MULTISCALE ENERGY NETWORK TOMOGRAPHY AND SMARTNIC-ACCELERATED IN-BAND NETWORK TELEMETRY FOR NETWORK INTERNAL STATE MONITORING

A Dissertation

by

YIXIAO FENG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Duffield, Nicholas G.
Committee Members,	Narayanan, Krishna R
	Chamberland, Jean-Francois
	Berkolaiko, Gregory
Head of Department,	Datta, Aniruddha

August 2021

Major Subject: Electrical and Computer Engineering

ABSTRACT

Network internal performance statistics are crucial for the control, operation and management of computer networks. This research work explores multiscale wavelet energy tomography using Discrete Wavelet Transform (DWT) based on end-to-end measurements for characterizing the delay statistics of internal links. Additionally, an efficient and flexible monitoring platform using a recently developed concept of In-band Network Telemetry is carefully designed and proved to be accurate and cost-efficient for monitoring network internal nodes.

Much effort and ingenuity has been applied to develop tomographic methods to derive information concerning link-level performance statistics from relatively available end-to-end measurements. However, there has been recognition in recent years that network phenomena, including network attacks, may manifest with distinct spectral distribution present in time series of associated network measurements. For time series in general, multiscale analysis using DWT is a powerful method to extract detailed signal components across frequencies. This research showed how a tomographic analysis of the DWT of end-to-end measurements can be used to provide an unbiased estimates of the energy spectrum of the contributions to those measurements from the path intersection. It also illustrates application of the method to detect low-rate periodic attack.

In-band Network Telemetry (INT), on the other hand, provides granular monitoring of performance and load on network elements by collecting information in the data plane without requiring intervention from control plane. INT enables traffic sources to embed telemetry instructions in data packets, avoiding separate probing or infrequent management-based monitoring. INT sink nodes track and collect metrics by retrieving INT metadata instructions appended by different sources of INT information. However, tracking the INT state in packets arriving at the sink is both compute intensive (requiring complex operations on each packet), and challenging for the standard P4 match-action packet processing pipeline to maintain line-rate. This research provided an accelerated monitoring platform on monitoring INT packets using SmartNIC and also showed how to optimize the INT operations to achieve cost-efficient over networks.

DEDICATION

To my mother, *Mei F.*, who devotes all her energy and time to support my twenty-seven years journey to my doctoral degree with sorrow and happiness.

To my beloved families, who guide me through every difficulties of my life and will always be my strongest shield wherever I go.

To my mentors, who give me precious research opportunities to learn the state-of-the-art techniques that will benefit my long-term research work.

To my friends and my cat *Gretta*, who company with me all the time and balance my life.

ACKNOWLEDGMENTS

I would like to thank my PhD mentor, Dr. Duffield, who patiently guide me through every research project and encourage me to try every new research idea.

I wish to thank the committee members, Dr. Duffield, Dr. Narayanan, Dr. Chamberland and Dr. Berkolaiko to challenge my research work, which helps me produce a better research result.

I would like to acknowledge and thank Texas A&M University for having a great research community. Special thanks to those staff members in the university and engineering department for their support and patience.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

It is my honor to have a dissertation committee consisting of Dr. Duffield [advisor, ECEN], Dr. Narayanan [ECEN], Dr. Chamberland [ECEN] and Dr. Berkolaiko [MATH].

The research work of my dissertation on INT is collaborated with University of California, Riverside. The network tomography work is guided by my advisor Dr. Duffield.

Funding Sources

My graduate study was supported by graduate assistant work under my supervisor Nick Duffield.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Network monitoring aims to provide feedback on network devices or connections among network devices such as queue occupancy and available bandwidth. Those measurements are prior knowledge to detect network anomalies and prevent undesirable network operations that would potentially affect the normal traffic.

The commonly used approaches are SNMP[1], RMON[2], NetFlow[3], OpenFlow[4]. SNMP[1] has passive sensors to collect traffic statistics, while a network control and management (NC&M) system pull statistics from its network elements. NetFlow sampling traffic and report to a flow collector for further analysis. OpenFlow, on the other hand, lets the switch report the network state periodically to the controller. Non-router based approaches such as actively injecting probes into networks are also popular for network monitoring because of its flexibility, but probing traffic consumes available bandwidth. Those approaches for network monitoring are frequently used by network operators and are sufficient for earlier networks. However, increasing limitations over those approaches has been observed for more complex and faster networks.

The aforementioned approaches (*i.e.,* SNMP[1], RMON[2], NetFlow[3]), by collecting information at each router and answering relatively infrequent polling requests, could not provide accurately flow-based end-to-end measurements and consume available bandwidth to transmit data. OpenFlow goes beyond the power of network management tools and provide a comprehensive and centralized view of global network configurations even in dynamic networks. However, some issues such as data fetching latency and scalability and security still exists. Most importantly, it is not generally possible to directly access and measure each point connections of networks due to the growing size of networks [5]. Therefore, the network internal statistics need to be measured in a more promising approach.

The Network Tomography approach is one possible solution to reconstruct the network internal performance by analyzing the end-to-end measurements[6]. Generally, it uses the observed path correlations to infer the statistics on the common portion. Additionally, the state-of-art technique

named In-band Network Telemetry (INT), embedded internal measurements by programmable switches into the network traffic itself, which totally independent from the control plane, provides granular monitoring of performance and load on network elements.

A large number of works [7, 8, 9, 10, 11, 12] have addressed different aspects of processing and collection of INT packets. Here, we focus primarily on INT monitors. IntMon [13] implements an INT monitoring service on the Open Networking Operating System (ONOS). However, it achieves very low processing rates and high cpu utilization. IntCollector [14] also uses UDP encapsulation for INT packets and the monitor reports INT change events based on a predefined threshold. However the INT packet processing and event detection are implemented on the host CPU, splitting INT packet processing into a fast path (accelerated by an eXpress Data Path (XDP)) and a slow path for exporting and inserting INT events into a database. Because of the packet processing being done on the host CPU, performance is limited. The work in [15] is closest to ours. They implement the INT packet processing and INT event detection using the sNIC P4 pipeline. But, they only report simple threshold-crossing INT events to the stream processor (running on the host CPU) using the kernel bypass technique AF XDP. However, the use of P4 pipeline restricts the per-flow state information to simple registers and counters only, and does not give us the ability to maintain complex per flow state that are required by most server-based networking applications [16]. Also, additional miscellaneous functions such as timeouts *etc.* are not easily implementable using P4 [16]. By using callable C functions and P4, we design a highly efficient INT monitoring platform that not only supports notification of INT events, but also exports the basic INT telemetry report for every INT packet.

The research focus of this thesis is on providing an inference method on the network internal statistics based on the end-to-end measurements and also present an architecture, design and implementation of an optimized high-performance INT processing and INT event detection framework for network internal state monitoring.

The network tomography approach researched in this work provides inference on the individual link energy at different scales based on end-to-end measurements using wavelet decomposition.

Additionally, an unbiased estimation on the wavelet energy at multiple scales of the time series on the path interactions is introduced. Model based simulation and network simulations using NS3 are implemented to evaluate the proposed inference approach.

Furthermore, a more practical monitoring platform is designed based on the flexibility of INT accelerating by the smartNIC. The proposed monitoring platform further addressed the limitations exposed by similar monitoring approaches using INT. The evaluations of the monitoring platform is conducted on a server with real network traffic.

2. MULTISCALE ENERGY NERTWORK TOMOGRAPHY

2.1 Overview

2.1.1 Time Series Tomography on Graphs

In the canonical framework for network performance tomography we wish to recover a set of edge properties X through the linear relation $Y = AX$ that expresses measurable path properties Y where A is the routing matrix of paths over edges. We start by formalizing these relations from time-series tomography. Let $G = (V, E)$ denote a directed graph and equip each directed edge $e \in E$ with a time series $X_e = \{X_{e,t} : t \in T\}$ for some temporal index set $T = \{1, \ldots, |T|\} \subset \mathbb{N}$. Let $V_B \subset V$ denote a subset of vertices that we shall call the *boundary*. For each ordered pair (b, b') of distinct boundary vertices, let there be designated a particular directed path $\mathcal{P}_{b,b'}$ of contiguous directed edges in E that connects from b to b'. Let $\mathcal{P}$ denote the set of paths connecting each ordered pair of vertices in V_B, and for each path $\pi \in \mathcal{P}$ let $Y_\pi = \{Y_{e,t} : t \in T\}$ denote the time series of sums or aggregates

$$Y_{\pi,t} = \sum_{e \in E} A_{\pi,e} X_{e,t} \tag{2.1}$$

where $A_{\pi,e}$ is the incidence matrix of edges e over paths π, i.e., $A_{\pi,e} = 1$ if edge e occurs in path π and zero otherwise. Network tomography seeks to infer properties of the $\{X_e\}$ from the properties of the path variables $\{Y_\pi\}$. However, the linear system (2.1) is in general underconstrained and so does not admit a unique solution.

In this paper we shall be concerned with the problem of how to infer multiscale temporal properties of the edge time series $\{X_e\}$ from measurements of the path time series $\{Y_e\}$. This is inspired by *network performance tomography* where the edges represent directed links in a communications network, and the $X_{e,t}$ represent additive link performance metrics such as the mean latency of a set of packets traversing edge e in a time slot t. In the network context, we cannot assume the $X_{e,t}$ to be directly measurable due, e.g. to cost constraints of providing equipment to perform such measurements in the network interior. Although multiscale properties have been

used to characterize network traffic and protocol and serve as features for anomaly detection, no current methods exist to localize these features to specific network links.

2.1.2 Multiscale Traffic Analysis and Anomaly Detection

Multiscale analysis has been proposed to characterize the complex nature of network traffic, examine the interaction between traffic demands and network protocols at different timescale, and compute features who can support anomaly detection. Specially, wavelet analysis provides a quantitative characterization through the set of wavelet coefficients associated with different timescales [17]. While self-similar behavior has been widely observed in network traffic traces since its original discovery in Ethernet traffic [18], wavelet analysis of WAN traffic reveals departures from self-similarity at timescales corresponding to round trip times. This is attributed to the flow control mechanism of TCP senders that is governed by acknowledgement from receivers [17]. Traffic source demands have been characterized as cascades, i.e. a multifactal hierarchy of arrivals of sessions, flows within sessions, and packets within flows, each member of the cascade presenting at its own timescale [19]. The set of signal energies across different timescales can be used as feature for anomaly detection; see [20, 21]. Wavelet-based multifractal models have been applied to the effective bandwidth estimation of network traffic flows [22]. These applications of wavelet-based multiscale analysis suggests that localization of observed signal energies to specific edges within a network can be a valuable tool in identifying the origins of network traffic anomalies.

Network Anomalies explore the vulnerabilities of network protocol and operations and bring down the normal traffic by taking available resources (*i.e.,* number of connections, bandwidth, etc) such as widely observed DDoS attack floods networks by an aggressive packet rate (*e.g.,* 1Tbps)[23]. Unlike the DDoS by attacking network with a high packet rate to take over network resources, low rate anomalies (*e.g.,* SlowComm [24], shrew [25], LoRDAS [26]) send very small attack traffic, which is around $10\% \sim 20\%$ of the total traffic and are extremely difficult to distinguish it from the normal traffic. Although the attack traffic sent by low rate anomalies is small, it still can cause terrible damage to networks. For example, shrew attack sends burst of packets with a rate that matches the RTO value would cause network frequently timeout. Spectral analysis over

signals generated by the low rate attack can capture the fluctuations over the number of packets or the transmission delay caused by the periodic burst traffic well and separate out the dominate frequencies of attack traffic from the normal traffic.

In this paper we shall be concerned with the multiscale signal tomography on networks, by which we mean the attribution to internal links of multiscale features observed from measurement of end-to-end signals. Additionally, we will explore the impact of low rate traffic anomalies present inside the network with multiscale analysis and discuss the possibility on localizing the traffic anomalies through the observed end-to-end measurements.

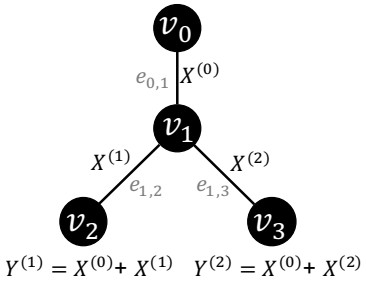


Figure 2.1: Two leaf tree model. $x^{(i)}$ are the internal statistics, and $y^{(i)}$ are the aggregated path statistics.

2.1.3 Problem Specification and Contribution

This paper addresses the problem of *how to infer per link energy at different scales in additive link metrics based on end-to-end measurements.* Figure 2.1 illustrates a simple two leaf tree topology where each node v_i denotes the real network device and edge $e_{i,j}$ denotes the connection between two network devices i and j. Our goal is to estimate the wavelet energy spectrum of the internal link $e_{0,1}$ from the collected path measurements $Y^{(1)}$ and $Y^{(2)}$ and we assume that internal links statistics are not prior knowledge. Let $X^{(i)}$ denote the actual statistics for each link and let $\hat{F}(X^{(0)})$ denote the estimation of the wavelet energy on the internal link $X^{(0)}$ (*i.e.,* the common

path),

$$\hat{F}(X^{(0)}) = f(Y^{(1)}, Y^{(2)}) \tag{2.2}$$

We shall model the function f to provide an unbiased estimator (*i.e.,* $\mathbb{E}[\hat{F}(X^{(0)})] = \mathbb{E}[F(X^{(0)})]$) based on observed path measurements $Y^{(1)}$ and $Y^{(2)}$. In addition to the established results on the two leaf tree model, we also scale to a large network topology that represents any possible scenarios of realistic networks. The traffic statistics we collected on each edge/path generally refer to the delay experienced from source to destination. The traffic patterns (*i.e.,* delay statistics) between the normal traffic and the traffic with anomalies are different but can be difficult to distinguish. Therefore, we provided mutiscale analysis for extracting energy of observed path measurements across different scales to recover the energy of each individual link and potentially localize the link with prominent energy. Our contributions are as follows:

- We introduced Discrete Wavelet Transform (DWT) analysis of time series of end-to-end measurements to infer the energy at multiple scales of measurements on the path interaction, and showed that the estimator is unbiased. (section 2.2)

- We showed how the theoretical results of the simple two leaf tree extends to general trees and thence to networks (Section 2.3 and Section 2.4)

- We showed how our estimator can be applied when nonstationary signals present within measurements and evaluated the scenarios with low rate anomaly traffic attacking network using network simulation (*i.e.,* NS3 [27]) in Section 2.5.

- We evaluate the performance of the proposed methods on RTT measurements gathered from the Ark platform and investigate network factors that influence performance of the method (Section 2.6).

- Asymmetric routing schemes were also discussed and we showed how to correctly model f in equation (2.2) to provide unbiased estimation in section 2.7.

2.2 Multiscale Binary Network Tomography

In this section, we shall describe the unbiased estimator for inferring the common path multiscale statistics based on the network tomography of end-to-end measurements using DWT under the assumption of mutual independence among link metrics and stationarity of time series.

2.2.1 Wavelet Decomposition of Signals

Let $Z = \{Z_t : t \in [T]\}$ be a signal with $T = 2^M$ components. The wavelet decomposition of Z involves writing it as linear combination of functions from an orthonormal basis as $Z_t = \sum_m \sum_n \tilde{Z}_n^m \phi_{n,t}^m$ where m and n label scale and translation parameters respectively. The wavelet coefficients are the scalar products with the corresponding basis vectors, namely $\tilde{Z}_n^m = \sum_{t\in[T]} Z_t \phi_{n,t}^m$. Let ϕ^m be a $T/2^m \times T$ orthonormal DWT matrix (*i.e.,* constructed by shifting the Daubechies wavelet filter [28] over time) with components $\phi_{n,t}^m$ such that wavelet coefficients $\tilde{Z}$ on scale m of time series Z can be obtained by $\phi^m Z$. DWT usually requires a dyadic length of sample size T (*i.e.,* $T = 2^M, M \in \mathbb{Z}^+$) but signal extrapolation methods (*e.g.,* zero-padding, reflect-padding, etc) can be used to construct dyadic length.

Wavelet coefficients $\tilde{Z}$ of a signal Z with length $T = 2^m$ on a certain scale m, which represents the difference of local averages and reflects the amount of fluctuation on corresponding frequencies, and has advantage in capturing time locations compared to Fourier transform. By using wavelet analysis on decomposing sample variance of time series collected over networks across multiple scales, we are able to characterize how the fluctuation change over time and how much it contributes to each scale (*i.e.,* each set of frequencies).

2.2.2 Unbiased Estimation of Common Path Multiscale Statistics

Figure 2.1 shows a simple network model with two leaf nodes and three internal links, and we shall use this model to illustrate our estimator for internal statistics by only observing path aggregation. Let $X^{(0)}, X^{(1)}, X^{(2)}$ be mutually independent signals on $[T]$ which are stationary in the sense that for any $S \subset [T]$ and $s \in [T]$ for which the translation $s + S \subset [T]$, $\{X_t : t \in S\}$ has the same distribution as $\{X_{s+t} : t \in S\}$. Set $Y^{(i)} = X^{(0)} + X^{(i)}$ for $i = 1, 2$ represents the

aggregated path statistics. Much of our analysis will rest of the properties of the product of the wavelet transformation matrix with its adjoint. We abstract this as a T-dimensional square matrix B for which we will henceforth assume has the constant signal $\mathbf{1}$ in its null space, and our analysis will then apply to wavelet bases for which this property holds.

