characterized by the technique that exploits DNS servers (DNS open-resolver) that respond to the requests from unspecified multiple people. Because there are many DNS open-resolvers distributed on the Internet, the attacker tries to invade by sending a falsified DNS query (the source IP address is masquerading as the IP address of the attack target) to a DNS open-resolver. The author conducted wide-area investigation on the DNS open-resolvers that may serve as the cause for DNS amp attacks [15]–[17]. Figure 5 shows global distribution of DNS open-resolvers plotted on a world map. The investigation revealed that, as is apparent from this figure, numerous DNS open-resolvers are distributed worldwide.

To research into the distribution of DNS open-servers on the internet, use was made of a system consisting of the

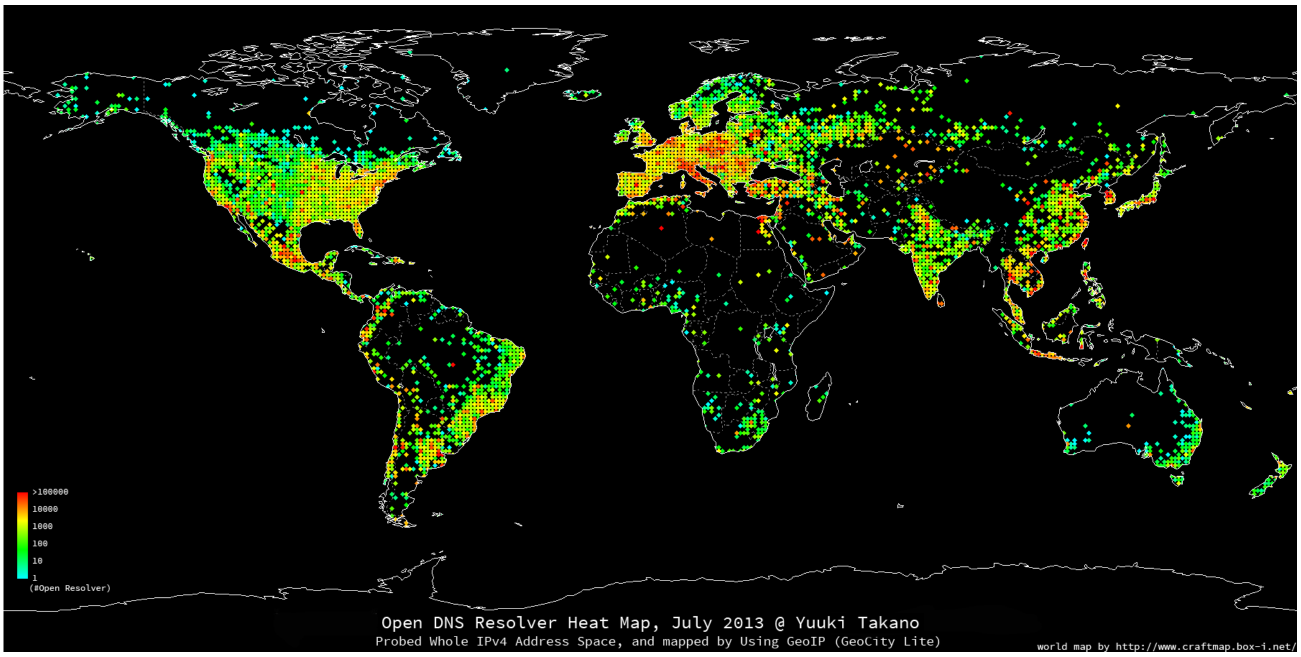

>100000
10000
1000
100
10
1
(#Open Resolver)
Open DNS Resolver Heat Map, July 2013 @ Yuuki Takano
Probed Whole IPv4 Address Space, and mapped by Using GeoIP (GeoCity Lite)
world map by http://www.craftmap.box-i.net/

Fig. 5 Global distribution of DNS open-resolvers (as of Jul. 2013)

following three components: a DNS Prober that explores DNS open-resolvers on the Internet, a Reverse Lookupper that traces back the data from the DNS Prober to locate the DNS open-resolver, and a Statistical Analyzer to obtain statistical information. A prototype version of the SF-TAP software was used, embedded in the DNS Prober and Reverse Lookupper, as the DNS packet analysis engine. As exemplified in this investigation, application of an appropriate traffic analysis mechanism makes the measurement and analysis of DNS servers much easier.

