4-5　柔軟で規模追従可能なトラフィック解析基盤 SF-TAP

高野祐輝

アプリケーションレベルでのネットワークトラフィック解析は、ネットワークセキュリティの基礎となる技術であり、侵入検知システムをはじめ、様々な場所で活用可能である。しかしながら、従来までのネットワークトラフィック解析技術では、そのパフォーマンスの低さから、機械学習などといった、計算資源を大量に消費するアルゴリズムを適用するのは難しかった。また、アプリケーションレベルでの解析器を実装するのは、非常に手間のかかる作業であった。そこで、本稿では、これら問題を解決するために、柔軟で規模追従可能なネットワークトラフィック解析基盤 SF-TAP の提案を行う。SF-TAP は水平スケール可能なアーキテクチャであるため、SF-TAP を利用すると、広帯域なネットワークに対しても、計算資源を大量に消費するネットワークトラフィック解析アルゴリズムを適用可能になる。さらに、SF-TAP の備えるモジュラリティにより、容易に、解析器の実装を行うことが可能となる。

1　はじめに

ネットワークトラフィック解析はネットワークセキュリティの基礎となる技術である。例えば、侵入検知システム（IDS）、ファイアウォール、ネットワークフォレンジックといった多くのネットワークセキュリティソフトウェアで、ネットワークトラフィック解析技術をコア技術として利用している。ネットワークトラフィック解析を行うためには、ネットワーク中に流れる IP パケットやイーサネットフレームを取得するための機構が必要であり、これを行う技術として、BSD の The BSD Packet Filter（BPF）[1] や Linux の PF_PACKET [2]、またはこれらをラップしたライブラリである libpcap[3] がある。しかしながら、標的型攻撃 [4] に見られるようなサイバー攻撃の高度化及びネットワークの広帯域化に対応したネットワークトラフィック解析を行うためには、libpcap など従来までの単純なトラフィックキャプチャ機構のみで対応するのは難しくなってきている。そこで、我々は、広帯域なネットワークに対しても、機械学習などを用いた高度な解析アルゴリズムを柔軟に適用できるようなネットワークトラフィック解析基盤の研究・開発を行っている。

SF-TAP（Scalable and Flexible Traffic Analysis Platform）[5] は、現在、我々が研究開発を行っている、アプリケーションレベルでネットワークトラフィック

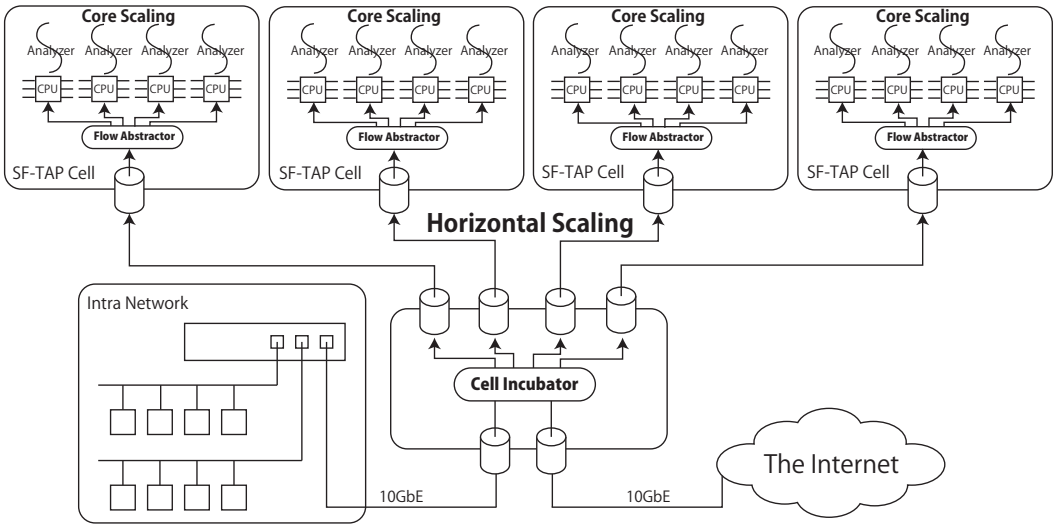

図 1　SF-TAP の動作図

解析を行うためのソフトウェア基盤である。SF-TAP は、1. フロー抽象化機構、2. モジュラリティ、3. コアスケール設計・実装、4. 水平スケールなアーキテクチャ、5. コモディティ機器上での動作という 5 つの特徴を備えており、SF-TAP を利用すると、容易かつ効率的にネットワークトラフィック解析を行うことができるようになる。

本稿では、この SF-TAP について解説を行う。ただし、SF-TAP のアーキテクチャや設計思想、パフォーマンス評価などの内容については既に発表済み [5] であるため、本稿では設計、内部的な実装方法及び応用事例といった、より実践的な視点から説明を行う。

