RL-001

ソフトウェアによるリアルタイムなネットワークトラフィックフロー監視手法
Software-based Real-time Network Traffic Flow Monitoring

坂本 裕紀† 中村 遼† 江崎 浩†
Hiroki Sakamoto Ryo Nakamura Hiroshi Esaki

概要

ネットワーク運用管理においてトラフィックをリアルタイムに監視分析する技術が必要とされている．現在高速なネットワークのリアルタイムトラフィック分析にはパケットをサンプリングして情報を取得する手法が利用されている．しかしこの手法では監視すべきパケットを見逃す可能性があり，またトラフィックの一部しか把握できず正確性に劣る．近年，ソフトウェアでの高速パケット I/O 処理を可能にする技術が開発され，ソフトウェアで高速なトラフィック分析を実現できるようになった．本研究では，ソフトウェア実装によって高速なネットワークでもトラフィックをサンプリングせずリアルタイムにフローを監視する手法を提案する．

1 はじめに

ネットワークトラフィック分析は，ネットワークの品質と信頼性を保つための運用管理において重要な手段のひとつである．ネットワークの運用管理者は，ネットワークトラフィック分析によって対象のネットワークについて利用状況を把握したり異常なトラフィックや障害の検知を行う．また，何か問題が起こったときには迅速な対応が求められるためネットワークトラフィックの監視や分析はリアルタイムにできなくてはならない．たとえば，内部のネットワーク機器が何者かに乗っ取られ不正な通信を行っていたとする．ここでリアルタイムにトラフィックを監視できれば，不正な通信を検知してすぐに通信の遮断や回線の切り離しといった対応を取ることができる．逆に，リアルタイムにトラフィックを監視できなければ，次に確認できるときまでその不正な通信には気付けず放置したままとなり，それがさらなる問題を引き起こす可能性もある．このようにリアルタイムにネットワークトラフィックを監視分析できることは運用管理の面で重要である．

ネットワークトラフィック分析のひとつの手法として，TCP や UDP のセッションを取得するフローの計測分析は必要な情報が少ないため軽量かつ高速なトラフィック分析が可能である．フローは「5 タプル」と呼ばれる 5 つの情報の組 (送信元 IP アドレス，宛先 IP アドレス，送信元ポート番号，宛先ポート番号，プロトコル) によって識別される．5 タプルを取得すればどの二者が何のサービスを利用して通信しているのかについての情報を得られ，これによりネットワークの利用状況の把握や異常なトラフィックの検知といったさまざまな分析が可能となる．また，5 タプルはパケットのヘッダのみ見れば取得できるため，軽量かつ高速なトラフィック分析が可能であることからリアルタイムに分析を行うのに適している．

ネットワーク機器のスループット向上にともないネットワークは高速化し，トラフィックは広帯域化している．このトラフィックのすべてを従来のネットワーク機器でリアルタイムに分析することは，プロセッサやメモリなどのネットワーク機器のハードウェア資源の制約のため困難である．そこで，sFlow [1, 2] のように分析対象のトラフィックの一部を抽出して取得することで負荷を減らしリアルタイム分析を可能にするパケットサンプリング技術が用いられるようになった．高速なネットワークでリアルタイムに分析を行う場合，処理速度の点でサンプリングの手法は有効である．しかし，サンプリングするということは意図的に一部のトラフィックを見逃すということである．昨今のネットワークを介した攻撃の中には，たとえば Shellshock を利用した攻撃 [3] のようにパケット数の少ない攻撃も存在する．大量のトラフィックを流し込む Distributed Denial of Service (DDoS) 攻撃とは異なり，これらの攻撃に相当するフローはサンプリングでは見逃してしまう可能性が高い．また，運用管理者がネットワークの利用状況を見る目的においても正確な把握ができない．ポートの統計情報から全体のトラフィック量を把握できても，個別の通信についての情報はサンプリングで取得したものに限られる．このようにサンプリングを用いた手法では監視すべきトラフィックを見逃し正確性に劣るといった問題がある．

近年，ソフトウェアでの高速パケット I/O 処理を可能にするライブラリやフレームワーク [4, 5, 6] が開発され，こうした技術を用いることでソフトウェアによる高速なトラフィックの取得や分析 [7, 8, 9] が可能となった．そこで本研究では，この高速パケット I/O 処理技術をパケットキャプチャに用いることで高速なネットワークでもトラフィックをサンプリングすることなくリアルタイムに監視できる手法を提案する．提案手法では，ポートミラーリングやネットワークタップなどによって取得したトラフィックを高速パケット処理ライブラリを用いて処理することで，トラフィックのすべてのパケットからサンプリングすることなく高速にフローの情報を取得する．また，取得したフローの情報を WebSocket [10] を用いて外部の web ブラウザへ秒単位で送信することでリアルタイムなフローの監視を実現する．