Define a quadratic form F of a signal Z on $\mathbb{R}^T$ by

$$F(Z) = Z^T B Z \tag{2.3}$$

And define the estimator

$$\hat{F}(X^{(0)}) = \frac{1}{2}\left(F(Y^{(1)} + Y^{(2)}) - F(Y^{(1)}) - F(Y^{(2)})\right) \tag{2.4}$$

For two stationary signals $Z^{(1)}$ and $Z^{(2)}$ on $[T]$, we define a G function, which essentially represents a quadratic form of two signals

$$G(Z^{(1)}, Z^{(2)}) = \sum_{t,t',s,s' \in [T]} \mathbb{E}[Z_t^{(1)} Z_s^{(1)}] B_{t,t'} B_{s,s'} \mathbb{E}[Z_{t'}^{(2)} Z_{s'}^{(2)}] \tag{2.5}$$

Let Δ be the bias of the estimator and

$$\Delta = \hat{F}(X^{(0)}) - F(X^{(0)}) \tag{2.6}$$

Theorem 1. *(i)* $\mathbb{E}[BX^{(i)}] = 0$

(ii) $\mathbb{E}[\Delta] = 0$ *and hence* $\mathbb{E}[\hat{F}(X^{(0)})] = \mathbb{E}[F(X^{(0)})]$

(iii) $\mathrm{Var}(\Delta) = G(X^{(0)}, X^{(1)}) + G(X^{(1)}, X^{(2)}) + G(X^{(2)}, X^{(0)})$

Proof. (i) Since the $X^{(i)}$ are stationary, $\mathbb{E}[X] = \mathbb{E}[X_T]\mathbf{1}$ and the results follows by assumption on the null space of B.

(ii)

$$\Delta = X^{(1)} \cdot BX^{(2)} + X^{(2)} \cdot BX^{(0)} + X^{(0)} \cdot BX^{(1)} \tag{2.7}$$

which has zero expectation as result of (i).

(iii) The covariances amongst distinct terms in (2.7) are zero since, for example

$$\begin{aligned}
&\mathrm{Cov}(X^{(1)} \cdot BX^{(2)}, X^{(2)} \cdot BX^{(0)}) && (2.8)\\
&= E[X^{(1)}] \cdot B\,\mathbb{E}[(X^{(2)})^T X^{(2)}] \cdot B\,\mathbb{E}[X^{(0)}] && (2.9)\\
&\quad -E[X^{(1)}] \cdot B\mathbb{E}[X^{(2)}]E[X^{(2)}] \cdot B\mathbb{E}[X^{(0)}] && (2.10)\\
&= 0 && (2.11)
\end{aligned}$$

by (i). The variance terms follow the pattern $\mathbb{E}[(X^{(0)} \cdot BX^{(1)})^2]$ from which the stated form follows (the square mean term in the variance is 0, similarly as in (ii)) □

Theorem 1 described the techniques of the unbiased estimator in (2.4) for the common path statistics based on observing aggregated path statistics by assuming mutual independence and stationarity of edge metrics. We shall use this fundamental result to show that the estimator $\hat{F}(X^{(0)})$ is to estimate the wavelet energy on the common path.

2.2.3 Wavelet Energy Estimation

The aforementioned estimator is constructed by the abstract matrix B. We now show how the estimation of wavelet energy is related to the estimator in (2.4).

The energy of a signal Z at scale m is the summation of square of wavelet coefficients $\tilde{Z}_n^m$

$$\sigma_m^2(Z) = \sum_n (\tilde{Z}_n^m)^2 = (\tilde{Z}^m)^T \tilde{Z}^m \tag{2.12}$$

This is in general a *stochastic* quantity. Denote

$$B_{t,t'}^m = \sum_n \phi_{n,t}^m \phi_{n,t'}^m \tag{2.13}$$

Then $\sigma_m^2(Z) = F_m(Z)$ where F_m is the quadratic form $F_m(Z) = Z^T B^m Z$. Defining $\hat{F}_m$ analogously with the estimator (2.4) then we have the following:

Theorem 2. *For each m, $\hat{F}^m(X^{(0)})$ is equal to $\sigma_m^2(X^{(0)})$ in expectation.*

We can bound the variance of the difference $\Delta_m = \hat{F}_m(X^{(0)}) - \sigma_m^2(X^{(0)})$ in terms of the energies of the underlying link processes by means of the following Theorem.

Theorem 3. $G(Z^{(1)}, Z^{(2)}) \leq \mathbb{E}[\sigma_m^2(Z^{(1)})\sigma_m^2(Z^{(2)})]$ *and hence* $\mathrm{Var}(\Delta_m) \leq \mathbb{E}[\sigma_m^2(X^{(0)})\sigma_m^2(X^{(1)})] + \mathbb{E}[\sigma_m^2(X^{(0)})\sigma_m^2(X^{(2)})] + \mathbb{E}[\sigma_m^2(X^{(1)})\sigma_m^2(X^{(2)})]$.

Proof of Theorem 3. Let G_m denote the version of G obtained using $B = B^m$ in (2.5). Then

$$\begin{aligned} G_m(Z^{(1)}, Z^{(2)}) &= \sum_{n,n'} \mathbb{E}[\tilde{Z}_n^{(1),m} \tilde{Z}_{n'}^{(1),m} \tilde{Z}_{n'}^{(2),m} \tilde{Z}_n^{(2),m}] && (2.14) \\ &= \mathbb{E}[((Z^{(1),m})^T \tilde{Z}^{(2),m})^2] && (2.15) \\ &\leq \mathbb{E}[(Z^{(1),m})^T \tilde{Z}^{(1),m} \ (Z^{(2),m})^T \tilde{Z}^{(2),m}] && (2.16) \\ &= \mathbb{E}[\sigma_m^2(Z^{(1)})\sigma_m^2(Z^{(2)})] && (2.17) \end{aligned}$$

□

Thus $\sigma_m^2(X^{(0)})\sigma_m^2(X^{(1)}) + \sigma_m^2(X^{(0)})\sigma_m^2(X^{(2)}) + \sigma_m^2(X^{(1)})\sigma_m^2(X^{(2)})$ is an upper bound for $\mathrm{Var}(\Delta_m)$ in expectation.

2.3 Network Generalizations

The foregoing work in Section 2.2 has focused on the canonical two-leaf tree. In this section we outline network generalizations under the standing assumption that the time series X_e associated with edge $e \in E$ are mutually independent and stationary. We assume a directed path $P_{uv} \subset E$ is specified from each vertex u to v in V such that for each $u \in V$ the edges $\cup_{v \in V \setminus \{u\}} P_{uv}$ forms a tree, and likewise $\cup_{v \in V \setminus \{u\}} P_{vu}$.

The key technical result enabling our approach is the following lemma, which says that F is additive in expectation on mutually independent signals.

Lemma 4. *Let X and X' be mutually independent and stationary signals on T. $\mathbb{E}[F(X + X')] = \mathbb{E}[F(X)] + \mathbb{E}[F(X')]$*

Proof. $F(X+X') = F(X) + F(X') + (X')^T BX + X^T BX'$. Since X and X' are independent, then under the null-space assumption on B, these last two terms have zero expectation. □

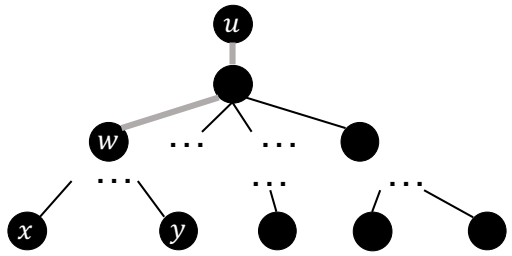


Figure 2.2: General topology. Grey lines indicate the common path of leaf nodes x and y

Now consider a vertex u and the source tree $T = (V_T, E_T)$ formed by the paths $\{P_{uv} : v \in R\}$ for some $R \subset V$. (Receiver trees can be treated in the same manner.) Without loss of generality we assume that R comprises leaf nodes of this tree; if not we can partition T into trees that have this property. For any interior node $w \in V_T$ let $R_w \subset R$ be the set of leaf nodes descended from w. Figure 2.2 shows a general topology with a source node and multiple leaf nodes. To any leaf pair $\{x, y\} \subset R_w$ we associated a binary logical tree with edges formed by the subpaths $u \to w$, $w \to x$ and $w \to y$. Thus based the composite signals $Y^{(x)} = \sum_{e \in P_{ux}} X_e$ we form for each scale m the estimator

$$\hat{F}_m^{xy}(Y^{(w)}) = \frac{1}{2}\left(F_m(Y^{(x)} + Y^{(y)}) - F_m(Y^{(x)}) - F_m(Y^{(y)})\right) \tag{2.18}$$

of $\sigma_m^2(Y^{(w)})$, which is unbiased according to Theorem 1. Convex combinations of estimates from distinct pairs are also unbiased and are expected to reduce variance, for example the average $\hat{F}_m^{\mathrm{avg}}(Y^{(w)}) = (|R_w|(|R_w|-1))^{-1} \sum_{x \neq y \in R_W} F_m^{xy}(Y^{(w)})$; see also [29].

Due to Lemma 4, for any two internal vertices w, w' in the original tree T, we can form an unbiased estimated of the energy $\sigma_m^2(Y^{(w,w')})$ for the total signal $Y^{(w,w')} - \sum_{e \in P_{ww'}} X_e$ associated with the path from w to w' by $\hat{F}_m^{\mathrm{avg}}(Y^{(w)}) - \hat{F}_m^{\mathrm{avg}}(Y^{(w')})$; where we assume w' is closer to the root

u of T than w.

Finally, a procedure for fusing a set of source and receiver trees derived from end-to-end measurements on any increasing path additive metric has recently been established in [30], including non-symmetric routing. Applied to the present case, this would enable establishing the virtual network topology that expresses the common contributions of network edges to end-to-end signal energy at any energy scale.

2.4 Model

An unbiased estimator was proposed to estimate the wavelet energy across different scales of the internal links based on aggregated end-to-end statistics. In this section, we demonstrate those statistical properties with a model satisfying requisite constraints (*i.e.,* independent and stationary).

2.4.1 Discrete synthetic process

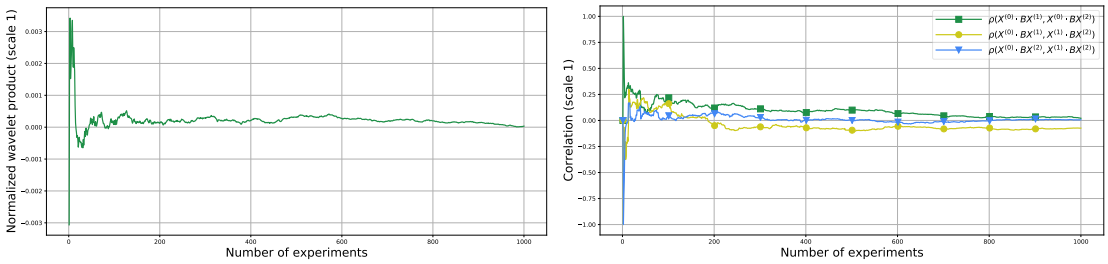
Synthetic signals are used to construct a network with mutual independent links statistics and stationary time series of links. Let ΔT denote as the time duration for a signal X with $T = 2^M$ ($M \in \mathbb{Z}^+$) components and let Δt, Δf be the time increment and the frequency increment (*i.e.,* $\Delta f = \frac{1}{\Delta T}$) between adjacent components. Let $X(f)$, S_X denote as the Fourier transform of the signal X and the power spectral density (PSD) of X respectively. Therefore, discrete time series can be generated by applying inverse Fourier transform (IFT) with random phases θ.

$$X^{(i)} = IFT(\sqrt{S_{X_i} \cdot \Delta f} \cdot e^{i\theta}) \cdot T \tag{2.19}$$

The fundamental concept on constructing synthetic signals for representing links statistics (*i.e.,* delay) is having different PSD to characterize potential variations over different frequencies. We used two PSD functions (*i.e.,* $S_a = (1/(0.1+(\frac{f}{10})^2))^2$ and $S_b = ((f/10)^2 \cdot (1+(\frac{f}{10})^2))^2$) with S_a, S_b characterizing large fluctuations within low frequency component and high frequency component respectively to model signals collected over network links, and we will further illustrate how the proposed estimator captured those dominant energy of the internal links over different frequencies generated by distinct PSDs.

2.4.2 Simple Binary Network Configuration

In the simple binary tree of figure 2.1, let v_0, v_2, v_3 denote three boundary nodes, which all connect to a interior node v_1 by edges $e_{0,1}$, $e_{1,2}$, $e_{1,3}$ respectively. Let $X^{(0)}$ be the time series generated by S_a and $X^{(1)}$, $X^{(2)}$ be the time series generated by S_b. Let $Y^{(0)}$, $Y^{(1)}$ be the aggregated time series from v_0 to v_2 and v_0 to v_3 such that $Y^{(0)} = X^{(0)} + X^{(1)}$, $Y^{(1)} = X^{(0)} + X^{(2)}$. Therefore, according to the proposed estimator 2.4, the wavelet energy on the common link (*i.e.,* $X^{(0)}$) can be estimated based on the path observations statistics $Y^{(0)}$ and $Y^{(1)}$. We configured the length T of time series $X^{(0)}$, $X^{(1)}$ and $X^{(2)}$ to be 2^{15} and repeated 5000 experiments with random phases θ in each experiment. For all experiments, we used *Haar* wavelet as our mother wavelet for simplicity on computing difference of adjacent local average to perform wavelet analysis and all other Daubechies wavelets [28] with orthonormal basis can also be used on constructing matrix B in (2.13).



(a) Normalized wavelet product of edge 0 and 1 (b) Correlation of two distinct wavelet products

Figure 2.3: Convergence on scale 1 with time series length $T = 2^{15}$

2.4.3 Demonstration of Statistical Properties

We demonstrate conformance to the statistical properties derived in previous Theorems for example processes of Section 2.4.2. Figure 2.3a illustrates unbiasedness by computing the average of wavelet products $X^{(0)} \cdot BX^{(1)}$ from distinct edges over a set independent experiments, and displaying result (normalized by the average of $||X^{(0)}|| * ||X^{(1)}||$) as a function of the number of

experiments. According to (2.6), the bias Δ is essentially consist of the product of wavelet coefficients at certain scale of two signals. We observe how the average over experiments approaches zero as the number of experiments increases for the normalized wavelet product, thus proved the unbiasedness.

Figure 2.3b shows that the empirical correlation amongst distinct terms in (2.7) approaches zero as the number of experiments increase, and hence difference between the empirical variance and the given expression in Theorem 1(iii) converges to zero with the number of experiments. The results of Figure 2.3 are for the finest scale $m = 1$, which also bounds the behavior for larger scales whose estimates are linear convex combinations of those from $m = 1$.

The variance of the estimator (2.4) is bounded by the wavelet energy products (Theorem 3). Figure 2.4 shows the variance of the estimator with $T = 2^{15}$ across all scales converge to a stable value as the number of experiments increase and the wavelet energy is normalized by $1/T$. As the stationary time series $X^{(i)}$ ($i \in \{0, 1, 2\}$ are generated according to a certain power spectral density function S_i, the variance of wavelet coefficients on a certain scale m is unchanged over time. Legend of figure 2.4 also showed the bounded value of wavelet energy products (*i.e.,* $\sigma_m^2(X^{(0)})\sigma_m^2(X^{(1)}) + \sigma_m^2(X^{(0)})\sigma_m^2(X^{(2)}) + \sigma_m^2(X^{(1)})\sigma_m^2(X^{(2)})$) on each scale and variances across all scales are smaller than the upper bounds as we proved in Theorem 3.