The results of this research have been cited as a reference in several papers [18]–[20]. Especially noteworthy for those interested in this field is the paper written by Kührer et.al. and presented in ACM IMC 2015 (the flagship conference in Internet measurement), which, in addition to the approach the author employed, carried out more detailed investigation including the device fingerprint of DNS open-resolvers.

4.2 Investigative research on third-party web tracking

Promotional websites and social networking sites usually scrape personal web browsing history secretly. This activity — called third-party web tracking — has become subject of debate as it may constitutes a serious breach of privacy [21] [22]. By the nature of this activity, many uses are suffering privacy breach without noticing it. Many users use these sites, especially social networking sites, by their real names, providing an easy link between their real names and web browsing history. The author is conducting research and development aiming at realization of informed consent in web advertising. The term “informed consent”, normally used in health-care field, means here that the user browses websites only based on a clear understanding and consent concerning what part of his/her private information may be transferred to the website.

The MindYourPrivacy [23] is a system of the author’s own development for the visualization of third-party web tracking. It first analyzes HTTP traffic using a prototype implementation of the SF-TAP, and then stores the data in MongoDB [24]. Then, it analyzes the stored data to create a visual presentation of third-party web tracking for the user. Figure 6 shows an illustrative result obtained from this system (a result from the demonstrative experiment presented in the WIDE training camp – a meeting of network researchers).

In this study, the author also proposed a method to extract — through clustering of the graph — the sites that are actually performing third-party web tracking. The results of MCODE [25]-based clustering are shown in the lower part of Fig. 6, The MindYourPrivacy stores the results of network traffic analysis in MongoDB for subsequent batched data processing. Visualization processing, as shown in Fig. 6, was performed manually using Cytoscape [26]. CHAKRA is an attempt to automatically carry out big data analysis on an online basis up to the point of visual presentation, i.e. a big data player with visualization capability. Figure 7 shows an example of graph visualized by

CHAKRA. CHAKRA employs SF-TAP as the network traffic analysis tool, and draws graphics using an algorithm in which a spring model is applied on Riemann manifold [27]. A demonstrative exhibition of SF-TAP and CHAKRA was staged in Interop Tokyo 2015, the largest business show in Japan in the field of networking, where the SF-TAP was awarded the grand-prix in the science section and CHAKRA was award a special prize in the demonstration section of ShowNet.