† 東京大学大学院情報理工学系研究科

2　関連技術・研究

ネットワークトラフィックのフローを計測分析する技術として NetFlow/IPFIX [11, 12, 13] と sFlow [1, 2] が現在広く利用されている．これらはルータやスイッチ上で動作し，収集したトラフィックの情報を外部のコレクタへ集約する．NetFlow/IPFIX はルータやスイッチ上でフローの集計まで行うため，特に高速なネットワークではプロセッサに大きな負荷がかかる．また，フローを集計するテーブルはハードウェア上に実装されるため大きさに限りがある．sFlow は集計するパケットをサンプリングして取得することで高速なネットワークトラフィックを監視分析する技術である．分析は外部のコレクタで行うため NetFlow/IPFIX のような制約はない．しかし，パケットをサンプリングして取得するためすべてのトラフィックを見ることはできない．これではパケット数の少ない攻撃を見逃してしまう可能性が高く，ネットワークの利用状況の把握も部分的にしか行えない．

このように現在利用されているフロー計測分析技術にはハードウェア資源の制約の問題とパケットサンプリングによる問題という 2 つの課題がある．このため，高速なネットワークにおけるトラフィックのすべてのパケットを対象とするリアルタイムなフローの計測分析は現在達成できていない．

近年，Intel Data Plane Development Kit (DPDK) [4] や netmap [5]，PacketShader [6] といったソフトウェアでの高速パケット I/O 処理を可能にするライブラリやフレームワークが開発された．これらの技術は，複雑でオーバーヘッドの大きいカーネルでの処理を経由せずにユーザ領域でアプリケーションが高速に直接パケットを処理することを可能にした．こうした高速パケット処理ライブラリを用いることでソフトウェアによる高速なトラフィックの取得や分析 [7, 8, 9] が可能となった．そこで本研究では，高速パケット処理ライブラリを用いたソフトウェア実装によって，ハードウェアの制約に縛られずトラフィックをサンプリングせずにフローを監視できる手法を実現する．

3　提案手法

3.1　要件

トラフィックをサンプリングせずにリアルタイムなフロー監視を実現するという本研究の目的から，提案手法の要件は次の 3 つである．

1. サンプリングせずすべてのパケットを取得
2. パケットをフローごとに集計
3. フロー集計情報のリアルタイムな監視

まず，パケットをサンプリングして取得する手法では監視すべきトラフィックを見逃してしまい正確性に劣るという問題があった．そのためサンプリングすることなくトラフィックのすべてのパケットを取得する必要がある．また，トラフィックのフローの情報があれば利用状況の把握や不正な通信の検知といった分析も可能であり，しかも軽量かつ高速に分析することができる．したがって，フローの監視を行うために取得したパケットの統計情報をフローごとに集計する必要がある．さらに，運用管理者が発生した問題に迅速に対応するためには計測分析をリアルタイムにできなくてはならない．そのため集計したフロー情報をリアルタイムに監視できるような機構が必要である．

3.2　想定環境

本手法は図 1 に示すように監視対象のトラフィックを光タップやポートミラーリングによって複製して取得することを想定する．このとき，たとえば企業や大学のネットワークの上流とのリンクにおけるトラフィックを監視すると考えると，1 リンク 10 Gbps 規模のネットワークのトラフィックを処理する必要がある．また，バックボーンネットワークのトラフィック規模の 1M フローエントリ[1]まで集計できる必要がある．

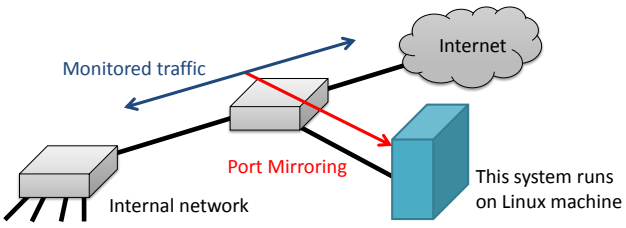

図 1　想定環境

3.3　手法の概要

本節では，3.1 節で示した 3 つの要件に合わせて手法全体を 3 つの部分に分けてその概要を述べる．

はじめに，ポートミラーリングなどによって取得してきたトラフィックについて，すべてのパケットヘッダから 5 タプルのみを高速に取得する．この部分には高速パケット処理ライブラリを用いることで，パケットをサンプリングすることなくトラフィックの高速な取得を実現する．トラフィックのフロー監視を行うため，パケットヘッダからはフローの識別に必要な 5 タプルの情報を取得すれば十分である．また，フローテーブルに必要な情報として，5 タプルのほかに時刻情報やパケット長などの統計情報をここで取得する．