2　設計

本節では、SF-TAP の設計について説明し、どのようにして、SF-TAP がスケーラビリティとフローの抽象化を行っているかについて解説する。

2.1　設計概要

図 1 は SF-TAP の動作概念図を示している。SF-TAP は、SF-TAP Cell Incubator、SF-TAP Flow Abstractor という 2 つのコンポーネントから構成され、SF-TAP Flow Abstractor と Analyzer を合わせて SF-TAP Cell と呼ぶ。SF-TAP Cell Incubator は水平スケールを実現するコンポーネントであり、これはイントラネットワークや対外接続ネットワーク上を流れるトラフィックをキャプチャした後、フロー単位でトラフィック分割を行い複数の SF-TAP Cell へ転送する。従来、コモディティハードウェアを用いて 10 Gbps のネットワークトラフィックをワイヤレートで転送することは難しかったが、我々は netmap [6] を用いてこれを実現した。SF-TAP Flow Abstractor はコアスケールを実現するコンポーネントであり、SF-TAP Cell Incubator から受け取ったフローを再構成し、複数の Analyzer へ転送する。このように、SF-TAP は多段にスケーラビリティを確保する設計となっている。そのため、トラフィック量や Analyzer の計算量に応じて、柔軟にハードウェア構成の規模を変更することができ、計算リソースを有効に活用することが可能になる。

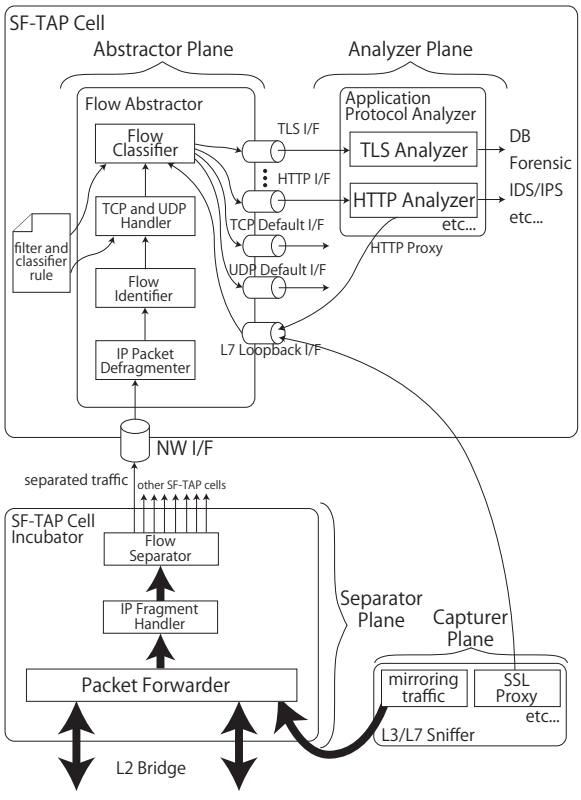

図 2　SF-TAP の詳細アーキテクチャ

図 2 は SF-TAP のアーキテクチャを示している。SF-TAP は、Capturer Plane、Separator Plane、Abstractor Plane、Analyzer Plane の 4 つの Plane から構成されるアーキテクチャとなっている。それぞれの Plane は以下で示す役割を果たす。

Abstractor Plane はフローの分類と抽象化を行う Plane となる。本 Plane では、IP フラグメント再構成、フロー識別、TCP ストリーム再構成、正規表現によるアプリケーションプロトコル判別を行い、最終的に適切な抽象化インターフェースへと出力する。本プラットフォームの利用者は、本 Plane にて提供される抽象化インターフェースを用いて解析器を作成する。

Analyzer Plane は、アプリケーションプロトコルを解析する Plane となり、本プラットフォームの利用者が作成した解析器が、本 Plane の構成要素となる。

Capturer Plane はネットワークトラフィックをキャプチャする Plane となる。具体的には、L2 /L3 スイッチのポートミーラリング機構や、SSL プロクシなどが相当する。

Separator Plane は広帯域ネットワークトラッフィクを解析する際に用いられる Plane となる。本 Plane では、構成要素である SF-TAP Cell Incubator がキャプチャしたトラフィックを、フロー情報に基づいて複数の SF-TAP Cell へと転送する。

なお、4 つの Plane のうち、Capturer Plane は L2 /L3 スイッチのミーラリング機構など、広く知られた機構の利用を想定しているため、本稿では詳細について言及しない。また、Analyzer Plane についても、本プラットフォームの利用者が実装することを想定しているため、ここについても詳細を述べない。

以下、本節では、SF-TAP のコアコンポーネントである、SF-TAP Flow Abstractor と SF-TAP Cell

Incoubator について解説する。

2.2　SF-TAP Cel Incubator

SF-TAP Cell Incubator は広帯域トラフィックの L2 ブリッジ、ミラーリング、フロー単位でのロードバランスを行う。SF-TAP Cell Incubator のアーキテクチャは、図 2 で示すとおりであり、Packet Fowarder、IP Fragment Handler、Flow Separator から構成される。Packet Fowarder は L2 フレームを受信し、これを、他の NIC、もしくは IP Fragment Handler へと転送する。IP Fragment Handler は IP フラグメントを考慮してフロー分割を行うためのコンポーネントである。IP Fragment Handler は、フラグメントされいないパケットと、されているパケットの両方を適切にフロー識別を行い、Flow Separator へ L2 フレームを転送する。Flow Separator は、フロー情報に基づいて複数の SF-TAP Cell へと転送する。