2.4.4 Internal Link Energy Estimation

The estimator $\hat{F}(X^{(0)})$ in (2.4) gives an unbiased multiscale energy estimation on the common edge $e_{0,1}$ from the end-to-end measurements $Y^{(i)}$. Under the dependence and stationary assumption, the wavelet energy on uncommon path $\hat{F}(X^{(i)})$ for $i = 1, 2$ can also be estimated by equation (2.20) with bias $\Delta_i = \hat{F}(X^{(i)}) - F(X^{(i)})$.

$$\hat{F}(X^{(i)}) = F(Y^{(i)}) - \hat{F}(X^{(0)}) \tag{2.20}$$

Lemma 5. *$\hat{F}(X^{(i)})$ is unbiased and* $\mathrm{Var}(\Delta_i) = \mathrm{Var}(\Delta)$

Proof. By lemma 4, $\mathbb{E}[F(Y^{(i)})] = \mathbb{E}[F(X^{(0)})] + \mathbb{E}[F(X^{(i)})]$, hence $\hat{F}(X^{(i)})$ is unbiased. Bias

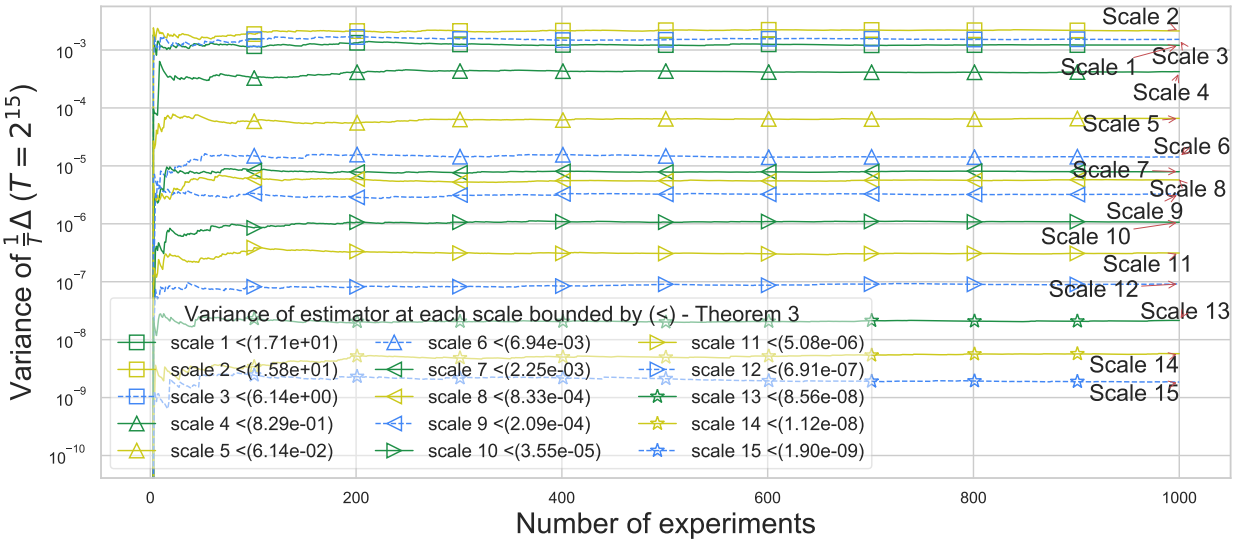


Figure 2.4: Variance of estimator on the common path with time series length $T = 2^{15}$

$\Delta_i = \hat{F}(X^{(i)}) - F(X^{(i)}) = 2X^{(0)}BX^{(i)} - \Delta$, where Δ defined in equation (2.7), then $\mathrm{Var}(\Delta_i) = \mathrm{Var}(\Delta)$ □

To illustrate the correctness of the unbiased estimators for common path and uncommon path by the model, we choose a dyadic length of 2^{15} (similar results would be obtained for different dyadic length) for time series $X^{(i)}$ ($i \in 1, 2, 3$) to construct aggregated path statistics $Y^{(j)}$ ($j \in 1, 2$).

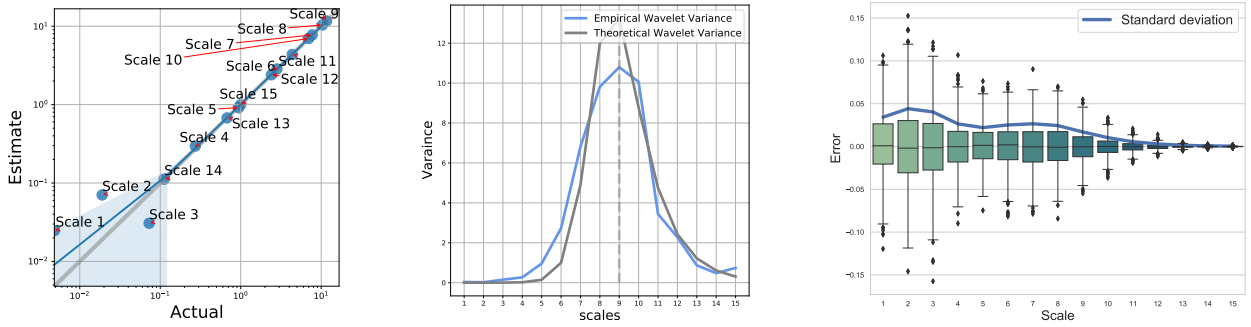
2.4.4.1 Common Link

Figure 2.5 gives estimated wavelet energy across multiple scales with time series length being $T = 2^{15}$ compared with the actual wavelet energy on the common path (*i.e.,* e_0) with the estimator (2.4). As we can see from the figure 2.5a, the estimated values (*i.e.,* y-axis) across multiple scales are very close to the actual values as those points align with the diagonal line (*i.e.,* gray line) well. We fit a linear regression model for the points (*i.e.,* blue line) and the translucent band shows confidence interval using a bootstrap. We expected larger scales (*i.e.,* scale 9) has more wavelet energy compared to smaller scales as the magnitudes of small frequencies dominate the PSD of

the common path. The theoretical variance σ^2 of the signal $X^{(0)}$ with spectrum S_X is decomposed into different scales and we have

$$\sigma^2 = \int S_X df = \frac{1}{T} \sum_m ||\tilde{X}_m^{(0)}||^2 \tag{2.21}$$

where $||\tilde{X}_m^{(0)}||^2$ is the wavelet energy at scale m of $X^{(0)}$. Specifically, the wavelet energy σ_m^2 at scale m is essentially summarizing information in the corresponding spectrum S_X such that $\sigma_m^2 \approx 2 \int_{\frac{1}{4m}}^{\frac{1}{2m}} S(f) df$ [31]. Figure 2.5b shows the estimated wavelet energy (*i.e.,* empirical wavelet variance $\frac{1}{T}||\tilde{X}_m^{(0)}||^2$) and the theoretical variance at each scale m (*i.e.,* $2 \int_{\frac{1}{4m}}^{\frac{1}{2m}} S(f) df$). Figure 2.5c shows errors generated from 5000 experiments with standard deviation (*i.e.,* blue line) and we can see those error are very small, which also characterized the variance of Δ in equation 2.7 at each scale.



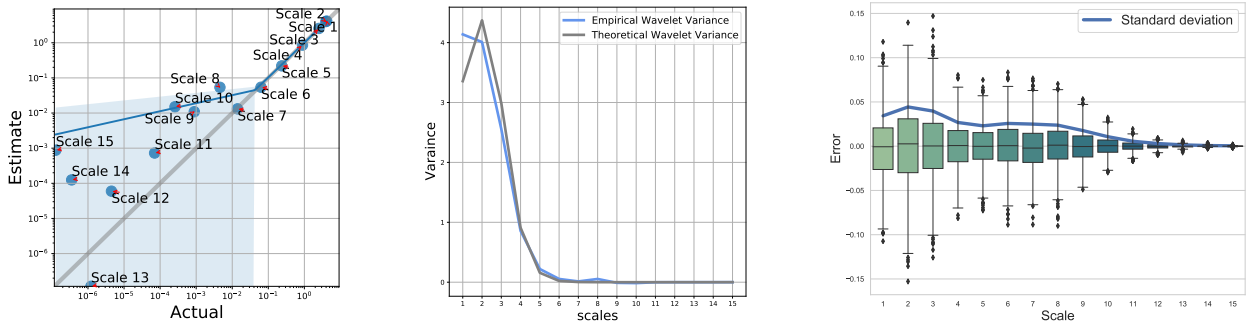
(a) Estimates across all scales (b) Variance and Energy (c) Errors across all scales

Figure 2.5: Common path wavelet energy estimation, normalized by the length of time series $1/T$

2.4.4.2 Uncommon Link

Similarly, wavelet energy for edges e_1 and e_2 can be estimated with equation (2.20). Here we show the estimates for e_1 in figure 2.6a. Those estimates are accurate with small deviations over scale 6 and large wavelet energy concentrated in small scales (*i.e.,* 1-5), which is expected as the PSD assigned on uncommon path has larger magnitude over high frequencies. Similar to the

common link, figure 2.6b describes the relationship between the theoretical variance and wavelet energy decomposition and figure 2.6c shows the errors across all scales as result of variance of our estimator.



(a) Estimates across all scales (b) Errors across all scales (c) Errors across all scales

Figure 2.6: Uncommon path wavelet energy estimation normalized by the length of time series $1/T$

2.5 Multiscale Wavelet Energy Inference On Low Rate Anomalies

The established unbiased estimator 2.4 stands on assumptions of mutual independence and stationarity of links statistics. With ideal setup on $Y^{(0)} = X^{(0)} + X^{(1)}$, $Y^{(1)} = X^{(0)} + X^{(2)}$ in sec 2.4.2, the aggregated path statistics of $Y^{(0)}$ and $Y^{(1)}$ experienced the identical time series $X^{(0)}$ on common path, which might be different in a realistic network. Additionally, abnormal traffic with unstable variance that could be caused by periodic patterns or low rate anomalies breaks the stationarity assumption, especially when the rate of patterns is changing over time. In this section, we shall describe how to estimate the common path and uncommon path statistics with low rate anomalies introduced on networks and loose the assumption of stationarity. We shall also discuss how to localize the path with abnormal traffic.

2.5.1 Unbiased Estimation with Nonstationary Signals

By Theorem 1, $\mathbb{E}[BX] = 0$ for a stationary signal X, which is invalid if X is nonstationary. We use the figure 2.1 again to illustrate the estimation with nonstationry signal presents and assume at

least one non-stationary signal presents in $X^{(i)}(i \in \{0,1,2\})$ and they are mutually independent.

Lemma 6. *If no more than one signal is nonstationary, then* $\mathbb{E}[\hat{F}(X^{(0)})] = \mathbb{E}[F(X^{(0)})]$.

Proof. $\mathbb{E}[X^{(i)} \cdot BX^{(j)}] = \mathbb{E}[X^{(j)} \cdot BX^{(i)}] = 0$ $(i \neq j)$, then $\mathbb{E}[\hat{F}(X^{(0)})] = \mathbb{E}[F(X^{(0)})]$. □

The estimator in (2.4) is unbiased even if one of the link statistics of the two leaf tree model is nonstationary and Theorem 1 (ii) and (iii) also remain valid. This is very useful when we estimate the wavelet energy of nonstationary signal presents in one of the links on the simple two leaf tree model illustrated in figure 2.1. We now assume that any collected time series on figure 2.1 can be nonstationary. Let $d^{(i)}$ $(i \in \{0,1,2\})$ be the backward differences of time series $X^{(i)}(i \in \{0,1,2\})$ such that $d^{(i)}$th order backward difference of $X^{(i)}$ are second order stationary time series with zero mean. Let L denote the even length of Daubechies wavelet filter [28].

Lemma 7. *If* $L \geq 2 \cdot max\{d^{(i)}\}$*, then* $\mathbb{E}[\hat{F}(X^{(0)})] = \mathbb{E}[F(X^{(0)})]$.

Proof. By assuming $L \geq 2 \cdot d^{(i)}$, the wavelet coefficients of $X^{(i)}$ generated by the wavelet filter with length L is a stationary process with zero mean [31]. Therefore, $\mathbb{E}[X^{(i)} \cdot BX^{(j)}] = 0$ $(i \neq j)$ and $\mathbb{E}[\hat{F}(X^{(0)})] = \mathbb{E}[F(X^{(0)})]$. □

Lemma 7 provides a constrain on nonstationary time series to have an unbiased estimator $\hat{F}(X^{(0)})$ with suitable wavelet filter length L. With $L \geq 2 * max\{d^{(i)}\}$, we make sure that wavelet coefficients for each collected time series $X^{(i)}$ generated by wavelet filter with length L are second-order stationary time series with zero mean and hence the estimator $\hat{F}(X^{(0)})$ is unbiased.

2.5.2 Network Simulation on Low Rate Anomalies

We used a packet-level simulation NS3 [27] to illustrate the effect of periodic patterns on the link statistics. In this demonstration we use the topology from Internet 2 [32] shown in Figure 2.7. Designate v_s, v_i, v_a and v_c to be the nodes at *Sunnyvale*, *Indianapolis*, *Atlanta* and *Chicago* respectively and let e_{mn} be the packet forwarding path (*i.e.,* logic edge) between two distinct nodes m and n. Let the endpoint at v_s denote the monitor node sending probing traffic to the two destinations v_c and v_a along common paths until routes diverge at v_i. The logical subtopology (*i.e.,*

Figure 2.7: Network topology map and traffic routes

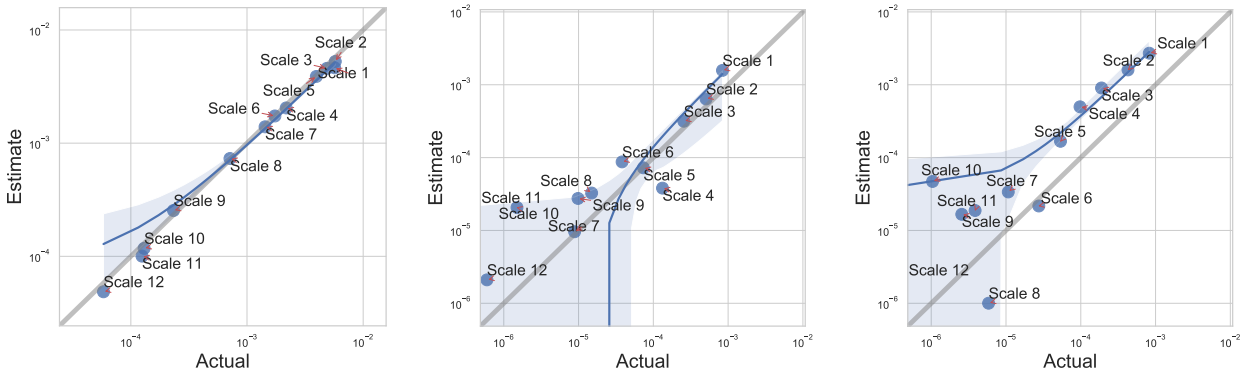
black nodes in figure 2.7) would be a two leaf tree and the gain by simulating this under Internet2 topology is to bring sufficient dynamic background TCP traffic.

Each network node sends TCP traffic (64 bytes per packet) at rate of 0.2 Mbps to every other node to saturate 10Mbps connection links as indicated by the grey edges between nodes in Figure 2.7. Packet transmissions occur in an on-off process with exponentially distributed holding times with mean 0.5 second, and transmission rates configured to a nominal size of 10 Mbps in order to limit simulation execution complexity. Link delays were configured to be $5ms$ over the network. End-to-end delay measurements from monitor node v_s to v_c and v_a were collected using UDP traffic with $0.01s$ interpacket time and the generated time series comprising 2^{12} packets. Each node is configured with FIFO policy buffer of 100 packets storage and we configured this topology to have a static routing to guarantee each packet forwarded as we expected. At first sight this probing rate seems small, however it represents roughly 1 probe packet per hundreds background packets. In practice it would be scaled up in proportional to actual traffic rates which might be 3 or 4 orders of magnitude larger in real networks.

2.5.2.1 Wavelet Energy Estimation on Internal Links

Probing packets to two destinations v_a and v_c were continuously sent out from v_s independently. Let $X_c^{(0)}$ and $X_a^{(0)}$ denote the time series experienced on the common path e_{si} from v_s to v_i with v_c and v_a as the destination respectively. Let $X^{(1)}$ and $X^{(2)}$ denote the time series experienced on the

uncommon path e_{ic} and e_{ia} respectively and the observed statistics $Y^{(0)} = X_c^{(0)} + X^{(1)}$ and $Y^{(2)} = X_a^{(0)} + X^{(2)}$ are the time series of aggregated path delay measurements. $X_c^{(0)}$ and $X_a^{(0)}$ are collected by independent probing packets passing through the common path with different destinations. Figure 2.8a shows the estimation of wavelet energy across all scales on the common path and figure 2.8b, 2.8c shows the uncommon paths estimation. As we can see from the figure, both inferred and actual wavelet energy across all scales present very small fluctuations experienced on the common path, though small errors are observed and they come from 1) measurements on the common path for two destinations are not strictly the same (*i.e.,* $X_c^{(0)}$ and $X_a^{(0)}$), and we use the average of $X_c^{(0)} + X_a^{(0)}$ as common path measurements. 2) small correlation among measurements and $\mathbb{E}[X^{(i)} \cdot BX^{(j)}] \neq 0$. For most of the scales, the wavelet energy are concentrated with in $1e{-}5 \sim 1e{-}2$ with small deviation. The wavelet energy of path delay under normal traffic exhibits a very stable behavior (*i.e.,* small variance).