次に，取得したフローの情報をフローテーブルによって集計管理する．テーブルの高速な参照と更新を可能にするため，5 タプルをキーとするハッシュテーブルとする．テーブル中の各キーに対するエントリには 5 タプルのほか，フローのパケット数，フローのバイト数，フローの開始時刻情報，フロー

1 WIDE MAWI Working Group の計測データ [14] より

の最新パケットの到着時刻情報といった基本的な統計情報をもたせる．パケットが到着するたびに取得した情報をもとにフローテーブルを更新する．

最後に，テーブルに集計したフローの情報をリアルタイムに監視できるよう秒単位でエクスポートする．このためにエクスポート用のテーブルをフローテーブルとは別に確保しておき，毎秒フローテーブルにアクセスしてエクスポート用の情報をテーブルに取得してくる．ここで別にテーブルを確保するのは，時間がかかるエクスポートの間にフローテーブルが随時更新されてしまうのでスナップショットとして情報を保管しておくためである．また，フロー情報のエクスポートにはリアルタイムな双方向通信を可能にする WebSocket を用いる．

3.4 設計

提案する手法のシステムの設計の概観を図 2 に示す．まず NIC で受信したトラフィックを Receive-Side Scaling (RSS) によって 3 つの CPU コアに分散して振り分ける．RSS は，マルチキューに対応した NIC において受信トラフィックを複数のキューに振り分け各キューに CPU コアを対応付けることで，複数の CPU コアでトラフィックを分散処理する仕組みである．ここで，振り分けられたトラフィックを処理するプロセスを RXFL と呼ぶ．RXFL ではパケットを取得してそこからフロー情報を取り出し，自身のもつフローテーブルの情報を更新する．また，フロー情報のエクスポートを担うプロセスを MONX と呼ぶ．MONX はエクスポート用のテーブルをもち，各 RXFL のフローテーブルを毎秒参照してこれを更新する．さらに MONX はエクスポートテーブルに保存したフローの情報を WebSocket によってエクスポートする．以下では RXFL と MONX の設計の詳細についてそれぞれ述べる．また，MONX がエクスポートした情報を受け取ってフロー情報の可視化を行う監視用クライアントの設計についても述べる．

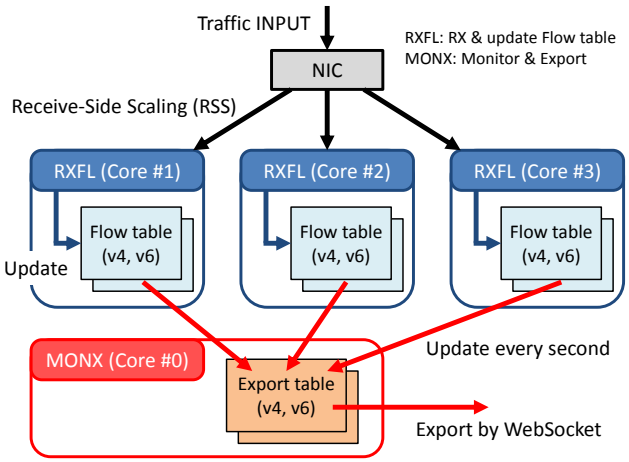

図 2　システムの設計の概観

3.4.1 RXFL: パケットの取得とフローテーブルの更新

各 RXFL は，パケットを取得してフローの情報を取り出し，フローテーブルを更新するまでの処理を行う．まず高速パケット処理ライブラリを用いてパケットを高速に取得し，ヘッダから 5 タプルを取得する．このとき，フローテーブルの更新に必要な情報として時刻情報とパケット長も同時に取得する．最後に，取得したフローの情報をもとに RXFL 自身のもつフローテーブルを更新する．受信トラフィックは複数コアに分散して RXFL で処理されるが，RSS によって NIC でトラフィックを振り分けるときにパケットヘッダ内の IP アドレスを振り分けのキーに用いることで，同じフローに属するパケットを必ず同じコアに振り分ける．これにより 1 つのフローは 1 つのフローテーブルのみで管理されることが保証される．また，フローテーブルを共有するのではなく各 RXFL が専用のフローテーブルをもつことにより，各 CPU のキャッシュを有効に使えるようにする．