フロー単位でトラフィックコントロール可能なソフトウェアとして、Open vSwitch [7][8] があるが、Open vSwitch の ovf-ctrl は、フラグメント化された IP パケットを適切に扱うことはできない。iptables [9] や pf [10] も、L4 ヘッダを用いたトラフィックコントロールが可能であるが、これらもまた、フラグメントに対応するのは難しい。さらに、上記ソフトウェアは広帯域トラフィック対応という点でパフォーマンスに問題がある。

2.3　SF-TAP Flow Abstractor

SF-TAP Flow Abstractor は、IP フラグメントパケット再構成、フロー ID 割当、TCP フローの再構成、フローのプロトコル分類を行い、トラフィック解析アプリケーションに対して抽象フローインターフェースを提供するコンポーネントである。図 2 のように、SF-TAP Flow Abstractor は、IP Packet Defragmenter、Flow Identifier、TCP and UDP Handler、Flow Classifier から構成される。IP Packet Defragmenter は、IP フラグメントパケットの再構成を、TCP and UDP Handler は、TCP パケットの再構成を行う。UDP パケットの場合は再構成する必要が無いため、何もせず、TCP and UDP Handler から Flow Classifier へ渡される。Flow Identifier は、フロー ID を、IP アドレス、ポート番号及び SF-TAP Flow Abstractor へ再注入された回数を示す Hop カウントから、フローの識別を行い、フロー ID を割り当てる。ここで生成されたフロー ID は、TCP の再構成や、マルチスレッドでのフロー処理に用いられる。

SF-TAP では、Plan 9 [11] や、BPF、UNIX の /dev のようにファイルを用いた抽象化を行う。図 3 は、SF-TAP Flow Abstractor が提供する抽象フローインターフェースのディレクトリ構造である。再構成、ID 識別されたフローはプロトコル別に分類され、最終的にこの図のようなファイル（UNIX Domain Socket）へ出力される。解析アプリケーションは、これらのファイルへ接続して、解析対象のフローを読み込むことになる。ただしここで、loopback7 インターフェースは再注入用のインターフェースであり、かつ、Proxy などトンネリングプロトコルを解析するための特殊なインターフェースである。default インターフェースは、Flow Classifier で判別できなかったフローが全て出力されるインターフェースである。

図 3 では、複数の HTTP プロトコル用インターフェース（http[0-3]）があることがわかる。これは、HTTP のフローをロードバランスし、複数の CPU 上で解析器を実行できるようにするためである。すなわち、HTTP のアナライザを 4 プロセス起動し、それぞれ異なる HTTP 用のインターフェースへ接続すれば、CPU リソースを有効に活用することができるようになる。

```
$ 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=
```

図 3　抽象フローインターフェースのディレクトリ構造

3 実装

図 4 は、SF-TAP Flow Abstractor の主要なクラス、関数、スレッドの関係を表したものである。本節では、この図に基づいて、SF-TAP Flow Abstractor の実装を解説する。

3.1 SF-TAP Flow Abstractor の主要クラス

SF-TAP Flow Abstractor は C++ で実装されており、基本的にオブジェクト指向的な考えに基づいて実装されており、関数や変数はクラスを用いてオブジェクト化されている。SF-TAP Flow Abstractor の主要なクラスは以下となる。