(a) Common path (b) Uncommon path (c) Uncommon path

Figure 2.8: Estimation of the wavelet energy without attack traffic

2.5.2.2 *Low Rate Attack*

We now introduce the potential anomalies over the common path to investigate how the attack change the energy spectrum with different attack rates. We simulate the low rate attacks as a small amount of burst traffic injected into the network over time with a stable rate (can be

changed). The attack traffic was sent every 1 second but with different burst time targeted on the common path (*i.e.,* e_{si} from *Sunnyvale* to *Indianapolis*). The observed time series contain the impact of the attack traffic that is generally nonstationary time series. However, the link statistics on the uncommon path is unaffected by the low rate attacks and should represent a stable collection. According to Lemma 6, the expectation of wavelet product of one stationary and one nonstationary time series is equal to zero, hence we have unbiased estimation on the common path and uncommon path. We configured 4 different low rate attacks with different parameters ($burst\ time, burst\ rate, attack\ period$) as shown in the legend of figure 2.9. Low rate attack with ($0.5s, 2Mbps, 1s$) introduced 1 Mbps traffic across the common path, which is around 10% of background traffic, and ($0.5s, 4Mbps, 1s$) contributes to 20% of background traffic. Additionally, we decreased the burst time from $0.5s$ to $0.25s$ and $0.1s$ to see how the estimator performs with low rate traffic under 10% of normal background traffic.

Figure 2.9 shows the wavelet energy estimation and upper / lower error bars for common path and uncommon paths with low rate attacks introduced on the common path based on 50 experiments for each attack setting. As we can see from the common path estimation, the wavelet energy increases with longer burst time and larger burst rate. Smaller burst time brings more rapid changes on the collected time series, which lower the scale with maximum wavelet energy (*i.e.,* from scale 6 of ($0.5s$, $4Mbps$, $1s$) to scale 4 of ($0.1s$, $2Mbps$, $1s$)). The uncommon paths, on the other hand, is stable with small wavelet energy captured within 1e-2 across all scales. The errors of both common path and uncommon paths are very small indicated by the error bars. By using end-to-end measurements (v_s to v_c and v_a) to estimate the wavelet energy of internal link, we accurately captured and localized the link (*i.e.,* e_{si}) with large fluctuation contributed by low rate anomalies.

2.6 Demonstration Application: Archipelago RTT Measurements

In this section we consider an extension of our approach to tomography from realistic round-trip time (RTT) measurements taken by the globally distributed Ark measurement platform [33]. We first described how to obtain the useful RTT data sets from Ark measurements (sec 2.6.1) and how the unbiased estimation of end-to-end measurements can be further applied in RTT mea-

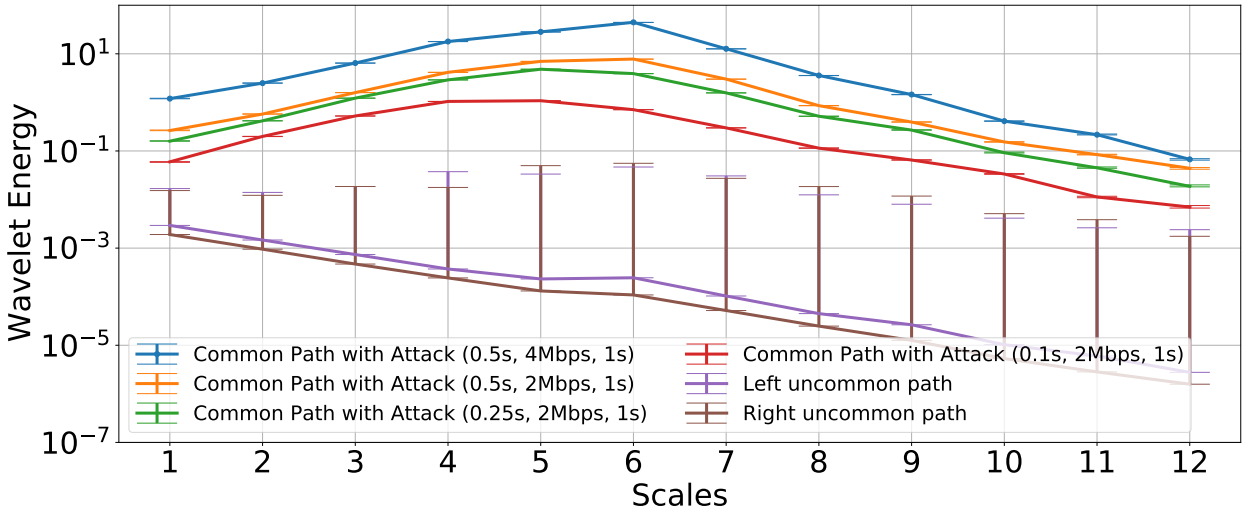


Figure 2.9: Internal links estimation with embedded low rate attacks (*burst time, rate, period*) on common path

surements on the abstracted topologies (sec 2.6.2). We further provided detailed analysis on how to estimate internal link energy from several constructed topologies and how to localize internal link/path with large fluctuation (sec 2.6.3 and sec 2.6.3.2).

2.6.1 Use of the Ark Platform

2.6.1.1 Ark Platform Capabilities

The Ark platform comprises 145 active monitor nodes deployed across 115 cities in 47 countries. The platform is equipped with several monitoring functions. For the present paper, we are are concerned with Ark's traceroute measurements, launched from monitor nodes to randomly selected addresses within each of the routed /24 networks, of which there are over 10 million. Each monitor node sends probe packets at an aggregate rate of 100pps. The monitors use Scamper [34] to collect RTT measurements to the destination host and each intermediate hop RTT via ICMP responses.

2.6.1.2 RTT Data for Subpaths

In this study we use the Ark platform to provide time series of RTT measurements for subpaths forming tree subtopologies rooted at a monitoring node. Specifically, given a monitoring node v_0,

we form a raw time series of RTT measurements between v_0 and any other node v_i traversed by probe packets from v_0, comprising the RTT values associated all ICMP responses from v_i back to v_0 generated by the probes generated by v_0. Thus the other v_i are not required to be monitoring nodes.

2.6.1.3 Derived Time Series and Data Quality

Ark reports probe dispatch times at a 1 second granularity, and therefore we derive a time series for study by averaging all raw RTT values for measurement from v_0 to a given node v_i associated with each 1 second bin. Since we wish to provide time-series of RRT measurements for a set of nodes $\{v_i\}$ we are in practice constrained in the choice of location of v_i relative to v_0 in terms of hop count, since with a greater hop separation, fewer destinations will be reached from v_0 through v_i and hence the sparser the time series, since some windows may not have any associated RTT measurements. For this reason, in this study we kept the hop count separations as small as possible. The raw RTT measurements suffer from incompleteness *i.e.,* if no response from some of the intermediate hops or their IP address cannot be resolved. After filtering these events, we truncated length of measurements T to satisfy the dyadic length condition $T = 2^m$ ($m \in \mathbb{Z}^+$) for computing wavelet coefficients.

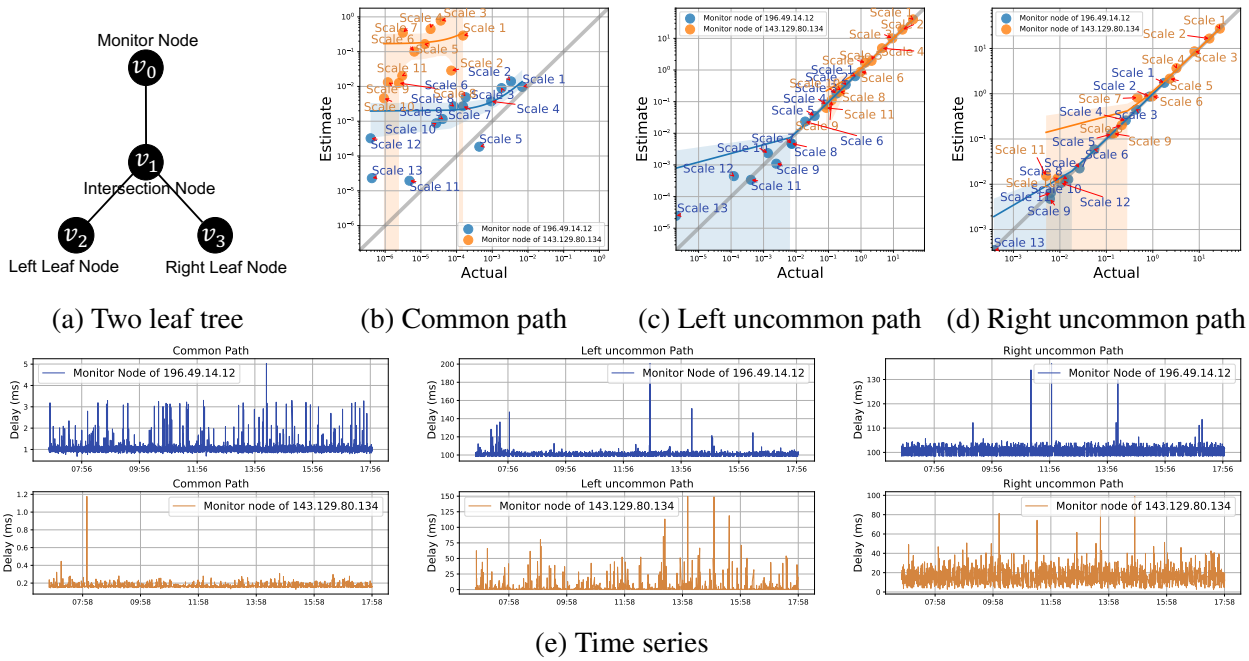
Table 2.1: Example on constructing subtopology (two leaf tree) for 2019-01-01 trace

Monitor node	**Intersection Node**	**Leaf nodes**	**#. Measurements**
196.49.14.12	81.199.8.105	10.46.0.209, 10.46.0.205	11519 ($\sim 2^{13}$)
10.42.4.201	10.42.4.1	96.120.4.253, 162.151.154.121	2095 ($\sim 2^{11}$)
143.129.80.134	143.129.80.190	143.129.67.252, 192.168.148.101	2665 ($\sim 2^{11}$)

2.6.2 Pitfalls for Statistical Modelling of RTT Series

We illustrate the estimation using the two leaf tree logical subtopology of figure 2.1, where $X^{(i)}$ ($i \in \{0, 1, 2\}$ represent RTT time-series associated with edges (v_0, v_1), (v_1, v_2) and (v_1, v_3)

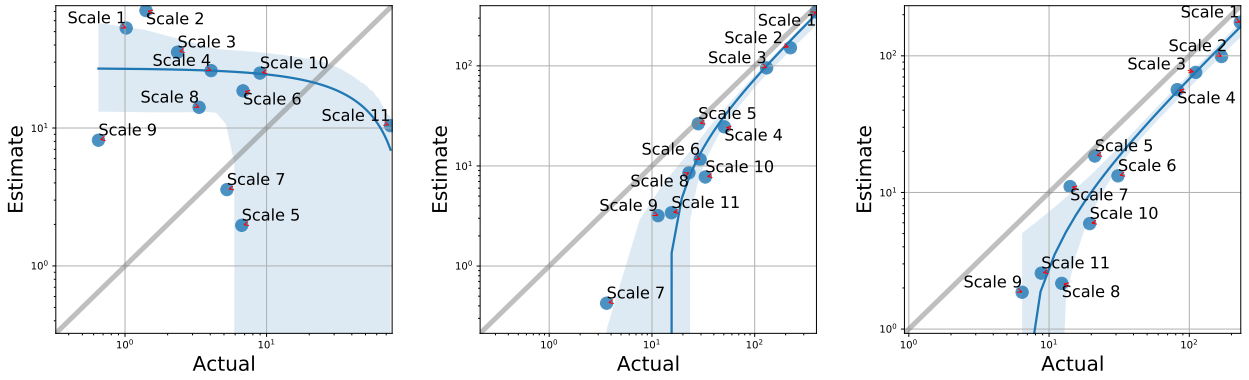
respectively. The time-series of RTT measurements collected by Ark then correspond to $X^{(0)}$, $Y^{(0)} = X^{(0)} + X^{(1)}$ and $Y^{(1)} = X^{(0)} + X^{(2)}$, i.e., the common path and complete paths RTT measurements. Under the assumption that RTT measurements $X^{(0)}$ taken on the common path for each destination as identical then the estimator $\hat{F}(X^{(0)})$ would unbiased. Previous work has used similar assumptions to justify tomography using trains unicast probe packets striped across a set of destinations in order to emulate multicast probes [35, 36]. These approaches we justified by control of the probing mechanism (enabling dispatch of back to back probes) together with empirical studies on the correlations between loss between packet probes. Unfortunately the Ark platform does not permit such detailed control over probes to different destinations, which limits the extent to which we can evaluate the general efficacy of our approach. Nevertheless, we are able to observe reasonable performance in some cases, and comment on the factors that militate against it in others; see Section 2.6.3. In the case of nonstationary RTT series that may occur due, e.g., to routing instabilties, we appeal to the results of Section 2.5.1.



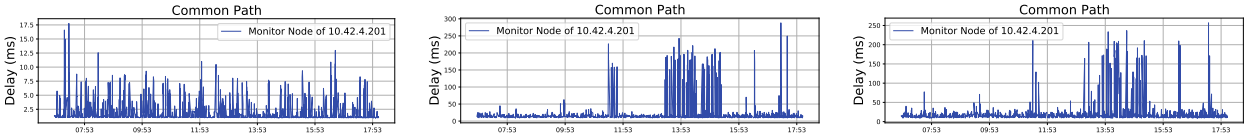
(a) Two leaf tree (b) Common path (c) Left uncommon path (d) Right uncommon path

(e) Time series

Figure 2.10: Estimation of the wavelet energy of two monitor nodes (*i.e.,* `196.49.14.12` and `143.129.80.134`) where the energy is normalized by the length of time series ($1/T$).



(a) Common path (b) Left uncommon path (c) Right uncommon path



(d) Common path (e) Left uncommon path (f) Right uncommon path

Figure 2.11: Estimation of the wavelet energy of the monitor nodes `10.42.4.201` where the energy is normalized by the length of time series ($1/T$).

2.6.3 Evaluations on Subtopologies of the Ark platform

2.6.3.1 Abstracted Two Leaf Tree Topologies

We first conducted three identical subtopologies constructed by three different monitor nodes as provided in Table 2.1 and illustrated in figure 2.10a. The estimation of first two subtoplogies of monitor node `196.49.14.12` and `143.129.80.134` is shown in Figure 2.10(b), (c), (d) of common path, left uncommon path, and right uncommon path respectively. The true common path measurements were calculated by the average of the two common path RTT measurements with different destinations v_2 and v_3, and the true uncommon path measurements were calculated by the difference between path measurements and the common path measurements with same destination. Although the common path estimations are not aligned with the diagonal line well in the figure 2.10b, the true wavelet energy of the common path and those errors deviated from the true value are small and both estimates and actual values are within 1, which indicates a stable link delay with very small variance for the common path of both topologies. The estimations of

wavelet energy on the uncommon paths, on the other hand, are accurate with scale 1 having largest wavelet energy for both topologies of two monitor nodes. As scale 1 contains the highest frequency information, we should expect a few rapid changes on the uncommon path at some time locations to contribute to the wavelet energy (*i.e.,* equation (2.21)).

Figure 2.10e shows collected RTT time series taken in *2019-01-01* for both the common path (average of two measurements) and uncommon paths of the two monitor nodes. For the common path statistics (*i.e.,* first column of figure 2.10e), we can see that most of the delay measurements are captured within 3 ms for monitor node `196.49.14.12` and 0.3 ms for monitor node `143.129.80.134` with very small variance (*i.e.,* 2.7e-2 and 8e-4 respectively). However, we noticed few large fluctuations (*i.e.,* spikes) for the uncommon paths on the collected time series (*e.g.,* Around 30 ms increased delays at 11:56 and 13:56 of monitor node `196.49.14.12`). There are more fluctuations in the uncommon paths of monitor node `143.129.80.134` and some of them are aggressive (*i.e.,* around $13:58$ with over 100 ms delay increased), which contributes to more wavelet energy than the monitor node `196.49.14.12` at small scales (*e.g.,* scale 1) resulted from those rapid changes.

We further perform the estimation on two leaf tree topology of the monitor node $10.42.4.201$ to show how the correlation among links might affect the accuracy of our estimator and the results are shown in figure 2.11. The two uncommon paths estimation align with the gray diagonal line with relative small errors. The estimation on the common path in the figure 2.11a, however, are not accurate and errors exceed 10, which is far larger than the previous estimation error captured within 1. As we can see from the time series associated with each path in figure 2.11(d)-(e), the common path has small fluctuations within 7.5 ms but the left and right uncommon paths have large spikes (*e.g.,* over 200ms at around 13:53) and correlation coefficient around 0.33. Most of the bias of the wavelet energy estimation on the common path would be contributed by $\mathbb{E}[X^{(1)} \cdot BX^{(2)}]$ where $X^{(1)}$ and $X^{(2)}$ are the left and right uncommon paths. Although large errors are observed on the common path, we have accurate estimates over uncommon paths due to more aggressive fluctuations and the wavelet energy of uncommon paths is far larger than the common path.