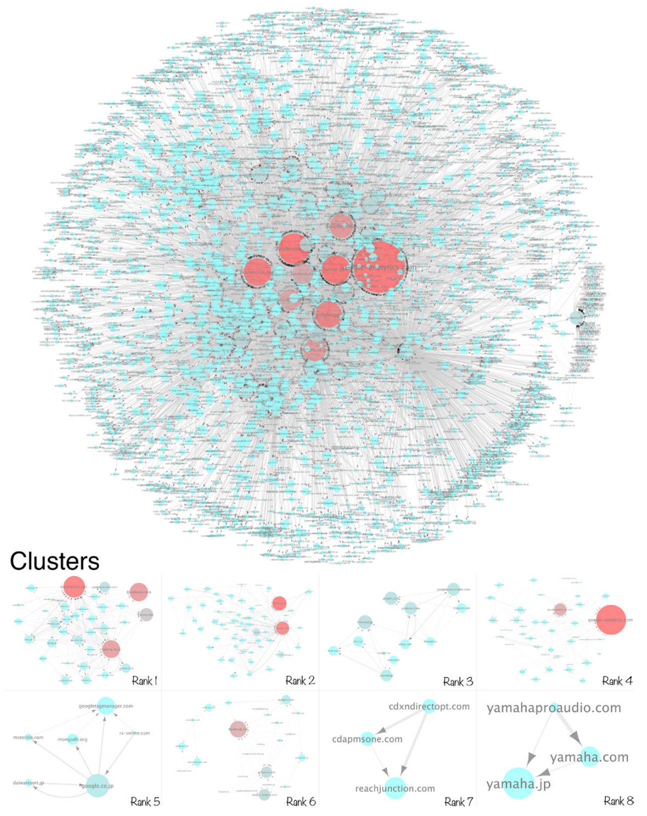
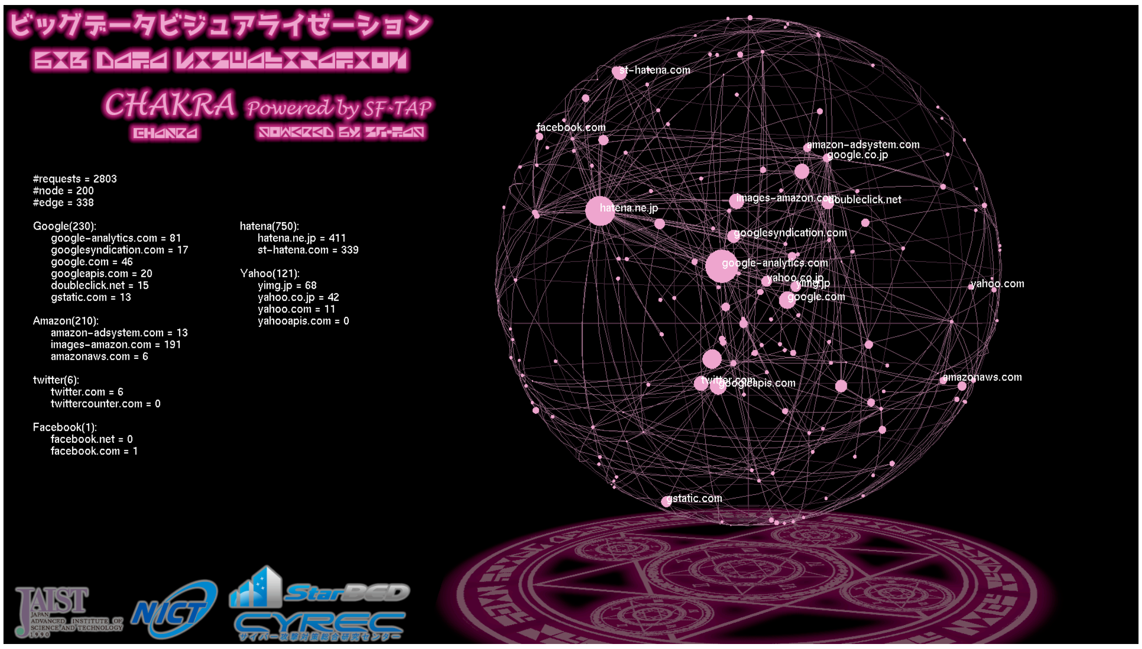


Fig. 6 Data presented in WIDE training camp (Sept. 2013)

5 Related researches

Traditional packet capture and analysis software tools, such as tcpdump [3] and WireShark [28], are still in wide use. The libnids [29] is a library for analyzing network traffic that can be used not only to capture packets but also to reassemble TCP flow. These tools operate basically on a single thread and are therefore unfit for the analysis of wideband traffic.