fabs_pcap　pcap を用いて Ethernet フレームをキャプチャするクラス。

fabs_netmap　netmp を用いて Ethernet フレームをキャプチャするクラス。

fabs_ether　キャプチャした Ethernet フレームを、適切な関数へ渡すクラス。

fabs_fragment　IPv4 の IP フラグメントパケットの再構成を行うクラス。

fabs_udp　UDP パケットに関する処理を行うクラス。

fabs_tcp　TCP パケットに関する処理を行うクラス。

fabs_appif　UNIX Domain Socket に関する処理を行うクラス。

fabs_id　フロー ID 管理を行うクラス。

fabs_id_dir　fabs_id クラスに、フローの方向に関する情報を持たせたクラス。

fabs_pcap と fabs_netmap は、Ethernet フレームのキャプチャを、pcap、あるいは netmap を用いて行

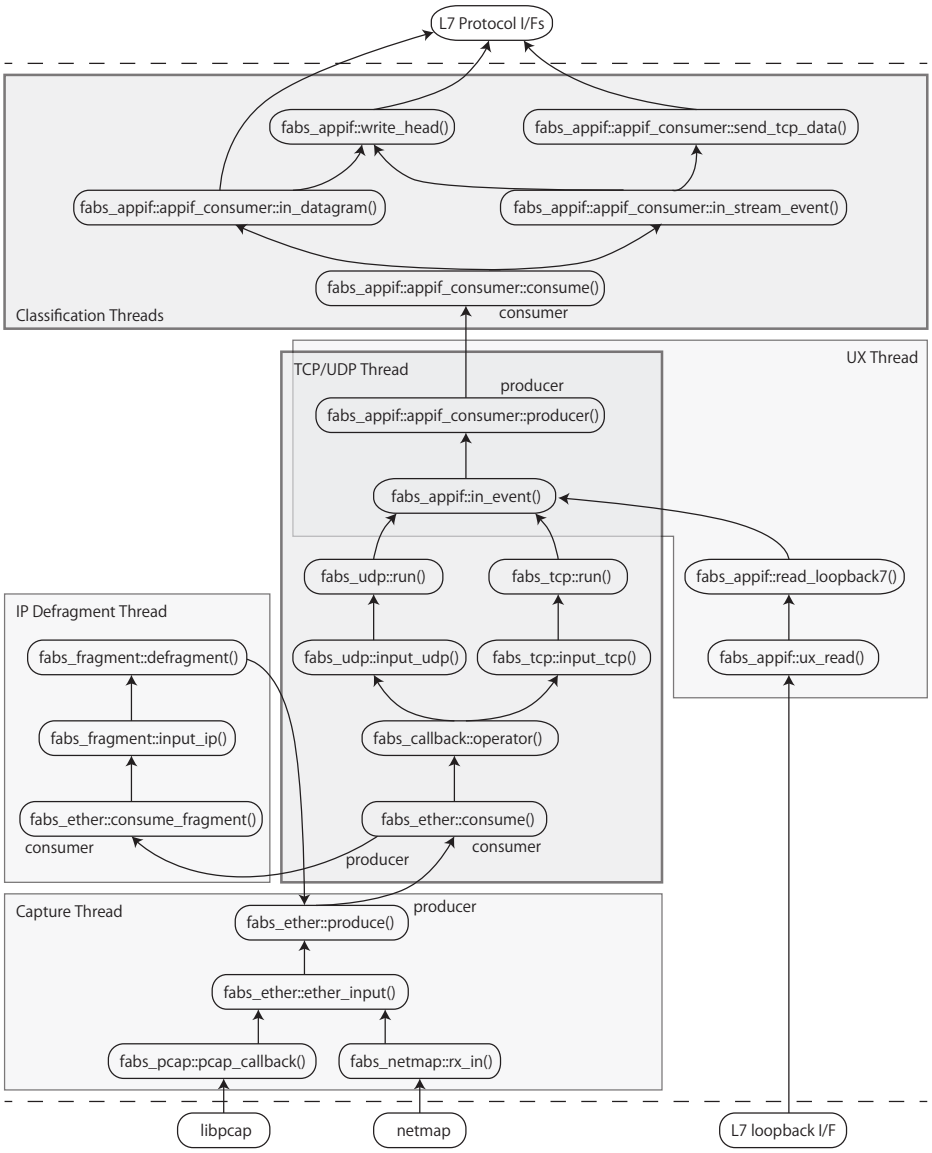

図 4　SF-TAP Flow Abstractor の実装

うクラスとなる。これらクラスがキャプチャした Ethernet フレームは、fabs_ether クラスの ether_input 関数へと渡される。fabs_ether クラスでは、Ethernet フレームから IP パケットを取り出した後、IPv4 のフラグメント化されたパケットかそうでないかを判別し、IPv4 のフラグメントであった場合は、fabs_fragment クラスの input_ip 関数へとパケットを転送する。IPv4 フラグメントパケットでなかった場合は、UDP か TCP パケットの場合のみ、fabs_callback クラスの operator () 関数を通して、fabs_udp か fabs_tcp クラスへとパケットが転送される。fabs_udp クラスは特に何も行わずに fabs_appif クラスへとパケットを転送を行うが、fabs_tcp クラスでは TCP フローの再構成を行った後、fabs_appif クラスへとパケットの転送を行う。

fabs_appif クラスは、受信したパケットのアプリケーションプロトコル判別、UNIX Domain Socket の管理（listen/accept/close）、UNIX Domain Socket へのデータ入出力を行うためのクラスである。アプリケーションプロトコルの判別は正規表現を利用して行い、正規表現ライブラリとして re2 [12] を利用しており、マッチング処理は、fabs_appif::appif_consumer クラスの in_datagram 関数か send_tcp_data 関数で行われる。アプリケーションプロトコルを判別した後、適切な UNIX Domain Socket へとデータを出力する。UNIX Domain Socket へは、ヘッダとパケットペイロードの順に出力を行うが、ヘッダの出力は fabs_appif クラスの write_head 関数にて行われる。UDP の場合は、in_datagram 関数がペイロードも出力し、TCP の場合は、send_tcp_data 関数がペイロード出力を行う。

fabs_id と fabs_id_dir は、それぞれ、フロー ID を管理するクラスである。fabs_id は一意にストリームを識別するために用いられ、fabs_id_dir は、fabs_id の情報に加え、ストリームの中での方向（アップストリームかダウンストリームか）という情報も保持する。SF-TAP Flow Abstractor では、これらフロー ID 管理クラスの情報をもとにしてフローの管理を行う。