RXFL がフローの集計管理に用いるフローテーブルは，テーブルの高速な更新を可能にするため，5 タプルをキーとするハッシュテーブルとする．3.2 節で述べたように 1 M フローエントリまで集計できる必要があることから，ここでは 1 M エントリを格納できるサイズを確保する．また，IPv4 と IPv6 とでは IP アドレスのサイズが異なり，つまり 5 タプルの大きさが異なるため，コアごとに IPv4 用のテーブルと IPv6 用のテーブルをそれぞれ用意する．テーブル中の各エントリにはキーである 5 タプルのほか，フローのパケットカウント，フローのバイトカウント，フローの開始時刻情報，フローの最新パケットの到着時刻情報といったフローの統計情報をもたせる．このフローテーブルの更新は次のようにして行う．まず取得した 5 タプルからハッシュ値を計算してインデックスを取得し，同一のフローエントリがあるかを確認する．同一のエントリがあればそのエントリの統計情報を更新し，なければ取得した情報をもとにエントリを新しく追加する．また，フローの期限切れの処理については後述する MONX で行う．フローテーブルについては以上のような設計とする．

3.4.2 MONX: フローテーブルの参照とエクスポート

MONX は，各 RXFL のフローテーブルを参照して取得したエクスポート用のフロー情報を WebSocket を用いて秒単位で送信する処理を行う．まず MONX はエクスポートテーブルをもち，各 RXFL のフローテーブルを毎秒参照してエクスポートテーブルに必要な情報を書き出す．フローテーブルと同様に IPv4 用のテーブルと IPv6 用のテーブルとをそれぞれ用意する．エクスポートテーブルのエントリには，5 タプルに加えそのフローの packets per second (pps) と bits per second (bps) を保存する．このために 1 秒前のフローテーブルのパケットカウントとバイトカウントの情報が必要になるのでそれらはエクスポートテーブルのエントリに管理用として保存する．エクスポートテーブルに書き出したフローの情報は，WebSocket を用いたサーバープッシュによって監視用

クライアントへまとめて送信する．

また，フローテーブルを参照するときに各フローの期限切れの確認処理を同時に行う．存在しなくなったフローの情報をいつまでもフローテーブルのエントリに残しておく必要性はない．むしろハッシュの衝突を避けるために，一定時間トラフィックがないフローは期限切れとして，フローテーブルのエントリから削除する．一定時間フローのトラフィックがないことを確認する手段として，フローテーブルのエントリに保存しているフローの最新パケットの到着時刻情報を用いる．フローテーブルの参照は毎秒行なうので，そのときに時刻情報を比較することによって期限切れの確認を行う．本システムでは期限切れまでの時間を 30 秒と設定する．

クライアントへのエクスポート時の通信量を削減するため，WebSocket でのエクスポートにはバイナリ形式を用いる．エクスポートするデータはヘッダ部分とレコード部分によって構成し，ヘッダ部分にはエクスポートごとのデータを識別するための情報や全体の pps や bps といった情報をもたせる．レコード部分にはフローごとに 5 タプルと pps や bps といった集計情報をもたせる．

3.4.3 監視用クライアント

提案システムにおける監視用クライアントは web ブラウザ上で動作する．MONX がエクスポートする情報を受信してそこからフロー情報を取り出し，取り出したフロー情報の可視化を行う．まずクライアントはシステムに接続し，WebSocket で毎秒エクスポートされるメッセージを受信する．受信のたびにバイナリ形式のメッセージをデコードしてヘッダ部分とレコード部分の情報をそれぞれ取得する．MONX が毎秒エクスポートするフローの情報はエクスポート時の情報のみであるため，クライアント側で取得した情報をデータセットとしてフローごとに一定数保存する．また，データセットとして保存したフローの pps と bps の変化をグラフでそれぞれ描画することで可視化を行う．