Tools capable of flow-level analysis of wideband traffic have also been proposed — i.e. SCAP [30] and GASPP [31] — and they proved the feasibility of realizing flow-level analysis of 10 Gbps traffic on commodity hardware. The SCAP works within Linux kernel, and is implemented with an acceleration mechanism — i.e. thread allocation to NIC's transmission and receive queue. It comes with another mechanism, called Subzero-Copy Packet Transfer, which makes it possible to analyze only the required traffic selectively. GASPP is an analysis engine that implement its functions through the use of GPGPU: the use of netmap [6] enables fast memory-to-memory transfer between NIC and GPU. The approach proposed by the author is similar to GASPP in view of its attempt to accelerate flow level analysis, but major differences exist in such aspects as: considerations on the scalability of flow analysis section,


```
#requests = 2803
#node = 200
#edge = 338

Google(230):                         hatena(750):
    google-analytics.com = 81            hatena.ne.jp = 411
    googlesyndication.com = 17           st-hatena.com = 339
    google.com = 46
    googleapis.com = 20              Yahoo(121):
    doubleclick.net = 15                 yimg.jp = 68
    gstatic.com = 13                     yahoo.co.jp = 42
                                         yahoo.com = 11
Amazon(210):                             yahooapis.com = 0
    amazon-adsystem.com = 13
    images-amazon.com = 191
    amazonaws.com = 6

twitter(6):
    twitter.com = 6
    twittercounter.com = 0

Facebook(1):
    facebook.net = 0
    facebook.com = 1
```

Fig. 7 CHAKRA: a big data visualization system

and equipment of common interface for easy realization of modularity.

Several types of high-speed packet capture frameworks have been proposed, i.e. netmap, DPDK [32], and PF_RING [33]. Conventional methods normally involve a proliferation of copying operations and interrupts in the transactions among NIC, kernel, and the user, which make capturing in wideband traffic quite difficult. One of the objectives of the proposed methods is to reduce the frequency of memory copy operations and interrupts drastically, making packet capturing at a 10 Gbps wire-rate a possibility. The author's implementation uses netmap for capturing network traffic.

Several software tools, such as nDPI [34], libprotoident [35], l7 –filter [36], and PEAFLOW [37], have been proposed to classify traffic at application levels. To identify the protocol used in the applications, these tools use either the Aho-Corasik [38] method or the regular expression pattern matching. PEAFLOW uses the parallel programing language FastFlow to realize fast traffic classification.

The IDS software packages, such as Snort [39], bro [40], and Suricata [41], perform flow reassembly and analysis at the application level. One of them, bro, uses a protocol server language binpac [42] to enable analysis at the application level. However, because Snort and bro operate on a single thread, they are unfit for the analysis of wideband traffic. On the other hand, Suricata is operable on multi-thread and capable of handling wider band traffic. One of the characteristics inherent to these packages is the close coupling between functions, typically the sections for flow reassembly and analysis at the application level. This hinders flexible modification of operation logics, and these packages tend to be bound to the rule description method and domain specific language provided by the software.

Schneider et al. [43] proposed an architecture to scale out through flow unit basis division of 10 Gbps traffic. Note, however, that they only demonstrated the validity of the method on 1 Gbps traffic, and the feasibility on 10 Gbps traffic still remains at a conceptual stage. SF-TAP, on the other hand, is implemented with the software to divide 10 Gbps traffic on a flow unit basis and its functionality has been verified.

Click [44], SwitchBlade [45] and ServerSwitch [46] have been conducting researches to modularize network switches, thus enabling this function to be deployed on the network in a highly flexible and programmable fashion. Based on their approach, the configuration of analysis logic in the SF-TAP is also modularized to incorporate programmability in analysis procedures.

BPF [1] is a well-known mechanism for capturing packets which abstracts network traffic using a method similar to the UNIX's /dev. In the SF-TAP, the concept used in BPF is further extended to enable traffic abstraction at flow levels. This also supports modularization of analysis logic and core scaling.