3.2　SF-TAP Flow Abstractor のスレッド

次に、本節では、SF-TAP Flow Abstractor のスレッドについて解説を行う。SF-TAP Flow Abstractor は、基本的に producer-consumer パターンを用いた、スレッドの利用を行っている。マルチスレッドプログラミングで難しいのは、同期処理であり、同期処理を正しく実装しないと、パフォーマンスの劣化やマルチスレッド関連のバグ（デッドロックなど）に悩まされることになる。しかし、producer-consumer パターンを利用すれば、スレッド間で共有するデータを減らすことができ、同期処理を単純化することができる。

SF-TAP Flow Abstractor では、アトミック演算を利用した低レベルなスピンロックによる同期処理機構を独自に実装している。これは、pthread_mutex などは、関数呼び出し分だけオーバーヘッドがあるのと、OS のスケジューラにスレッドの処理が渡される可能性があるためである。OS のスケジューラへとスレッド処理が渡されると、コンテキストスイッチが発生し、スレッド間の同期に大きな遅延が発生する可能性が高く、これを避けるために独自に同期処理機構の実装を行った。fabs_spin_lock クラスと fabs_spin_rwlock クラスが、低レベルな同期処理を行うためのクラスである。fabs_spin_lock クラスでは、単純なスピンロックを実装しており、fabs_spin_rwlock クラスでは、readers-writer ロックを実装している。

SF-TAP Flow Abstractor では、図 4 で示す通り、Capture スレッド、IP Defragment スレッド、TCP/UDP スレッド、UX スレッド、Classification スレッドの 5 種類のスレッドが用いられる。Capture スレッドは、pcap か netmap を用いて Ethernet フレームをキャプチャするスレッド、IP Defragment スレッドは、IPv4 のフラグメントパケットを再構成するスレッド、TCP/UDP スレッドは、TCP と UDP に関する処理を行うスレッド、UX スレッドは、UNIX Domain Socket に関する処理（listen、accept、close など）を行うスレッド、Classification スレッドはアプリケーションプロトコルの判別及び、UNIX Domain Socket への出力を行うスレッドとなる。

4　応用事例

これまで、SF-TAP は、DNS オープンリゾルバの調査研究とサードパーティウェブトラッキングの調査研究で応用された。本節では、これら SF-TAP の応用事例について説明する。

4.1　DNS オープンリゾルバの実態調査研究

DNS アンプ攻撃 [13][14] は、2013 年ごろから流行した DDoS 攻撃手法のひとつであり、不特定多数からのリクエストにも応答する DNS サーバ（DNS オープンリゾルバ）を悪用しているのが大きな特徴である。この DNS オープンリゾルバは、インターネット上に多数存在しており、攻撃者はソース IP アドレスを攻撃先の IP アドレスへ詐称した DNS クエリを、DNS オープンリゾルバへ送信することで攻撃を行う。そこで我々は、DNS アンプ攻撃の原因となる DNS オープンリゾルバについて広域調査を行った [15]–[17]。図 5

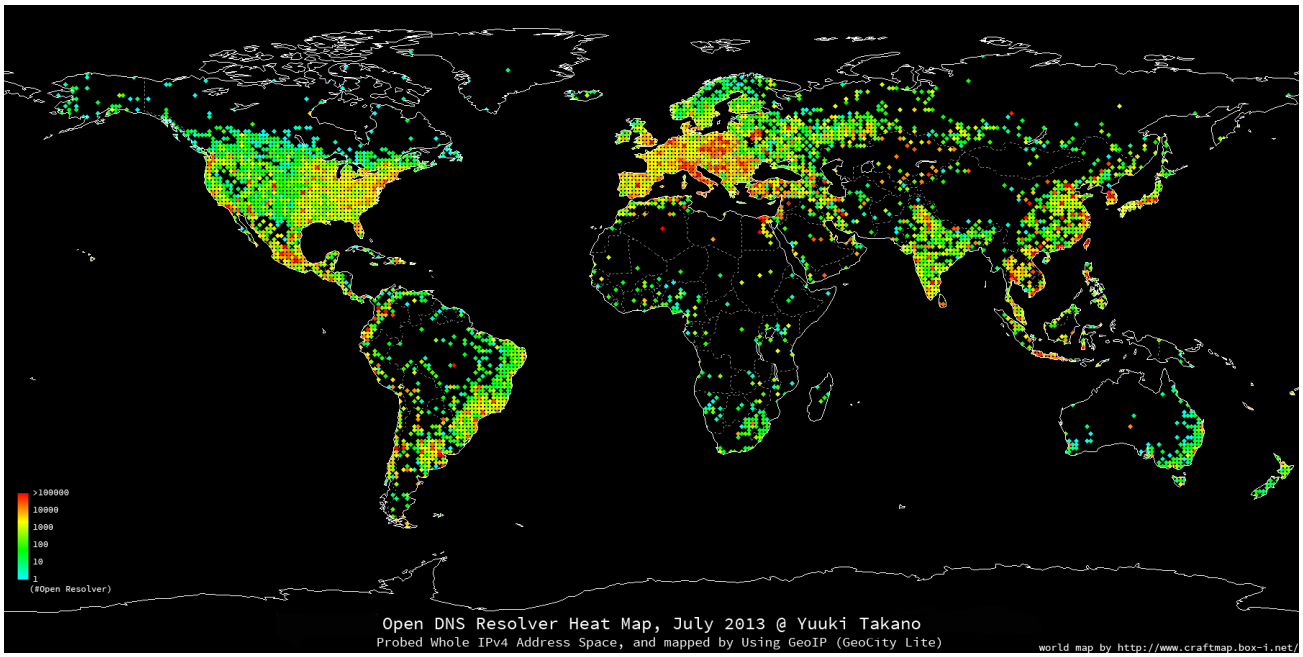

図 5　DNS オープンリゾルバの世界分布 (2013 年 7 月)