4 実装

前章で述べた提案システムは，監視用クライアントを除き C 言語により実装を行なった．まず高速にパケットを取得してフローの情報を取り出す処理を行う部分の実装には，高速パケット処理ライブラリとして Intel DPDK [4] を用いた．取得した 5 タプルとパケット長，時刻情報を構造体に格納して渡すことでフローテーブルの更新を行う実装とした．また，CPU クロックを基準とするタイムスタンプカウンタを用いることで高速に時刻情報を取得した．フローテーブルにおいてキーからハッシュ値を計算しインデックスを取得する部分の実装には，DPDK に含まれるハッシュテーブルのライブラリを用いた．ハッシュ関数はライブラリに付属の Jenkins ハッシュを用いた．このライブラリの仕様により，異なるキーの同一ハッシュ値への衝突については一定数までは処理をしてインデックスを返すが，その許容数を超えて衝突が起きた場合後から来たパケットのフローはインデックスを取得できず，つまりフローテーブルに格納できず捨てられる．

MONX でのエクスポートに用いる WebSocket サーバの実装部分には libwebsockets [15] というライブラリを用いた．このライブラリには簡易的な HTTP サーバ機能があるため，監視用クライアントの本体となる HTML と JavaScript のファイルを HTTP 経由でアクセスしてきた web ブラウザに渡すよう実装した．また，フロー数が増えるとエクスポートするデータのサイズも大きくなるが，メッセージを複数のデータフレームに分割送信できる WebSocket の仕様を利用できるよう実装した．

監視用クライアントでデータを処理する本体は JavaScript により実装を行なった．バイナリ形式のメッセージのデコードには DataView を用いた．また，フローの可視化にはグラフ描画ライブラリとして Chart.js [16] を用いた．ここでは，すべてのフローではなく選択したフローのみを描画できるようにグラフ描画に用いるデータセットを配列操作によって動的に操作できる実装を行なった．

5 評価

提案手法が 3.1 節で示した 3 つの要件を満たすかについて評価する．まず，高速なネットワークでもサンプリングせずにすべてのパケットを取得しフローごとに集計できることを検証するため性能評価実験を行なった．実験では，テストトラフィックを構成するフロー数を変化させながら実装した提案システムで受信し，どの程度のフロー数までパケットを落とさずに処理できるかを計測した．評価実験を行なったマシンの構成を表 1 に示す．この環境のマシンを 2 台用意し，それぞれの 10 G ポートを SFP+ ケーブルで接続した．1 台でテストトラフィックを生成し，もう 1 台で提案手法のシステムを動かしてテストトラフィックを受信した．テストトラフィックは 64-byte の Ethernet/IPv4/UDP フレームを 10 Gbps ワイヤーレートで 90 秒間送信した．このトラフィックは 14.88 Mpps に相当する．また，送信元 IP アドレスを変えることで異なるフローを生成した．このテストトラフィックの生成には MoonGen [17] を用いた．

表 1　評価実験を行なったマシンの構成

CPU	Intel Core i7-4770K 3.50 GHz (4 コア，HT: off)
Memory	32 GB DDR3 1333 MHz
NIC	10 Gbps Intel X520-DA2 (Intel 82599ES chipset)
OS	Ubuntu 14.04.3 LTS 64-bit with Linux kernel 3.16.0

性能評価実験の結果を図 3 に示す．16000 フローまではほぼパケットを落とすことなく 14.88 Mpps のトラフィックを取得し処理できた．それ以上になると，フロー数が増えるほど少しずつ性能は低下した．それでも 1 M フローで 14.64 Mpps，つまり 98.4% のトラフィックを取得できており，十分高速なネットワークでも処理できることが示された．また，グラフにはあらわれないが，1 M フローのテストトラフィッ

クを送信したときに 10 フローがフローテーブルに格納できずに捨てられた．これは同一ハッシュへの衝突が許容数を超えたためであり，格納できるエントリ数を増やすなどしてハッシュの衝突確率を下げることで回避可能と考えられる．また，フロー数が増えるほど性能が低下したのはハッシュの衝突回数の増加が原因と考えられる．