6 Conclusion

In this report, the author gave an explanation on the SF-TAP, a flexible and highly scalable infrastructure for network traffic analysis. With the use of the SF-TAP, network traffic analysis at application levels becomes much easier than the conventional network traffic capturing techniques: for example, the technique that uses libpcap. Typically, the conventional techniques present difficulties in applying a sophisticated analysis method such as machine learning. This favorable feature owes much to the modularity- and scalability-oriented design of the SF-TAP.

The SF-TAP consists of two main components: SF-TAP Cell Incubator and ST-TAP Flow Abstractor. The SF-TAP Cell Incubator is a mechanism that enables two or more machines to analyze network traffic, thus it supports the realization of system scalability. The SF-TAP Flow Abstractor assumes the role of flow abstraction, whereby the file-based abstraction - which is also utilized in such tools as UNIX's /dev, BFP, and Plan 9 - is used. This abstraction procedure enables the realization of modularity and effective utilization of multiple CPU cores.

This report also describes how the SF-TAP Flow Abstractor is implemented. In terms of session layer protocols, the current SF-TAP Flow Abstractor is capable of handling only TCP and UDP. If it is to handle other protocols, such as QUIC in the future, the implementation method described in this report will serve as a useful reference. This is also true for addressing additional data frame link capturing mechanisms such as DPDK in the future: the current SF-TAP is capable of using only libpcap and netmap.

This report also gives some of illustrative applications of the SF-TAP. So far, the SF-TAP has been effectively utilized in such investigative researches as those on DNS open resolver and third-party web tracking. As the nature of these cases indicate, the methods used to analyze network traffic represent the infrastructure technology to maintain network security, and they are of critical importance in implementing algorithmic research and development in important subjects such as IDS. The author is planning to continue SF-TAP based research and development placing stronger focus on utility in the real world. The target areas include IDS, network traffic engineering, and network forensics.

References

1 Steven McCanne and Van Jacobson, "The BSD Packet Filter: A New Architecture for User-level Packet Capture" In Proceedings of the Usenix Winter 1993 Technical Conference, San Diego, Cali-fornia, USA, January 1993, pp.259–270. USENIX Association, 1993.

2 PF PACKET. https://www.kernel.org/doc/Documentation/networking/filter.txt.

3 TCPDUMP/LIBPCAP public repository. http://www.tcpdump.org/.

4 Olivier Thonnard, Leyla Bilge, Gavin O'Gorman, Se_an Kiernan, and Martin Lee, "Industrial Espi-onage and Targeted Attacks: Understanding the Characteristics of an Escalating Threat" In Davide Balzarotti, Salvatore J. Stolfo, and Marco Cova, editors, Research in Attacks, Intrusions, and De-fenses - 15th International Symposium, RAID 2012, Amsterdam, The Netherlands, Sept. 12–14, 2012. Proceedings, vol.7462 of Lecture Notes in Computer Science, pp.64–85. Springer, 2012.

5 Yuuki Takano, Ryosuke Miura, Shingo Yasuda, Kunio Akashi, and Tomoya Inoue, "SF-TAP: Scalable and Flexible Traffic Analysis Platform Running on Commodity Hardware" In 29th Large Installa-tion System Administration Conference (LISA15), pp.25–36, Washington, D.C., November 2015. USENIX Association.

6 Luigi Rizzo and Matteo Landi, "netmap: memory mapped access to network devices" In Srinivasan Keshav, Jörg Liebeherr, John W. Byers, and Jeffrey C. Mogul, editors, SIGCOMM, pp.422–423. ACM, 2011.

7 Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, Keith Amidon, and Martin Casado, "The Design and Implementation of Open vSwitch" In 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), pp.117–130, Oakland, CA, May 2015. USENIX Association.

8 Open vSwitch. http://openvswitch.github.io/.

9 net_lter/iptables project homepage - The net_lter.org "iptables" project. http://www.netfilter.org/projects/iptables/.

10 PF: The OpenBSD Packet Filter. http://www.openbsd.org/faq/pf/.

11 Rob Pike, David L. Presotto, Sean Dorward, Bob Flandrena, Ken Thompson, Howard Trickey, and Phil Winterbottom, "Plan 9 from Bell Labs" Computing Systems, vol.8, no.2, pp. 221–254, 1995.