は、DNS オープンリゾルバの分布状況を世界地図上に表したものである。この図からもわかるように、我々の調査の結果、DNS オープンリゾルバは世界中に多数分布していることが明らかとなった。

本調査では、インターネット上にある DNS オープンリゾルバを探索する DNS Prober と、DNS Prober から得られた結果を逆引きする Reverse Lookupper、統計情報を得る Statistical Analyzer の 3 つのコンポーネントからなるシステムを利用して、DNS オープンリゾルバの調査を行った。DNS Prober と Reverse Lookupper では、DNS パケットの解析を行うが、この解析部分に SF-TAP のプロトタイプ版ソフトウェアを用いた。このように、トラフィック解析用の機構を利用することで、DNS サーバなどの計測、解析も容易に行えることができるようになる。

本研究成果は、いくつかの論文にて引用されているが [18]–[20]、特に、インターネット計測分野のトップカンファレンスである ACM IMC 2015 にて発表された Kührer らの論文 [18] では、我々が行った調査手法に加え、DNS オープンリゾルバのデバイスフィンガープリントなどを含む更なる詳細な調査を行っているため、興味があれば参考いただきたい。

4.2　サードパーティウェブトラッキング調査研究

広告サイトや、ソーシャルネットワーキングサイトなどは、ターゲット広告やトレンド予測などを行うためにユーザのウェブ閲覧履歴を秘密裏に取得しており、このような、サードパーティウェブトラッキングは、深刻なプライバシ問題として議論されている [21][22]。しかしながら、その性質上から、多くのユーザはプライバシ問題について気づくことができないまま、プライバシが侵害されている。特に、ソーシャルネットワーキングサイトは、実名で利用しているユーザも多く、実名とウェブ閲覧履歴が紐付いてしまっているという問題がある。そこで我々は、ウェブ広告におけるインフォームドコンセントを実現すべく、研究開発を行っている。インフォームドコンセントとは、医療分野で利用される言葉であるが、ここでは、どのような個人情報が広告サイト等によって取得されているかを正しくユーザが把握し合意したうえでウェブを利用する、という事を示す。

MindYourPrivacy [23] は、我々が研究開発を行ったサードパーティウェブトラッキングを可視化するシステムである。MindYourPrivacy では、SF-TAP のプロトタイプ実装を用いて HTTP トラフィックを解析した後、いったん、MongoDB [24] へデータを保存し、最終的に、その保存したデータを解析してユーザへサードパーティウェブトラッキングに関する情報を見せる。本システムは、WIDE 合宿というネットワーク関連の研究会にて、デモ・実験を行い、その結果を表したのが、図 6 となる。

本研究では、グラフのクラスタリングを行い、サードパーティウェブトラッキングを行っているサイトを抽出する手法の提案も行っており、図 6 の下部が、MCODE [25] を用いてクラスタリングを行った結果である。MindYourPrivacy では、ネットワークトラフィックの解析結果を、いったん、MongoDB へ保存して、後でバッチ処理でデータ解析を行っており、

図 6 のようなグラフの可視化などは、Cytoscape [26] を用いて手作業で行っていた。これを、オンライン処理でグラフの可視化までを自動で行うようにしたのが、ビッグデータ解析を行っているビッグデータプレイヤを可視化するシステム、ビッグデータビジュアライゼーションシステム CHAKRA であり、図 7 は、CHAKRA を用いてグラフの可視化を行ったものである。CHAKRA では、ネットワークトラフィックの解析に SF-TAP を利用しており、グラフ描画のアルゴリズムに、バネモデルをリーマン多様体上へ適用する描画手法 [27] を用いている。SF-TAP 及び CHAKRA は、ネットワーク分野における日本最大のビジネスショーである Interop Tokyo 2015 にてデモを行い、その結果、サイエンス部門でグランプリ賞を、ShowNet デモンストレーション部門で特別賞を、それぞれ受賞した。