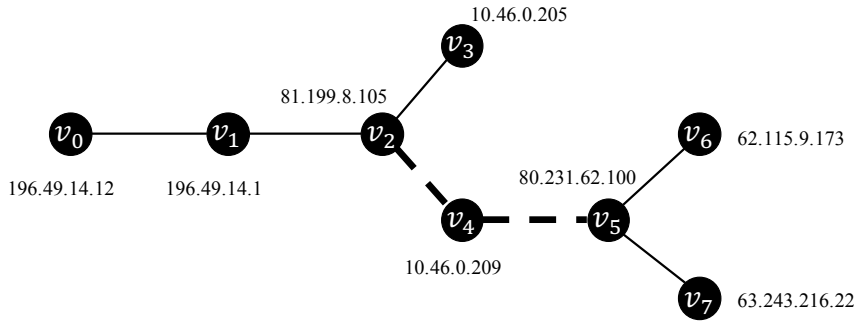


Figure 2.12: Subtopology of monitor node at 196.49.14.12

2.6.3.2 General Topology

We extracted a more general topology from Ark measurements at monitor node `196.49.14.12` as shown in figure 2.12. We regard this as a tomography problem by assuming the only observed RTT measurements are from v_0 to v_j ($j \in \{3, 6, 7\}$), and our goal is trying to estimate the internal links or paths wavelet energy based on those observed measurements. However, the end points v_6 and v_7 are far from the monitor nodes, which results the total number of available RTT measurements is 347 ($\sim 2^8$) on *2019-01-01* after time alignment. The topology in figure 2.12 essentially contains two common paths (*i.e.,* v_0 to v_2 and v_0 to v_5). The estimation of the common path wavelet energy can be performed by selecting any two leaf nodes shared the common path as proved in equation 2.18 of section 2.3 and we will use leaf node pairs of $\{v_3, v_6\}$, $\{v_6, v_7\}$ for the two common paths estimation. Additionally, the internal path (v_2 to v_5) can be estimated by the difference between the wavelet energy of two common paths.

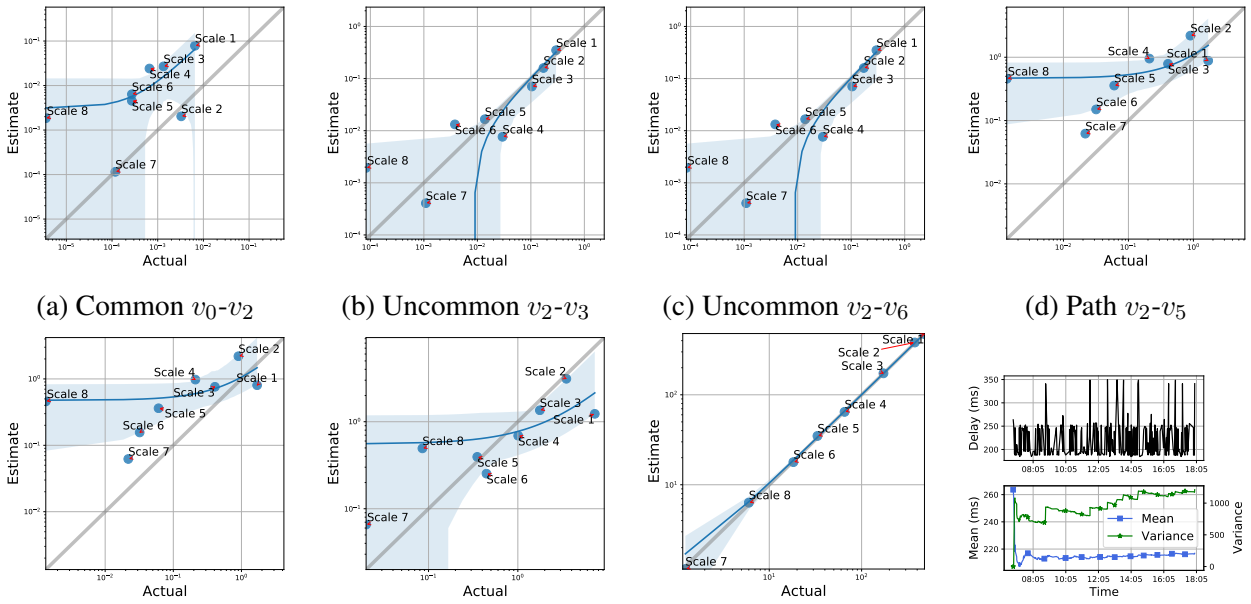
Figure 2.13 shows the estimation of the general topology. The common path from v_0 to v_2 has very small wavelet energy across all scale as shown in figure 2.13a and both the estimate and the actual values are very small (*i.e.,* within 0.025), which indicates a very small variance for the RTT measurements on the common path. Figure 2.13b is the uncommon path (*i.e.,* v_2 to v_3) estimation and it also represents a small variance RTT collection. We also estimate the same link (*i.e.,* v_2 to v_3) in the previous two leaf tree topology but with higher estimates and larger variance in figure 2.10d, which is because we filtered down more measurements and the time series are not the same

for the two topologies. Both measurements collected at common path from node v_0 to node v_5 and the uncommon path from v_5 to v_6 have very small fluctuations as we can see from the figure 2.13e and 2.13f with minor errors on estimation. However, large wavelet energy ($\sim$ 500) was identified on link v_5-v_7 as shown in figure 2.13g. The small deviations in figures 2.13e and 2.13f could be caused by the large fluctuations in the RTT measurements at link $v_5 - v_7$, which *leaked* energy when computing wavelet products, and taking average of RTT measurements as the true common path RTT measurements is another reason. Figure 2.13h shows the RTT measurements on the uncommon path $v_5 - v_7$, we noticed that the time series fluctuated between 200 to 250 ms with large rapid change about 150 ms (*e.g.,* time at 11:40am, 18:00pm, *etc.*) and the variance is not stable over time with above 1000 at the end. Compared with the time series collected in the two leaf tree mode and the estimation on the right uncommon path in the figure 2.10d, link $v_5 - v_7$ has larger wavelet energy as a result of more fluctuations (*i.e.,* larger variance). Figure 2.13d gives the estimation on the path $v_2 - v_5$ as the difference of wavelet energy of the two common paths (*i.e.,* $v_2 - v_6$, $v_0 - v_2$) with minor errors, which also indicates a small variation connection path wavelet energy. Overall, the link (v_5-v_7) with large fluctuation contributed to the dominant wavelet energy on the topology is localized by the estimator.

2.6.3.3 Analysis of Estimation Errors and Prevention Techniques

We observed small errors of the estimation over subtopologies from the realistic data sets provided by CAIDA Ark [33] and we summarized those reasons on introducing estimation error with simple two leaf tree topology (*i.e.,* figure 2.1), which can be generalized to larger topology. Let $X_1^{(0)}$ and $X_2^{(0)}$ denote the time series collected by probing packets with destination v_2 and v_3 and let $Y^{(1)} = X_1^{(0)} + X^{(1)}$, $Y^{(2)} = X_2^{(0)} + X^{(2)}$ be the path aggregation at leaf nodes.

– The RTT measurements taken by the Ark platform on the common path did not consider about the interpacket time of packets to different leaf nodes. We observed nearly uncorrelated time series of $X_1^{(0)}$ and $X_2^{(0)}$, which results $\mathbb{E}[\hat{F}(X^{(0)})] = \mathbb{E}[F(X_1^{(0)} \cdot BX_2^{(0)})] = 0 \neq \mathbb{E}[F(X^{(0)})]$ if $X_1^{(0)}$ and $X_2^{(0)}$ are independent and assume $\mathbb{E}[\Delta] = 0$.



(a) Common v_0-v_2 (b) Uncommon v_2-v_3 (c) Uncommon v_2-v_6 (d) Path v_2-v_5

(e) Common v_0-v_5 (f) Uncommon v_5-v_6 (g) Uncommon v_5-v_7 (h) Uncommon v_5-v_7

Figure 2.13: Estimation of the common paths and uncommon paths for the extended subtopology

– More than one nonstationary signals on the two leaf tree topology might introduce errors. Lemma 6 states the unbiased estimation with no more than one nonstationary signal. However, two nonstationary signals $X^{(i)}$ and $X^{(j)}$ generally give $\mathbb{E}[X^{(i)} \cdot BX^{(j)}] \neq 0$ $(i \neq j)$, then $\mathbb{E}[\Delta] \neq 0$.

– Correlations among links. Generally we would expect time series of two links exhibit very small correlation. However, significant errors would be introduced if two signals are correlated with large magnitude. For example, figure 2.11a has errors on the common path estimation as the result of correlation between uncommon paths and nearly independent measurements on the common path, but the uncommon paths estimation of figure 2.11b and figure 2.11c diminished the impact by having large fluctuations in the signals, which provides a good estimation over uncommon paths.

To reduce the estimation errors, we can send probe packets to different destinations with inter-packet time close to zero at each time window to maximize the correlation of measurements on the common path. Another technique is to have a finer granularity and consistent measurements. Currently the realistic trace has 1 second granularity and missing measurements at some time win-

dow as we described in 2.6.1.3. By having finer granularity and consistent measurements may potentially help to reduce the correlation among different links.

Table 2.2: January 2019 probing statistics

Monitor nodes	**Selected leaf nodes**	**Total probings**	**probings/cycle**
acc-gh (196.49.14.12)	10.46.0.205, 10.46.0.209	5,262,000	92,315
cjj-kr (150.183.95.135)	134.75.23.1, 134.75.23.9	5,851,365	86,049
dtw2-us (198.108.63.13)	12.123.159.50, 12.123.159.54	6,268,230	90,844
eug-us (128.223.157.8)	10.252.9.13, 10.252.10.13	5,731,500	84,287
yhu-ca (96.127.255.96)	62.115.134.52, 62.115.137.142	5,269,095	94,091
pbh2-bt (10.10.10.68)	103.80.109.65, 103.80.109.1	1,135,500	70,969
sao-br (200.160.7.159)	154.54.11.1, 63.223.54.30	5,235,000	93,482
tij-mx (199.48.225.18)	38.140.128.49, 69.174.12.97	5,361,000	95732
wbu-us (192.43.244.202)	38.140.128.49, 69.174.12.97	4,801,500	92,337

2.6.4 Ark Performance

2.6.4.1 Dataset.

We extracted subtopologies from 9 different monitor nodes in different locations across the world and each of them can be converted to the virtual two-leaf tree topology. For a topology with multiple destinations (*i.e.,* the number of leaf nodes ≥ 3), two or more smallest unit (two-leaf tree) can be constructed by selecting any two destinations (Section 2.3). For each selected monitor node, we evaluated 1 month RTT measurements with statistics shown in table 2.2. The wavelet energy estimation for each monitor node is conducted for every probing cycle during this month.

2.6.4.2 Mean relative error.

Figure 2.14 shows the mean relative error (MRE) of estimating the wavelet energy across all scales on the link with largest total wavelet energy, which is the sum of wavelet energy across all

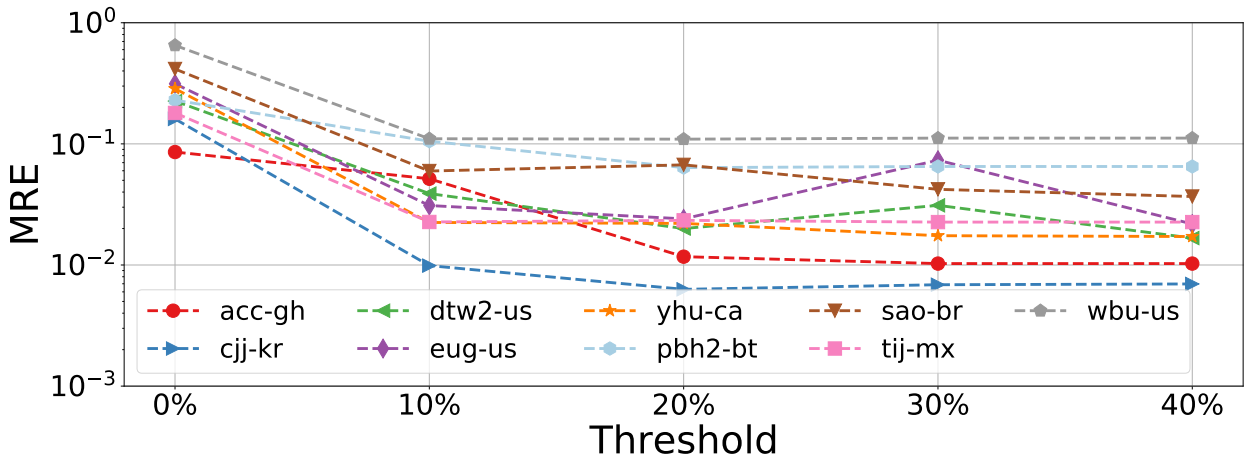


Figure 2.14: MRE of selected monitor nodes

scales. The MRE is calculated by

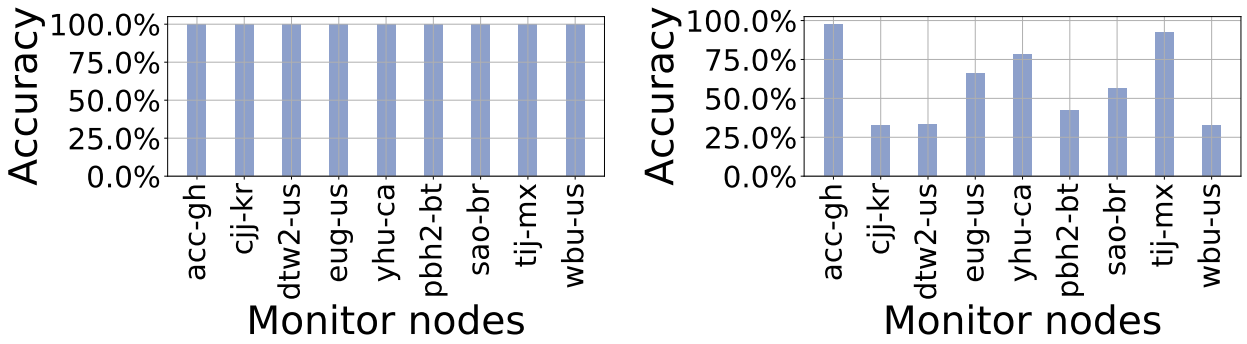
$$\frac{1}{M}\sum_{m}\frac{1}{C}\sum_{c=i}^{C}\frac{|\hat{F}_m^c(X)-F_m^c(X)|}{F_m^c(X)} \tag{2.22}$$

where $\hat{F}_m^c(X)$ represents the estimates in cth probing cycle and M, C are the largest number of scales observed and the number of probing cycles taken in this month respectively.

We further applied a filter over the energy of each scale and only calculated MRE of scales with energy larger than a certain percentage of total estimated energy (*i.e.,* threshold in the x-axis of figure 2.14), and we stopped at 40% of total energy as only 1 scale left for all monitor nodes. The MREs are small (≤ 0.1) over the selected monitor nodes especially for scales with larger energy ($\geq 10\%$ of total energy).

2.6.4.3 Accuracy.

Figure 2.15 shows the accuracy on correctly ranking links according to their total wavelet energy. The accuracy is calculated by $1-\frac{\#.\ false}{\#.\ links}$. We observed errors over several monitor nodes such as *cjj-kr* on ranking all links in figure 2.15 (b), which results from these links with similar energy and estimates over these links fluctuate around true energy. However, the energy estimation errors are small for network links as shown in the next Section (2.6.4.4) with analysis on the amount



(a) Link with highest energy

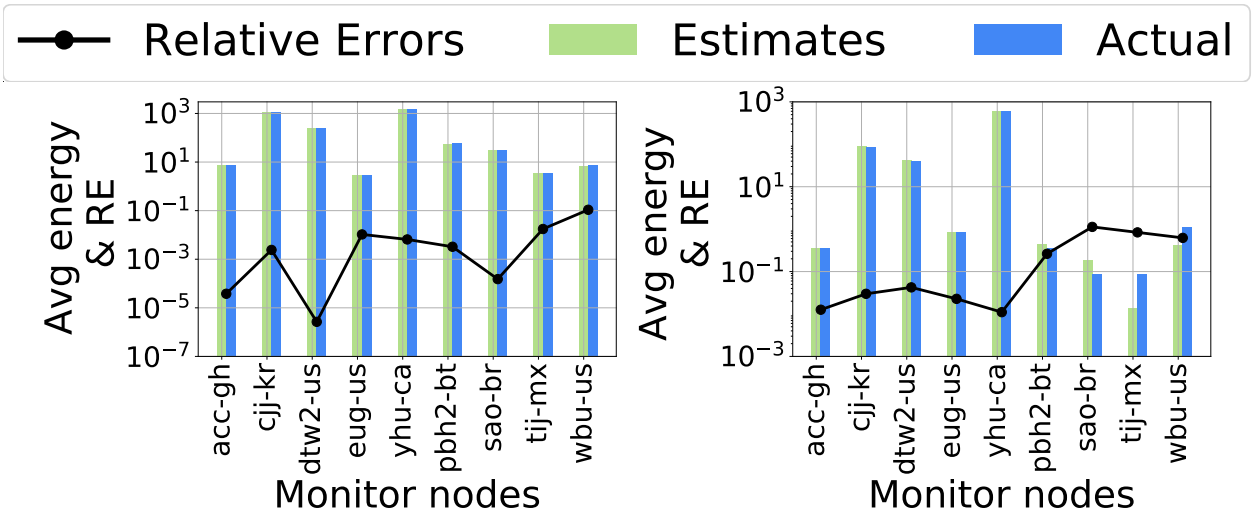
(b) Rank all links by energy

Figure 2.15: Accuracy.

of energy distributed over different links. For identifying the link with largest wavelet energy as shown in figure 2.15 (a), our estimator achieved 100% accuracy over all monitor nodes.