12 RE2. https://github.com/google/re2.

13 Marios Anagnostopoulos, Georgios Kambourakis, Panagiotis Kopanos, Georgios Louloudakis, and Stefanos Gritzalis, "DNS ampli_cation attack revisited" Computers & Security, vol.39, pp.475–485, 2013.

14 US-CERT Alert(TA13-088A) DNS Ampli_cation Attacks. http://www.us-cert.gov/ncas/alerts/TA13-088A.

15 Ruo Ando, Yuuki Takano, and Satoshi Uda, "Unraveling large scale geographical distribution of vulnerable DNS servers using asynchronous I/O mechanism", 2013.

16 Yuuki Takano, Ruo Ando, Takeshi Takahashi, Satoshi Uda, and Tomoya Inoue, "A measurement study of open resolvers and DNS server version" In Internet Conference 2013, pp.23–32, 2013.

17 Yuuki Takano, Ruo Ando, Takeshi Takahashi, Satoshi Uda, and Tomoya Inoue, "The Ecology of DNS Open Resolvers" IEICE Transaction, Vol.J97-B, No.10, pp.873–889, 2014.

18 Marc Kührer, Thomas Hupperich, Jonas Bushart, Christian Rossow, and Thorsten Holz, "Going Wild: Large-Scale Classi_cation of Open DNS Resolvers" In Kenjiro Cho, Kensuke Fukuda, Vivek S. Pai, and Neil Spring, editors, Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015, Tokyo, Japan, Oct. 28-30, 2015, pp.355–368. ACM, 2015.

19 Hajime Tazaki, Kazuya Okada, Yuji Sekiya, and Youki Kadobayashi, "MATATABI : Multi-layer Threat Analysis Platform with Hadoop" In IEICE ICSS, pp.113–118, 2014.

20 Saeed Abbasi, "Investigation of Open Resolvers in DNS Reection DDoS Attacks", 2014.

21 Jonathan R. Mayer and John C. Mitchell, "Third-Party Web Tracking: Policy and Technology" In IEEE Symposium on Security and Privacy, SP 2012, 21–23 May 2012, San Francisco, California, USA, pp.413–427. IEEE Computer Society, 2012.

22 Franziska Roesner, Tadayoshi Kohno, and DavidWetherall, "Detecting and Defending Against Third-Party Tracking on the Web" In Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), pp.155–168, San Jose, CA, 2012. USENIX.

23 Yuuki Takano, Satoshi Ohta, Takeshi Takahashi, Ruo Ando, and Tomoya Inoue, "MindYourPrivacy: Design and implementation of a visualization system for third-party Web tracking" In Ali Miri, Urs Hengartner, Nen-Fu Huang, Audun J_sang, and Joaqu_n Garc_a-Alfaro, editors, 2014 Twelfth Annual International Conference on Privacy, Security and Trust, Toronto, ON, Canada, July 23-24, 2014, pp.48–56. IEEE, 2014.

24 MongoDB. http://www.mongodb.org/.

25 Gary D. Bader and Christopher W. V. Hogue, "An automated method for _nding molecular complexes in large protein interaction networks" BMC Bioinformatics, vol.4, p.2, 2003.

26 Cytoscape: An Open Source Platform for Complex Network Analysis and Visualization. http://www.cytoscape.org/.

27 Stephen G. Kobourov and Kevin Wampler, "Non-Euclidean Spring Embedders" IEEE Trans. Vis. Comput. Graph., vol.11, no.6, pp.757–767, 2005.

28 Wireshark - Go Deep. https://www.wireshark.org/.

29 libnids. http://libnids.sourceforge.net/.