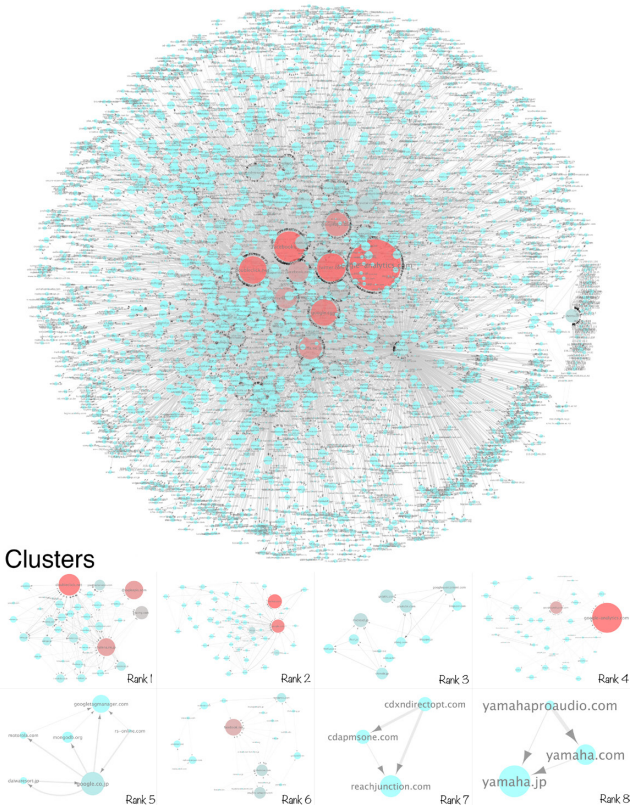

図 6　WIDE 合宿 (2013 年 9 月) のデータ

5　関連研究

tcpdump [3]、WireShark [28] は伝統的なパケットキャプチャ・解析ソフトウェアであり、現在でも幅広く利用されている。libnids [29] は、パケットキャプチャのみではなく、TCP フロー再構成も行うネットワークトラフィック解析用のライブラリである。これらは、基本的にシングルスレッドで動作するため、広帯域トラフィックの解析には不向きである。

フローレベルで広帯域トラフィックの解析を行うためのものとして、SCAP [30]、GASPP [31] が提案されており、コモディティハードウェアを利用した 10 Gbps トラフィックのトラフィックのフローレベルでの解析を実現している。SCAP は Linux カーネル内で動作し、NIC の送受信キューに対してスレッドを割り当てることで高速化を行っている。また、

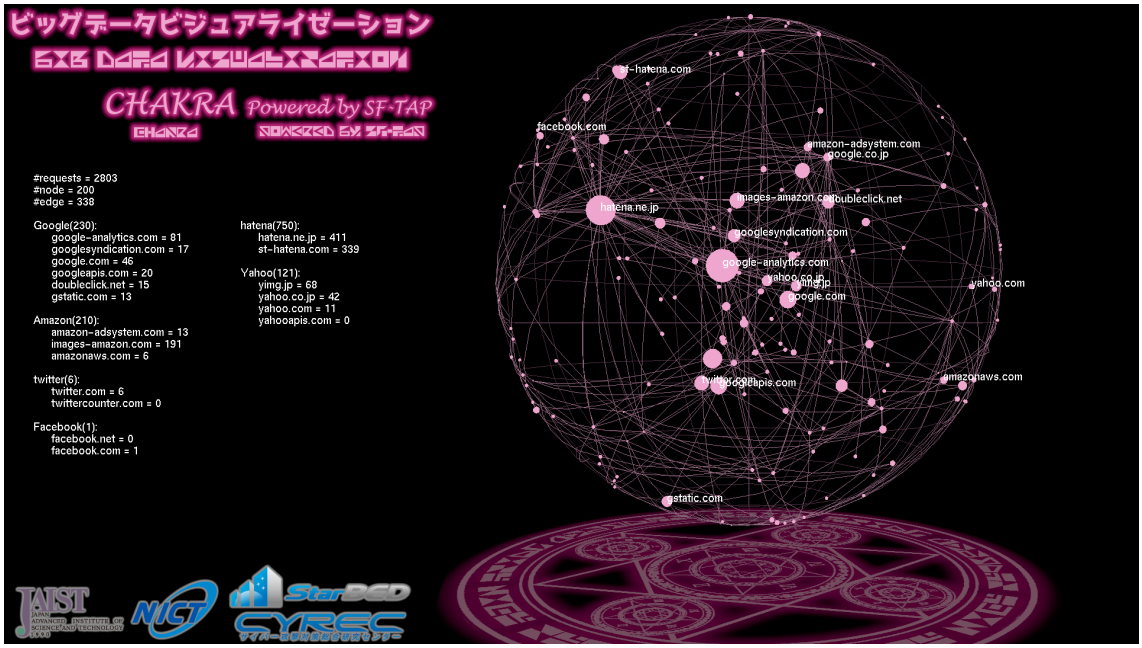

図 7　CHAKRA: ビッグデータビジュアライゼーションシステム

Subzero-Copy Packet Transfer と呼ばれる機構を備えており、必要なトラフィックのみ選択的に解析することが可能となっている。GASPP は GPGPU を利用したフローレベルの解析エンジンであり、netmap [6] を利用して、NIC と GPU 間での高速メモリ間転送を実現している。フローレベル解析の高速化という点では我々の提案方式と似ているが、我々の提案では、フロー解析部分のスケーラビリティにも考慮を行っており、さらに、共通インターフェースによるモジュラリティを備えているのが大きな違いである。