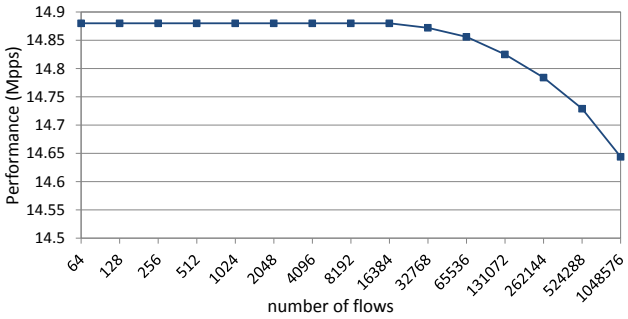

図 3　処理できたトラフィック量のフロー数に対する変化

次に，web ブラウザ上でフローをリアルタイムに監視できることを検証するため，実装した監視用クライアントをシステムに接続して実際に動作させた．動作中の web ブラウザ画面のスクリーンショットを図 4 に示す．このとき，web ブラウザ上で各フローの pps や bps の情報が毎秒更新されることを確認できた．また，グラフで可視化する数フローを選択し，それらのグラフが毎秒更新されることも確認できた．これにより，エクスポートした情報をもとにリアルタイムなフローの監視が可能であることを確認した．ただし，フロー数が増加するとエクスポートに用いる通信速度によってはエクスポートにかかる時間が 1 秒を超える可能性があり，その場合毎秒の更新ができなくなってしまう．したがって，フロー数の増加に対応できるようエクスポートに高速な通信を用いる必要がある．

図 4　web ブラウザ上でのフローのリアルタイムな監視

6　まとめと今後の課題

本研究では，10 Gbps 規模のネットワークにおいてもトラフィックをサンプリングせずリアルタイムにフローを監視する手法を実現した．提案手法では，ポートミラーリングなどによって取得したトラフィックを高速パケット処理ライブラリを用いて処理することで，トラフィックをサンプリングすることなく高速にフローの情報を取得した．さらに，取得したフローの集計情報を WebSocket を用いて秒単位でエクスポートすることにより，リアルタイムなフローの監視を実現した．

本研究ではトラフィックから取得したフローの集計情報を秒単位でエクスポートするところまでを実現した．しかし，監視用クライアントの処理能力やエクスポートに用いる通信の速度によって，実際に可視化してフローを監視する部分の性能は大きく変わる．また，フロー数が多い場合にすべてのフローの情報を表示させるのはグラフの描画に限らず現実的でない．そこで，いくつかのフローを集約してエクスポートするという方法が考えられる．フローの集約によってエクスポートの通信量を節約できクライアント側の処理量を削減できる一方，集約すると失われてしまう情報が存在する．したがって，フローの集約を行うとしたときに適切な集約の手法について検討すること，また，その集約をリアルタイムに行う実現可能性について検討することを今後の課題とする．

参考文献

[1] sFlow.org. `http://www.sflow.org/`.
[2] Sonia Panchen, Neil McKee, and Peter Phaal. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks. RFC 3176, September 2001.
[3] bash の脆弱性対策について (CVE-2014-6271 等). `https://www.ipa.go.jp/security/ciadr/vul/20140926-bash.html`.
[4] Intel DPDK: Data Plane Development Kit. `http://dpdk.org/`.
[5] Luigi Rizzo. netmap: a novel framework for fast packet I/O. In *Proceedings of the 2012 USENIX Annual Technical Conference*. USENIX Association, 2012.
[6] Sangjin Han, Keon Jang, KyoungSoo Park, and Sue Moon. PacketShader: A GPU-accelerated Software Router. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pp. 195–206. ACM, 2010.
[7] Pedro M. Santiago del Rio, Dario Rossi, Francesco Gringoli, Lorenzo Nava, Luca Salgarelli, and Javier Aracil. Wire-speed Statistical Classification of Network Traffic on Commodity Hardware. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, IMC '12, pp. 65–72. ACM, 2012.
[8] Luca Deri, Maurizio Martinelli, and Alfredo Cardigliano. Realtime High-Speed Network Traffic Monitoring Using ntopng. In *Proceedings of the 28th Large Installation System Administration Conference (LISA14)*, pp. 69–79. USENIX Association, 2014.
[9] Jihyung Lee, Sungryoul Lee, Junghee Lee, Yung Yi, and KyoungSoo Park. FloSIS: A Highly Scalable Network Flow Capture System for Fast Retrieval and Storage Efficiency. In *Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '15, pp. 445–457. USENIX Association, 2015.
[10] Alexey Melnikov and Ian Fette. The WebSocket Protocol.

RFC 6455, December 2011.
[11] Cisco IOS NetFlow. http://www.cisco.com/.
[12] Benoit Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954, October 2004.
[13] Benoit Claise and Brian Trammell. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC 7011, September 2013.
[14] MAWI Working Group Traffic Archive. http://mawi.wide.ad.jp/mawi/.
[15] libwebsockets. https://libwebsockets.org/.
[16] Chart.js. http://www.chartjs.org/.
[17] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. MoonGen: A Scriptable High-Speed Packet Generator. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*, IMC '15, pp. 275–287. ACM, 2015.