2.6.4.4 Average wavelet energy.

Figure 2.16 shows the average energy per probing cycle on the link with highest energy and all other links of both estimates (green) and actual (blue) with relative errors (black). The average energy is calculated by the total normalized wavelet energy of all probing cycles (*i.e.,* $\sum \frac{1}{T}\hat{F}(X)$) divided by the number of probing cycles. As we can see from the figure 2.16 (a), all estimates are accurate with small relative error (as interpreted by Section 2.6.4.2 also). We aggregated the energy over all other links except the link with highest energy, and the results are also accurate as shown in figure 2.16 (b) with slightly larger relative errors in cities such as *sao-br* and *tij-mx* but with small absolute errors. Additionally, we also observed that most of the energy is concentrated on scale 1, which is the scale with high frequencies. Unlike the periodical patterns generated by the low rate anomalies (Section 2.5), which have larger energy over lower frequencies, those high energy observed from those monitor nodes is caused by suddenly increased latency (non-persistent over time).



(a) Link with highest energy

(b) All other links

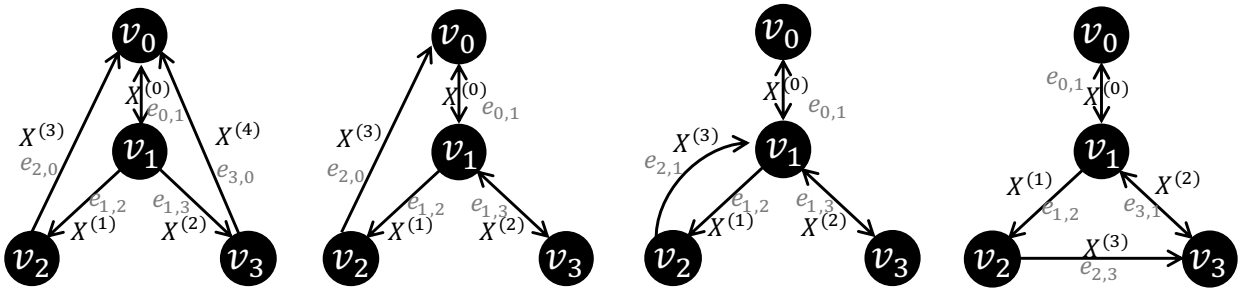
Figure 2.16: Average energy per probing cycle

2.7 Asymmetric Routing

We assumed that the RTT measurements on previous section based on the symmetric routing between source and destination. However, asymmetric routing is possible for the communication between two network nodes to achieve optimized and efficient network operations. In this section, we shall analysis the possible asymmetric routing schemes and address limitations of our estimator on RTT measurements compared with end-to-end (*i.e.,* one way delay (OWD)) measurements. We illustrate asymmetric routing on a simple two leaf tree model as it could scale to a large network topology and assume the time series collected on each link satisfied the constrains of unbiased estimator for both stationary and nonstationary scenarios. Figure 2.17 shows potential asymmetric routing schemes. Let $X^{(i)}$ denote the OWD measurements taken on each link and let $Y^{(1)}$ and $Y^{(2)}$ denote the RTT measurements from root node to the two destinations.

In the first scenario asymmetric routing occurs on both paths from root node as shown in figure 2.17a. Then $Y^{(1)} = X^{(0)} + X^{(1)} + X^{(3)}$ and $Y^{(1)} = X^{(0)} + X^{(2)} + X^{(4)}$, and

$$\hat{F}(X^{(0)}) = X^{(0)} \cdot BX^{(0)} + \sum_{a\in\{1,3\},b\in\{2,4\}} X^{(a)} \cdot BX^{(b)} \tag{2.23}$$



(a) Case 1 (b) Case 2 (c) Case 3 (d) Case 4

Figure 2.17: Asymmetric scenarios on the simple two leaf tree model.

By having asymmetric routes on root to two destinations, the RTT measurements $Y^{(j)}$ ($j \in \{1,2\}$) only contains one way measurements on the common path from v_0 to v_1. According to theorem 1, we have $\mathbb{E}[\sum_{a\in\{1,3\},b\in\{2,4\}} X^{(a)} \cdot BX^{(b)}] = 0$ and $\mathbb{E}[\hat{F}(X^{(0)})] = \mathbb{E}[X^{(0)} \cdot BX^{(0)}] = \mathbb{E}[F(X^{(0)})]$. $\hat{F}(X^{(0)})$ estimates the OWD of the common path from the RTT measurements of Y. However, the estimation is slightly different when only one of the path from root node to destinations is asymmetric as shown in figure 2.17b. With the only two observed path measurements $Y^{(1)}$ and $Y^{(2)}$, we have $Y^{(1)} = X^{(0)} + X^{(1)} + X^{(3)}$ and $Y^{(1)} = 2X^{(0)} + 2X^{(2)}$ if we assume the OWD measurements taken for each direction on a path is the same, and our estimator based on the path observations is

$$\hat{F}(X^{(0)}) = \frac{1}{4}\left(F(Y^{(1)} + Y^{(2)}) - F(Y^{(1)}) - F(Y^{(2)})\right) = X^{(0)} \cdot BX^{(0)} + \frac{1}{2} \sum_{a\in\{1,3\},b\in\{2\}} X^{(a)} \cdot BX^{(b)} \tag{2.24}$$

The estimator still estimates the wavelet energy of OWD on the common path based on the RTT measurement taken from the root to the destinations but with extra coefficient $\frac{1}{2}$.

Figure 2.17c illustrate the scenario when the uncommon path has asymmetric routing. This is similar to the symmetric routing where they both experienced RTT measurements on the common path and the only difference is that wavelet energy estimation of the uncommon path (*i.e.,* $(X^{(1)} + X^{(3)}) \cdot B(X^{(1)} + X^{(3)})$ for the asymmetric scenario). Figure 2.17d shows asymmetric routing when RTT time measurements for both destinations overlap. Specifically, we have

$Y^{(1)} = 2X^{(0)} + X^{(1)} + X^{(2)} + X^{(3)}$ and $Y^{(2)} = 2X^{(0)} + 2X^{(2)}$. The common path experienced by the two destinations is changed to $e_{0,1} + e_{1,3}$ with time series $X^{(0)}$ and $X^{(2)}$. This asymmetric routes shows the limitations on estimating wavelet energy of the original common path $e_{0,1}$ since $\frac{1}{2}\mathbb{E}[\big(F(Y^{(1)} + Y^{(2)}) - F(Y^{(1)}) - F(Y^{(2)})\big)] = 4\mathbb{E}[X^{(0)} \cdot BX^{(0)}] + 2\mathbb{E}[X^{(2)} \cdot BX^{(2)}]$, which contains not only the wavelet energy of path $e_{0,1}$ but also the path $e_{1,3}$. However, the expectation of wavelet energy of the measurements to destination v_2 is $\mathbb{E}[Y^{(2)}] = 4\mathbb{E}[X^{(0)} \cdot BX^{(0)}] + 4\mathbb{E}[X^{(2)} \cdot BX^{(2)}]$ by the independence assumption. Therefore, our estimator on the common path can be described as

$$\hat{F}(X^{(0)}) = \frac{1}{4}\left(F(Y^{(1)} + Y^{(2)}) - F(Y^{(1)}) - 2F(Y^{(2)})\right) \tag{2.25}$$

which is similar to the equation (2.24) but with different wavelet products. More asymmetric schemes can be constructed by combining the discussed the scenarios and use the similar methods to conduct the estimator on the common path. Besides the illustration over the simple two leaf tree topology with asymmetric routes, which might be complex on estimating the common path or the uncommon path wavelet energy, we can also use a larger topology to recover wavelet energy on internal paths as illustrated in section 2.3 and experimental evaluation in section 2.6.3.2.

2.8 Related Work

2.8.1 Network Internal Link Statistics Measurements

Common router based measurements techniques like SNMP[1], RMON[2], NetFlow[3] directly access routers to retrieve information about network states by answering infrequent polling requests and consume available network bandwidth for transmitting data. However, increasing limitations over those approaches has been observed for more complex and faster networks. Recently proposed In-band Network Telemetry (INT) provides a way of monitoring network internal state independent from the control plane by having the measurements data embedded inside the network traffic and proposed monitoring techniques [7, 9, 11, 12, 13, 37] focus on handling the processing and collection mechanisms of the INT packets. However, this requires an INT capable switches being installed over a network and the amount of information carried on network packets

is limited by the *maximum transmission unit* (MTU), which also consumes network bandwidth. To overcome those limitations, network tomography approach provides a way on reconstructing the network internal performance by analyzing the end-to-end measurements[29].

2.8.2 Prior Approaches to Time Series Tomography

Much work in network tomography has focused on overcoming the limitations of underconstrained linear systems such as (2.1) that arise from network measurement. Several previous works have focused specifically on the problem of inferring properties of unobserved network times time series $\{X_e\}$. [38] considered the related problem of time-varying traffic-matrix tomography, introducing power-law constraints between signal mean and variance in order to render the model identifiable from observations of these variables. While a modified EM approach was used to find maximum likelihood solution, its computational complexity was prohibitive for real networks. Performance tomography from multicast probing is another approach to rendering an edge metric model identifiable by introducing constraints between path measurements [39, 40, 41, 42]. This approach has been extended to estimate the frequencies of temporal subsequences of internal link packet loss [43] and latency [44]. Sparsity conditions have been applied as a constraint through a regularization in order to select "simpler" explanation of observed path time series [45, 46]. A computationally simpler approach has been to infer summary link summary statistics in a parameter-free way, in particular, the variance of internal link metric time series. This approach is based on the observation that for additive link metrics that are mutually independent, the covariance of metrics along two intersecting paths is equal to the variance of the aggregate metric over their path intersection; see [29]. Coupled with empirical models relating the relative ordering of metric variances and averages on path, this approach has been used to identity worst performing links and reduce dimension of the linear system (2.1) to the point that a restricted form can be solved for these links [47]. The conditions under which an unknown topology can be identified from covariance metrics has recently been established in [30].

2.8.3 Wavelet-based Anomaly Detection

Wavelet coefficients across different scales computed the difference on the local averages and wavelet energy on the certain scales gives valuable statistics on characterizing changes at different frequencies, which may signal the presence of abnormal network activities or attacks. By decomposing signals into different frequencies (*i.e.,* scales), wavelet analysis are good at capturing network abnormal behavior caused by anomalies (*e.g.,* [48, 49, 50, 51, 52, 53, 54, 55, 56]). Feldmann used a wavelet-based detection mechanism on finding network anomalies and also illustrated the relationship between the wavelet energy and the RTT. Hussain [57] proposed a frame work using spectral properties to classifying DoS attacks. Huang [20] characterize the deviation of wavelet coefficients with several different mother wavelet on detecting different network attacks. Dainotti [21] designed a two-stage DoS attack detection use wavelet analysis on capturing the change points. Barford [58] provided a detailed signal analysis on traffic anomalies by grouping scales using wavelet decomposition to show wavelet analysis is effective on capturing ambient and attack traffic.

The low rate DoS attack, on the other hand, sends small burst of traffic periodically to occupy network resources and lower the rate of normal TCP traffic based on periodic interference with TCP retransmission timers [25], which modeled the low rate DoS attack as periodic pulses represented traffic bursts. Many approaches (e.g [59, 60, 61, 62]) on detecting low rate DoS attack are focusing on exploring the spectral properties of traffic signals to extract information in the frequency domain. Luo [59] proposed a mechanism on combining DWT and CUSUM for characterizing the fluctuated incoming traffic and Yue [62] use wavelet energy spectral in addition to neural network to detect the low rate anomalies. While some works have focused on identifying spectral properties from end-to-end measurements [23], as far as we are aware these have not been detected by tomographiy.

3. INT MONITORING PLATFORM*

3.1 Overview

Network monitoring provides information crucial for the control, operation and management of computer networks. Over the last few years, In-band Network Telemetry (INT) [63] has represented an important development in network monitoring that supports collecting and reporting network state in the data plane. INT enables traffic source to embed telemetry instructions in data packets. This capability will improve on long-standing practices of management based monitoring via relatively infrequent SNMP-based polling or event detection via RMON [64], or performance measurement using probe packets, by providing timely and precise measurements from user packets in the network data plane. In addition to enabling detailed and granular monitoring of network performance, the information provided through INT can potentially improve congestion control and alert to the occurrence of complex events defined by the cumulative information of a packets traversal of network elements.

The advent of programmable switches has been a significant development, both for enabling acquisition of information from packets, and to enable both control *and* data paths to adapt the dynamic conditions in the network informed through monitoring. In particular, the P4 [65] language provides a framework with which to program switches to process packet flows more flexibly with a richer match/action functionality than was previously possible. In the context of INT, P4 is able to leverage recent advances in the compute capabilities of the networking devices such as switches, routers and end host network interface cards (NICs) to enhance monitoring capability. Specifically, additional packet fields can be introduced for each switch (we use the generic term, switch, to refer to both layer-2 switches and layer 3 routers) in the network to provide monitoring information. These fields are embedded within the data packets and switches are able to use them to insert granular telemetry information concerning the performance and functioning of the net-

*©2020 IEEE. REPRINTED, WITH PERMISSION, FROM YIXIAO FENG, A SMARTNIC-ACCELERATED MONITORING PLATFORM FOR IN-BAND NETWORK TELEMETRY, IEEE WORKSHOP ON LOCAL AND METROPOLITAN AREA NETWORKS (LANMAN), JULY/2020.

work switches and links in the data path. Thus, to provide the most actionable information to an end system requires processing all INT packet fields populated by switches in a packet's path and providing that information in a timely manner can be a significant challenge.

With the need for ever higher link bandwidths in datacenters (DC), servers increasingly depend on smartNICs (sNIC) to offload network packet processing* because of the performance boost they can provide. The main challenge that our work addresses is *how to design a cost-efficient traffic measurement and analysis infrastructure that can act as a monitoring sink platform for In-Network Telemetry (INT) to provide loss-free and timely INT notification of complex events dependent on entire packet paths to the monitoring host by leveraging the sNIC.*

Our work addresses a gap in current approaches, which do not achieve scalable cost-efficient INT packet processing at the monitoring sink. Specifically, most of the existing solutions [66, 15, 67] rely on the host-CPU to perform INT packet processing and event detection and typically are unable to achieve high-performance (line-rate) INT event processing.

In this work, we present the architecture, design and implementation of a high-performance INT processing and INT event detection framework, in which we partition INT processing between the sNIC and the monitoring host. The key advantage of this architecture, compared to existing solutions, is the ability of the sNIC to provide key processing functions – transformation and application level steering – that are much more difficult to support in a programmable switch, while relieving the host CPU of the requirement to process packets in software at full line rate. We illustrate the scheme to collect and aggregate INT information across the packets of different network flows (*i.e.,* network flows identified by unique IP 5-tuple) within the sNIC and export the INT metrics to the monitoring host. This allows us to improve packet processing rates by a factor of around 3 compared with rates reported in the literature [67] by having the sNIC relieve the packet processing demands on the host CPU. Furthermore, we propose mechanisms to program the INT event parameters on the monitoring framework to have the sNIC detect and export INT events to

*SmartNICs are the programmable and extensible network interface cards (NICs) that provide the network traffic processing capabilities within the NIC, allowing the CPU to program and offload certain packet processing to be performed within the NIC

the monitoring host in a timely manner. The host can then process these events to take necessary data/control plane measures, readjust/reprogram the INT event thresholds on the sNIC, and also timestamp and store the event information (time-series data) in externalized database (Redis).

Our work incorporates the most recent advances and specifications of INT v2.0 [68]. Our frameworks support all INT metrics (packet path information: switch id, hop latency, queue occupancy, congestion *etc.*) listed in the current INT specification and is also extensible to support additional metrics as necessary. Our approach not only enables us to achieve line-rate packet processing, but also drastically reduces the host CPU consumption for INT-related processing.

To summarize, the key contributions of our work include:

- **Flexible partitioning and INT processing offload**: The INT monitoring functionality is judiciously split between the sNIC and host to cooperatively process the INT packets and INT events at full line rate. With a large cohort of much less powerful microengines (MEs), the sNIC performs INT packet processing in-path at line rate (with the MEs performing the processing in parallel) to extract the key flow information, while intelligently avoiding slowdowns from locks used for coordination;

- **INT data and Event Aggregation**: The sNIC processes each flow to collect and aggregate INT information across different packets in a hash table and also transforms the INT packet data to INT events and notifies the events in a timely manner to the monitoring host;

- **Adaptable and Mutable INT data and event support**: Our platform provides a data structure to facilitate platform specific INT data collection that is adaptable to custom requirements and allows for filtering the INT events based on configured thresholds, while allowing us to mutate threshold parameters dynamically based on the INT event processing rate and characteristics of the monitoring platform.