30 Antonis Papadogiannakis, Michalis Polychronakis, and Evangelos P. Markatos, "Scap: stream-oriented network traffic capture and analysis for high-speed networks" In Konstantina Papagiannaki, P. Krishna Gummadi, and Craig Partridge, editors, Internet Measurement Conference, IMC'13, Barcelona, Spain, Oct. 23–25, 2013, pp.441–454. ACM, 2013.

31 Giorgos Vasiliadis, Lazaros Koromilas, Michalis Polychronakis, and Sotiris Ioannidis, "GASPP: AGPU-Accelerated Stateful Packet Processing Framework" In Garth Gibson and Nickolai Zeldovich, editors, 2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19–20, 2014., pp.321–332. USENIX Association, 2014.

32 Intel@ DPDK: Data Plane Development Kit. http://www.ntop.org/products/pf_ring/.

33 PF RING. http://www.ntop.org/products/pf_ring/.

34 nDPI. http://www.ntop.org/products/ndpi/.

35 WAND Network Research Group: libprotoident. http://research.wand.net.nz/software/libprotoident.php.

36 L7-_lter | ClearFoundation. http://l7-filter.clearfoundation.com/.

37 Marco Danelutto, Luca Deri, D. De Sensi, and Massimo Torquati, "Deep Packet Inspection on Commodity Hardware using FastFlow" In Michael Bader, Arndt Bode, Hans-Joachim Bungartz, Michael Gerndt, Gerhard R. Joubert, and Frans J. Peters, editors, PARCO, vol.25 of Advances in Parallel Computing, pp.92–99. IOS Press, 2013.

38 Alfred V. Aho and Margaret J. Corasick, "Efficient string matching: An aid to bibliographic search" Commun. ACM, vol.18, no.6, pp.333–340, 1975.

39 Snort :: Home Page. https://www.snort.org/.

40 The Bro Network Security Monitor. http://www.bro.org/.

41 Suricata | Open Source IDS / IPS / NSM engine. http://suricata-ids.org/.

42 Ruoming Pang, Vern Paxson, Robin Sommer, and Larry L. Peterson, "binpac: a yacc for writing application protocol parsers" In Jussara M. Almeida, Virg__lio A. F. Almeida, and Paul Barford, editors, Internet Measurement Conference, pp.289–300. ACM, 2006.

43 Fabian Schneider, Jörg Wallerich, and Anja Feldmann, "Packet Capture in 10-Gigabit Ethernet Environments Using Contemporary Commodity Hardware" In Steve Uhlig, Konstantina Papagian-naki, and Olivier Bonaventure, editors, PAM, Vol.4427 of Lecture Notes in Computer Science, pp.207–217. Springer, 2007.

44 Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek, "The click modular router. ACM Trans" Comput. Syst., vol.18, no.3, pp.263–297, 2000.

45 Muhammad Bilal Anwer, Murtaza Motiwala, Muhammad Mukarram Bin Tariq, and Nick Feamster, "SwitchBlade: a platform for rapid deployment of network protocols on programmable hardware" In Shivkumar Kalyanaraman, Venkata N. Padmanabhan, K. K. Ramakrishnan, Rajeev Shorey, and Geoffrey M. Voelker, editors, Proceedings of the ACM SIGCOMM 2010 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, New Delhi, India, Aug. 30 –Sept. 3, 2010, pp.183–194. ACM, 2010.

46 Guohan Lu, Chuanxiong Guo, Yulong Li, Zhiqiang Zhou, Tong Yuan, Haitao Wu, Yongqiang Xiong, Rui Gao, and Yongguang Zhang, "ServerSwitch: A Programmable and High Performance Platform for Data Center Networks" In David G. Andersen and Sylvia Ratnasamy, editors, Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2011, Boston, MA, USA, March 30–April 1, 2011, pp.15–28. USENIX Association, 2011.

Yuuki TAKANO, Ph.D.

Researcher, Cybersecurity Laboratory,
Cybersecurity Research Institute
Computer Architecture, Network System,
Network Security