高速なパケットキャプチャフレームワークとしては、netmap、DPDK [32]、PF_RING [33] が提案されている。従来手法では、NIC、カーネル、ユーザの間で多くのメモリコピーと割り込みが発生したため、広帯域トラフィックのキャプチャは難しかった。これら提案では、メモリコピー回数と割込みの回数を劇的に減らすことで、10 Gbps ワイヤレートでのパケットキャプチャを可能にしている。我々の実装では、ネットワークトラフィックキャプチャ部分に netmap を採用している。

アプリケーションレベルでのトラフィック分類を行うものとしては、nDPI [34]、libprotoident [35]、l7-filter [36]、PEAFLOW [37] などが提案されている。これらは、Aho-Corasik [38]、あるいは正規表現を用いてパターンマッチを行い、アプリケーションレベルでのプロトコル判別を行う。PEAFLOW では FastFlow と呼ばれる並列プログラミング言語を利用し、高速なトラフィック分類を行っている。

Snort [39]、bro [40]、Suricata [41] などの IDS ソフトウェアでは、フロー再構成と、アプリケーションレベルでの解析を行っている。bro は、プロトコルパーサ用言語の binpac [42] を利用して、アプリケーションレベルでの解析を行うことができる。しかし、Snort、bro はシングルスレッドで動作するため、広帯域トラフィックの解析には不向きである。Suricata はマルチスレッドで動作するため、より広帯域なトラフィックに対応可能である。これらは、フロー再構成部分と、アプリケーションレベルでの解析部分が密に結合しているため、柔軟にロジックを付け替えることはできず、これらソフトウェアが提供するルールの記述方法やドメイン固有言語に縛られてしまう。

Schneider [43] らは、10 Gbps トラフィックをフロー単位で分割し、スケールアウトさせるアーキテクチャの提案を行っている。しかしながら、実際に彼らが行ったのは 1 Gbps での検証のみであり、10 Gbps の場合は概念の提案のみとなっている。SF-TAP では、フロー単位で 10 Gbps トラフィック分割を行うソフトウェアを実装し、実証まで行った。

Click [44]、SwitchBlade [45]、ServerSwitch [46] では、ネットワークスイッチのモジュラ化を行っており、非常に柔軟に、かつプログラマブルにネットワークにおける機能を実装可能にしている。SF-TAP では、これらの思想を元にし、解析ロジックのモジュラ化を行い、プログラマブルな解析ロジックの実装を可能にしている。

BPF [1] は、非常によく知られたパケットキャプチャのための機構であり、ネットワークトラフィックを UNIX の /dev のような方法で抽象化を行っている。SF-TAP は、BPF の考えを拡張し、フローレベルでトラフィックを抽象化している。これにより、解析ロジックのモジュラ化とコアスケールを可能にしている。

6　おわりに

本稿では、柔軟で規模追従可能なネットワークトラフィック解析基盤である SF-TAP の解説を行った。libpcap など、従来のネットワークトラフィックキャプチャ機構では、広帯域なネットワークに対して、機械学習など高度な解析機構を適用することが難しかったが、SF-TAP を利用すると、アプリケーションレベルでのネットワークトラフィック解析を容易に行うことができるようになる。これは、SF-TAP が、モジュラリティとスケーラビリティを備えた設計となっているためである。

SF-TAP は、SF-TAP Cell Incubator と SF-TAP Flow Abstractor という 2 つのメインコンポーネントから成り立つ。SF-TAP Cell Incubator は、複数台マシンを用いてネットワークトラフィック解析を行うための機構であり、本コンポーネントを用いてスケーラビリティを実現してる。SF-TAP Flow Abstractor は、UNIX の /dev、BPF、Plan 9 などで利用されている、ファイルを用いた抽象化手法を適用し、フローの抽象化を行っている。この抽象化によって、モジュラリティ及び複数 CPU コアの有効活用が行えるようになってる。

本稿では、SF-TAP Flow Abstractor の実装についても解説した。現状、SF-TAP Flow Abstractor は、セッション層のプロトコルに TCP と UDP のみ対応していたが、QUIC などのプロトコル対応も行うためには、本稿で行った実装の解説が参考になるだろう。また、データリンクフレームのキャプチャ機構として、現在、libpcap と netmap のみに対応しているが、DPDK などの機構にも対応したい場合にも参考となるだろう。

さらに、本稿では、SF-TAP を用いた応用事例についても解説した。これまで SF-TAP は、DNS オープ

ンリゾルバの調査研究や、サードパーティウェブトラッキングの調査研究に活用されてきている。これら事例からも明らかなように、ネットワークトラフィック解析技術は、ネットワークセキュリティの基礎となる技術であり、IDS などのアルゴリズム研究・開発に極めて重要であるといえる。今後は、SF-TAP を利用し、IDS、ネットワークトラフィックエンジニアリング、ネットワークフォレンジックなどといった、実社会に役立つ研究開発を行っていきたい。

【参考文献】

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 Installation System Administration Conference (LISA15), pp.25–36, Washington, D.C., Nov. 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.

高野祐輝　（たかの　ゆうき）

サイバーセキュリティ研究所
サイバーセキュリティ研究室
研究員
博士（情報科学）
コンピュータアーキテクチャ、ネットワークシステム、ネットワークセキュリティ