3.2 Why sNIC for Efficient Monitoring?

To demonstrate the limitation of a pure host-based platform to perform lossless line-rate monitoring even in current-day high-performance systems utilizing the DPDK libraries (e.g., [69]),

we implemented only the volumetric analysis (tracking the packet count for each packet flow) using a conventional 10 Gbps Intel Ethernet network interface card. We used the publicly available CAIDA Traces [70] from year 2015 to 2019 each containing about 4.8-11.5 million distinct flows and about 100-200 millions packets. Fig 3.1 shows the loss-free packet processing throughput

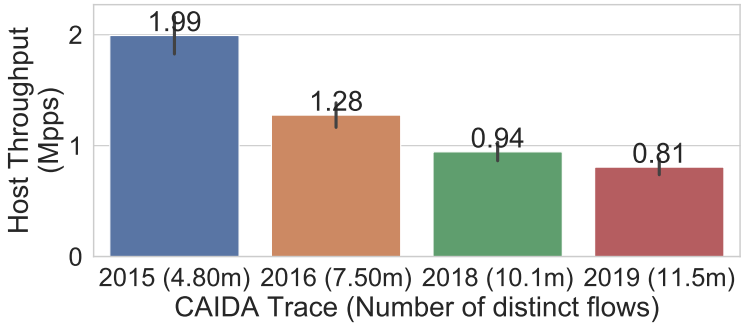


Figure 3.1: Volumetric analysis using Host CPUs.

sustainable when performing just the volumetric analysis on a system with Intel Xeon 2.2 Ghz CPUs using a two CPU processing cores (one core for packet Rx/Tx processing using DPDK and another for updating a hash table maintaining counts). We observe that the throughput with a host based implementation is quite low ranging from (0.8 - 2Mpps). Further, we observe that the throughput depends on the traffic mix: having a relatively small number of flows does achieve higher throughput, while increasing the number of distinct flows results in throughput degradation because of the increased hash collisions. Even though scaling the number of CPU processing cores may attain higher throughput, this approach is expensive (in terms of CPU cores). Moreover, using multiple CPU cores would require additional explicit synchronization techniques across the CPU cores, which would then make the overall host processing even more complex.

The sNIC we use has a large number of PMEs that can process incoming packets in parallel through the P4 pipeline. This, by design enables better handling of the hash collisions, as the PME that encounters the collision only needs to take care of that entry (as long as there is no lock, and there are a large enough number of other flows that can be processed in the meantime), while the

rest of the PMEs that do not suffer hash collision can process other packets in parallel.

In this work, we used the Netronome SmartNIC [71] for offloading critical INT processing functions. In addition to network processing capabilities of traditional NICs, *e.g.,* checksum, segmentation and reassembly, the sNIC also has 60 flow processing cores (that run at 633MHz), also known as Microengines (MEs), for implementing the more complex network data plane functionality. This may include encryption/decryption, flow table (match-action) processing, traffic shaping, security and traffic analysis functions, *etc.* The Netronome sNIC supports P4 for programmable parsing (header extraction), which is conducted by a subset of MEs, known as the packet processing cores. We dedicate 54 MEs for packet processing pipeline and INT header processing. A global load balancer on the sNIC distributes the incoming packets among these 54 PMEs based on a credit-based scheme to provide considerable parallelism for high throughput packet processing. The rest of the MEs run dedicated MicroC programs *i.e.,* separate functions executed asynchronously to enable stateful packet processing within the sNIC. (e.g., stateful event processing, micro-burst detection, *etc.*).

The sNIC has a hierarchical memory sub-system, providing a large external memory $2GB$, which we leverage to please edit as appropriate. implement custom data structures (user configured) for caching flow specific INT data (called a FlowCache) for several flows, while aggregating INT metrics across packets of a flow. We also use the memory to store the P4/SDN match action tables.

3.3 INT and telemetry report specification

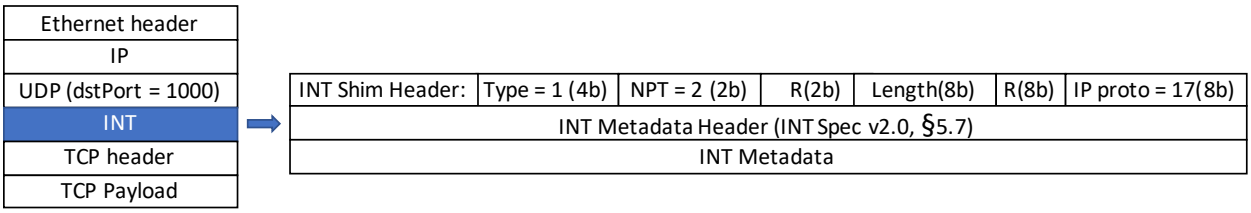


Figure 3.2: INT over TCP packets

INT [68] provides real-time, fine-grained, end-to-end network monitoring in the dataplane [14]. Our platform currently supports INT processing and monitoring for the INT-MD type where the INT data is carried along with the packet.

INT Header format and location: The current specification [68] has multiple ways to place the INT header and metadata fields. To allow support for INT even with SSL *i.e.,* encrypted TCP data, we choose the INT with new UDP header encapsulation approach for TCP packets where we insert a new UDP header, INT header and INT data in between the original Layer-3 (IP) and Layer-4 (TCP) headers. However, for UDP packets we leverage the original UDP header, as per the specification. For both cases, we set the destination port of the outer UDP header to INT_PORT (0x1000) to indicate the presence of INT data. Although this approach requires the sink node to perform different processing for TCP vs. UDP packets (which we feel is not ideal), we seek to be consistent with the current specification. The primary motivation for the different treatment for UDP packets in the specification appears to be to try and minimize the overhead for inserting the INT headers with UDP packets.

Fig. 3.2 shows the INT shim header, which is 4 bytes long, followed by the INT metadata header of 12 bytes. After the INT shim and metadata header, each INT hop adds the same length of metadata[68] as set in the metadata header. In our experiment settings, every packet traverses a fixed path of 5 INT intermediate switches.

INT Event and Telemetry Reporting At the INT sink, the INT metadata is extracted and telemetry reports are generated and communicated to the monitoring host based on the guidelines detailed in P4 telemetry report specification[68]. Further, we incorporate additional mechanisms to distinctly report the aggregate information for the regular INT telemetry data, and to provide specific INT event notifications along with the associated INT data to the monitoring host.

3.4 Design Considerations on SmartNIC and Host

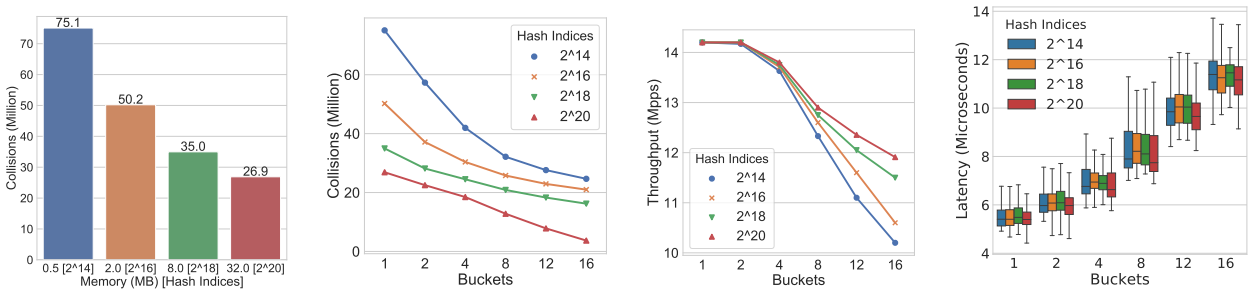
3.4.1 Data structure and operations

We build a hierarchical Flowcache data structure across the sNIC and the host. The sNIC Flowcache is updated in-line with the packet processing on sNIC, while the host updates its Flowcache based on the evictions and periodic snapshot from the sNIC.

Rationale Compared to the host, the sNIC provides low-latency and high throughput in the packet processing path, due to considerable parallelism. Hence, it is desirable to process and extract per-packet information within the sNIC. But, the sNIC has limited memory and off-path compute capabilities. It can at best support a hash table with a few million entries. Further, the limited compute capability bounds how frequently we can coalesce and transfer the large data structure to the host. Thus, *correctly sizing the hash table in sNIC is vital to achieve low-latency and high throughput.* Studying datacenter traffic traces, we observe more than a million flows per second [72, 70]. So, hash collisions are inevitable. *We need effective policies and mechanisms to mitigate potential collisions and associated processing overheads.*

Sizing: We empirically determine a reasonable size for the sNIC hash table. We use publicly available CAIDA traces (2015, 2016, 2018, and 2019) [70] and resize the packets to 64 bytes, and they replay at the full 10Gbps line rate (14.2Mpps) using MoonGen [73]. We start from a small memory footprint and increase it to the largest possible size on the sNIC. Fig. 3.3a shows the number of collisions observed with different hash table sizes for the CAIDA 2018 trace (our observations are similar with other traces). While a single bucket, large hash table may help reduce collisions, it will still have a significant amount of hash collisions ($\sim$30M out of 350M packets).

Structuring: To address collisions and processing overhead, we use a limited, linear probing-based hash table with multiple entries (also called buckets) per hash index. Fig. 3.3b, figs. 3.3c & 3.3d show the impact on the number of collisions, throughput and latency due to additional linear probing on hash collisions respectively. Increasing the number of buckets (or bins) helps reduce the number of collisions. *Note: For the same memory footprint, a hash table with multiple linear*



(a) # of collisions, 1 bucket.

(b) # of collisions w/# buckets

(c) Thruput w/linear probing

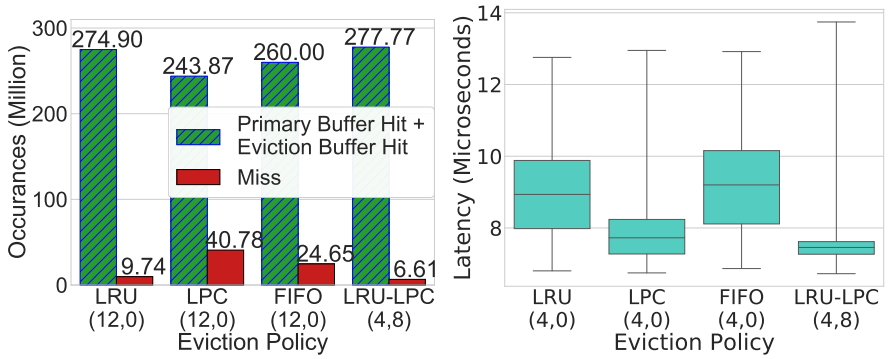
(d) Latency w/linear probing

Figure 3.3: Impact on Collisions & Throughput for different hash table configs. with CAIDA 2018 trace: (a) Collisions with increasing size of hash tables (single bucket). (b) Number of collisions for different hash table sizes, with linear probing. (c) Throughput due to linear probing. (d) Latency for linear probing. (NB: non-0 y-axis start)

probe entries show fewer collisions than a hash table with more indices. e.g., a hash table with 2MB memory footprint with 64K (2^{16}) indices and a single bucket has ~ 50.2M collisions, while 16K (2^{14}) indices with 4 buckets results in about ~ 41.95M collisions.

However, increasing the number of bins per hash index degrades both sNIC's throughput and per-packet latency. This indicates that to be loss-free, we need to handle collisions efficiently and consider traffic characteristics to alleviate the performance impact of collisions and the resulting evictions. We therefore restrict the linear probing depth to be at most 12 buckets and dynamically adapt to ensure the sNIC can maintain high throughput.

Partitioning & Eviction Policies: The key insights we obtain based on work reported on typical Internet data center (DC) traffic characteristics and our observations in this section analyzing the CAIDA packet traces is that i) even though a small fraction of large flows account for a majority of the packets, there are a large number of small flows that frequently compete for a hash entry and knock each other out from the table; and ii) typically packets of a given flow arrive in short bursts. We experiment with a number of widely used eviction policies, while keeping fixed the number of entries per hash index. Figure 3.4a and 3.4b shows their impact. Amongst Least Recently Used (LRU), Least Packet Count (LPC) and First-In-First-Out (FIFO) policies, the hit rate is highest with LRU. But, the mean and 75% latency is significantly lower for the LPC case. In order to effectively

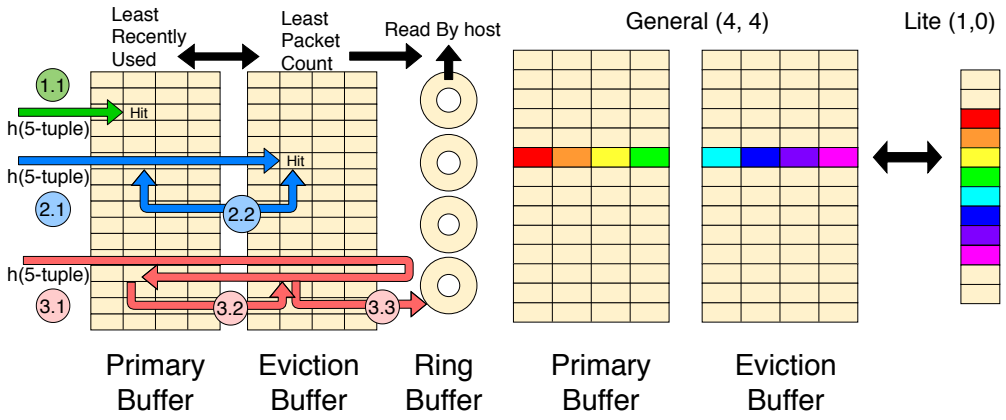


(a) Hit and Miss counts (b) Latency profile

Figure 3.4: Hash table eviction polices w/ CAIDA 2018 trace. (a) # hits & misses; (b) Latency; for different eviction policies (NB: non-0 y-axis start).

reap the benefits of LRU (*i.e.,* handle continuous train of packets from a flow) and at the same time benefit from the low latency with LPC (large number of hits coming from the large number of packets from a small set of flows), we devise a split, data structures namely the 'Primary' ($\boldsymbol{P}$) and 'Eviction' ($\boldsymbol{E}$) buffers to support both LRU and LPC eviction policies. On P, we use LRU eviction policy to retain the most recent flows among the initial hash entries, and minimize the linear probe latency. On a collision in P, an incoming packet would evict the oldest entry and move that evicted entry to the 'Eviction' (E) buffer. In E, we use LPC eviction policy to retain large flows (heavy hitters), that may not currently (in very short time scales) be seeing frequent packet updates. This LRU+LPC policy outperforms the other policies in terms of both the hit rate and latency (mean and 75%ile).

sNIC Data structure Update Operations: Figure 3.5a shows the organization and structuring of the sNIC data structure and the corresponding packet update operations on it. Each flow entry in P and E consists of the flow key (we hash the IPv4 or IPv6 5-tuple to identify entry), packet count, the latest timestamp (when this flow was last updated), the time window index (which monitoring interval this flow entry is currently at).Any packet processed by a PME can result in one of three potential outcomes: **1) $\boldsymbol{P}$ hit:** The packet's five tuple matches one of the bucket's five tuple in $\boldsymbol{P}$. Then, we increment the packet counter field and also update the timestamp for the matched



(a) Adaptive Data structure (b) General to lite mode

Figure 3.5: Flowcache structure and operations

entry. **2) E hit:** The packet's 5-tuple matches one of the bucket's five tuple in $\boldsymbol{E}$. In this case, we swap this entry with the oldest entry (LRU) in $\boldsymbol{P}$. **3) Miss:** The packet's 5-tuple does not match any entry in P or E. Then, we evict the LPC entry from E to make room for the new entry, and then swap that entry with the oldest entry in P. Increase in memory footprint results in higher throughput due to increased number of hits in $\boldsymbol{P}$ and $\boldsymbol{E}$.

The maximum width (number of buckets or entries) in $\boldsymbol{P}$ and $\boldsymbol{E}$ are configured at the compile time. Note: we use the notation (x, y) to designate the configuration of a row in the hash table with x buckets in $\boldsymbol{P}$ and y buckets in $\boldsymbol{E}$ respectively.

3.4.2 Host Support & Data Structures

Host Data Structures We dedicate a total of up to 14 (**8(Ring) + 1(redis)**) CPU cores for processing on the host side. We need to setup a minimum of 2 threads pinned to 2 distinct cores for R, with one to continuously process the sNIC exported flow records from the ring buffer R, and another for reading flow records evicted from the sNIC FlowCache. We use a CPU core to perform the data store update and snapshot of the host data structure to Redis [74].

The host also implements a hash table with 2^{24} rows having a single entry per row and dynamically probing linearly for a free entry. We leverage the sNIC to compute and populate the hash digest for each flow entry. The host explicitly computes the hash for a small fraction of packets

(received over the SR-IOV ports). In our experiments, we partition the hash table update across 8 CPU threads, with each thread operating on 2^{21} distinct row entries. Each CPU thread reads from a specific ring buffer R on the sNIC. The hashing on sNIC enforces the flow entry to be always mapped to one of the 8 rings and the threads on the CPU also end up updating a specific portion of the hash table on the host. This eliminates the need for locks among the host CPU threads. A dedicated REDIS thread continuously exports the flow entries of the hash table to the REDIS data store. This is done as a series of linked snapshots (one per measurement interval) of the hash table content pushed to storage for long-term forensics.

3.5 Design And Implementation

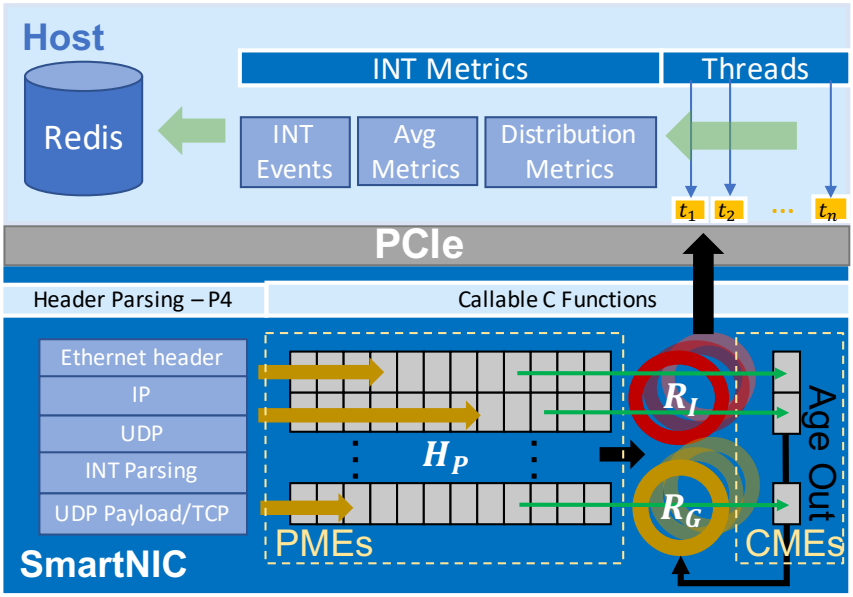


Figure 3.6: Architecture

3.5.1 Architecture and Data Structure

Fig. 3.6 presents the high-level architecture of our INT monitoring platform. While generally an INT sink is the 'last' switch and the monitoring node is a host connected over a link to that switch, we propose an architecture that utilizes a combination of a sNIC and host (commercially

available off-the-shelf server) to serve as combined INT sink and monitoring node. Packets flow through the monitor (which acts as a line-rate bump-in-the-wire) to the destination.

The packet processing pipeline in the sNIC takes advantage of P4 packet parsing, match-action rules and Micro-C algorithms to achieve fine-grained INT traffic analysis at line rate. The sNIC aggregates the INT metrics across multiple packets of a flow and collects the INT metrics for a large number ($\sim 6Million$) of flows in the INT Flowcache. We utilize the general-purpose Micro-C algorithms to perform more complex INT tasks (*e.g.,* averaging across INT messages per flow, event detection, and notification). We describe below the key components, data structure and algorithms that seek to achieve an accurate, timely, and close to loss-free INT monitoring platform.

3.5.2 sNIC Data Structure and operations

Fig. 3.6 also shows the data structure on sNIC and the corresponding operations for INT packet processing. We allocate a large hash table (a.k.a. INT FlowCache, H_P), with 2^{19} rows on the sNIC memory and 12 buckets per row to accommodate hash collisions. The IP 5-tuple of the incoming packet header is hashed to identify the row index in the hash table. Each bucket in the hash table includes a flow key (*i.e.,* 5-tuple), packet count, the timestamp of the most recent update and the INT metrics. The INT metrics consists of the most recent as well as the average statistics of the INT metadata (*i.e.,* switch ID, hop latency, queue occupancy, link utilization) for each INT transit node. We allocate 52 micro engines for the packet processing pipeline (PMEs) and dedicate 2 micro engines for custom processing (CMEs). We leverage the CMEs to age out the entries in H_P that have not been updated over a predefined time period. Based on the available credits at each of the PMEs, a global load balancer on the sNIC distributes incoming packets among these 52 PMEs. This allows all PMEs to process packets in parallel. (Each ME has 4 threads and each of these threads may process the packets concurrently). Hence, to guarantee consistency and accurate operations on primary hash table, each thread locks the corresponding row entry to update or evict to host.

Ring buffers: We leverage the ring buffers to export necessary INT information to the monitoring host. Each ring buffer is configured to hold 64K entries. We use a total of 8 general ring buffers

(R_G) to export an evicted bucket (INT entry of a flow) to the host (*i.e.,* upon collision and no free bucket in the row, we evict one bucket (least recently updated) to make space for a new flow.) We also use 8 INT event ring buffers (R_I) to export the INT events that are generated by the packet processing pipeline. The collected INT flow metrics would be inaccurate if the ring buffer overflows. We found that 64K entries for each ring buffer is sufficient space to prevent evictions from overflow on the ring buffer.

Packet Processing and key operations: Data update operations are described in sec 3.4.1. In either case of **hit** or **miss**, the PME thread will update the data structure and update the most recent as well as the average statistics of the INT metadata (*i.e.,* switch ID, hop latency, queue occupancy, link utilization) for each INT transit node. In addition, an INT data update may result in one or more INT event notification operations, as described below.

3.5.3 INT Data and Event Notification

Our INT Flowcache aggregates and collects INT information carried in each packet of the flow on the sNIC. For every packet, we extract the INT metrics (*i.e.,* the four metadata fields: switch ID, hop latency, queue occupancy, link utilization) embedded by each INT transit node. Further, when the incoming packet's INT data for a metric at any transit node exceeds a predefined threshold– configurable for each INT metric for each of the $\mathcal{T}$ and $\mathcal{C}$ events–an INT event (*i.e.,* event $\mathcal{T}$ and/or event $\mathcal{C}$ described below) is generated and notified to the monitoring host through the INT event ring buffer R_I. For each INT event, we store the corresponding INT fields (*i.e.,* switch ID, measurement value at that switch, bitmap indicating the event type and the timestamp) in the ring buffer R_I. In the flow entry in H_P, we also track the average for each of the metrics for each of the switches in the path.

We specify two broad classes of INT events: 1) Change events $\mathcal{C}$, and 2) Threshold-crossing events $\mathcal{T}$. Change events, $\mathcal{C}$, occur when the difference between the previous and the current INT metadata value (*i.e.,* hop latency, queue occupancy, or link utilization) exceed a predefined threshold. Threshold-crossing events, $\mathcal{T}$, occur when the current INT metadata value exceeds a predefined threshold.

We further categorize Event set for $\mathcal{C}$ and $\mathcal{T}$ into a) Per switch and b) End-to-end.

Per switch: Since we store the INT metadata value for each of the INT transit nodes (switches), we can account for i) the current INT metadata value that exceeds the predefined threshold; ii) the change in INT metadata value for these switches by comparing the previous and current reported values for the same flow. We generate an event when current absolute value or the difference between two values of the same metric exceeds a predefined threshold.

End-to-end The use of Micro-C allows us to also compute the aggregate (end-to-end measure) across all the INT transit node reported values, especially for the hop latency and also to build the path information by concatenating the IDs of each of the transit nodes. We generate an event when either the current aggregate value exceeds the threshold or the difference (previous and current) for the aggregated hop latency exceeds the predefined threshold.

Tracking Distribution for INT Metrics Each switch reports metrics in INT packets, and it is desirable to collect the distribution of the INT metadata by storing the occurrence for each possible value, or a range of values, over time. We allocate three arrays for the three INT measurements (*i.e.,* hop latency, queue occupancy, link utilization) per switch in the sNIC memory. Each entry in the array has a counter for a certain range of values (bin) of an INT metric, and the counter is incremented by 1 each time the observed metadata value corresponds to the bin's range. The distribution of each measurement per switch can provide important information on switch and path status (*e.g.,* determine the bottleneck by identifying the switch with the worst hop latency).

3.5.4 Host Data Structure and operations

The monitoring host reads the exported INT information including the flow statistics from sNIC. The INT notifications from the sNIC are also periodically flushed to the Redis database at the host. We dedicate a number of host threads (up to 10 CPU cores/threads) for reading from the sNIC rings R_G, R_I and the distribution arrays plus a CPU core to flush the INT metrics to the Redis [74] database.

3.6 Evaluation

3.6.1 Evaluation Setup

3.6.1.1 Testbed

We evaluate our monitoring platform on a Linux (kernel version 4.4.0-142) server with 10 Intel Xeon 2.20GHz CPU cores and 256GB memory and Netronome Agilio 4000 CX Dual-Port 10 Gigabit sNICs.

3.6.1.2 Evaluation Trace

We use a publicly available packet trace from CAIDA 2019 [70] containing about 186 millions packets over a 5 minute interval. We speed up the trace by reducing the packet size to 64 bytes, to achieve the highest packet arrival rate. The INT headers are inserted into every packet using the UDP encapsulation option, and emulate each transit node and the number of metrics (INT instructions) by creating a version of the trace with the corresponding number of INT metadata fields for each packet. We simulate a number of INT transit nodes, varying from 1 to 5 and a number of instructions varying from 1 to 4 for each packet.

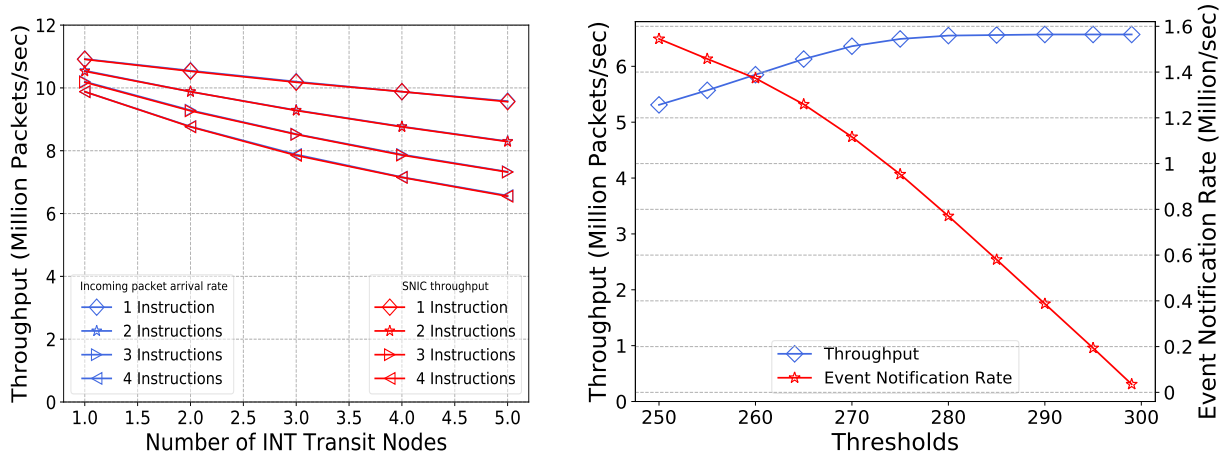
3.6.1.3 Synthetic INT metadata

Assuming for example that there are 5 INT transit nodes and 4 instructions on each node, the values embedded inside each packet are randomly generated according to a exponential distributed probabilities across all possible values for each measurements (*i.e.,* hop latency, queue occupancy, link utilization). For example, the queue occupancy takes values from 0 to 300 with exponential decreasing probabilities. The INT layer and corresponding metadata are inserted into the 64-byte CAIDA 2019 trace to be used in our evaluation. For our experiments, we use MoonGen [73] as pcap trace replay tool.

3.6.2 Throughput and INT Events Rate

When traffic arrives at the sNIC, we process the packet through the packet processing pipeline and update the primary hash table H_P with the flow information, count, timestamp as well as the

INT metadata. Fig. 3.7(a) shows the throughput achieved through our system (packets being processed and then routed back to the MoonGen packet generator. The throughput essentially overlaps the incoming packet arrival rate at sNIC, for different numbers of INT transit nodes and varying number of INT metrics reported (instructions). The highest incoming packet arrival rate is around 11 Mpps with 1 instruction on 1 INT transit node (64 byte packets plus the INT headers and meta-data). As the number of INT transit nodes and the number of instructions increases, the size of each packet would also increase, reflected in the lower per-packet throughput in Fig. 3.7(a). We are able to maintain full throughput across all of the configurations tested, and match the incoming rate even when processing multiple (4) instructions per packet at each of the 5 transit nodes. Thus, our monitoring node can fully function as a 'bump-in-the-wire'. Fig. 3.7(b) shows the throughput

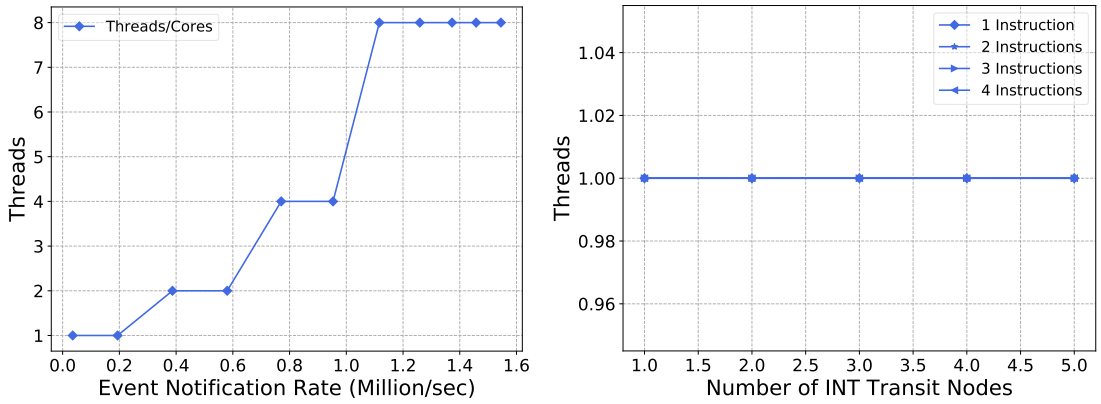


(a) Packet Throughput (processed by sNIC) (b) Tput vs. Event Notif. Rate, varying Q len. threshold

Figure 3.7: sNIC & complete system Tput vs. event notifications

achieved by our complete INT monitoring platform as the rate of INT event notifications posted successfully to the host changes when we vary the threshold for when the metric reported by an INT packet is detected as an event. The operation on receiving a packet with INT meta-data in it is to first update the primary hash table H_P (*i.e.,* update the latest INT information and update the cumulative statistics) for every packet. We also have to evict entries to the general ring buffer R_G on a hash collision, to accommodate a new flow. When the threshold for a metric is smaller, an arrival of an INT packet is more likely to generate an INT event notification to the host. While

the sNIC can receive INT packets at the full line rate for different configurations of transit nodes and metrics reported in each INT packet (as shown in Fig. 3.7(a)), the throughput of INT packets processed through the complete platform (including notifications delivered up to the host without loss) varies depending on the threshold used to generate an event. We show the result for varying INT threshold-crossing events, since they generate a much higher notification rate than change events, thus stressing our platform more. Fig. 3.7(b) is for varying the queue occupancy threshold, generating a varying amount of INT notification events to R_I. Our monitoring platform can maintain full line rate (*i.e.,* 6.57 Mpps as shown in Fig. 3.7(a) with 5 switches and 4 instructions each) even when the notification rate is around 0.6-0.8 Million INT events per second. The throughput decreases when there is a higher rate of notifications, as this introduces more overhead on the packet processing pipeline. However, our monitoring platform can still maintain a throughput of around 5.3 Mpps when there are 1.6 Million INT event notifications to R_I.



(a) Host Thread Usage: R_I (b) Host Thread Usage: R_G

Figure 3.8: Number of Host Threads Required

3.6.3 Host Thread Usage

Fig. 3.8 shows the number of CPU threads used by the host to read the two ring buffers R_I and R_G. Fig. 3.8(a) shows the thread usage for reading the INT ring buffers R_I at different event notification rates. The maximum threads/cores we need to allocate on the host for each ring buffer is 8, which can support a notification rate of 1.6 Million INT events per second. The host only needs 1 thread(core) to read the general ring buffer across all of the experiment scenarios.

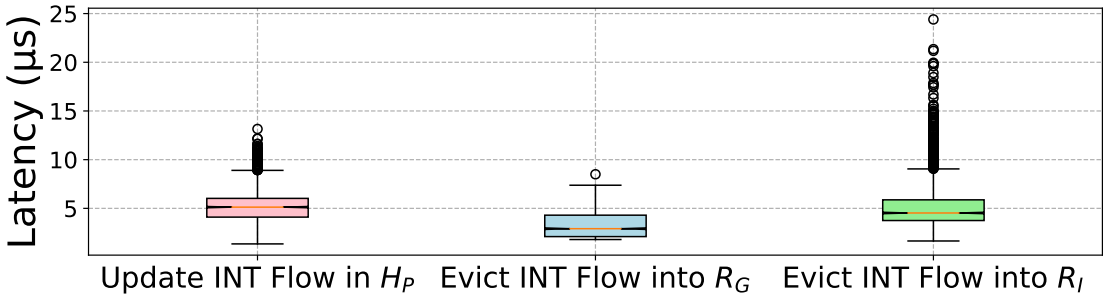
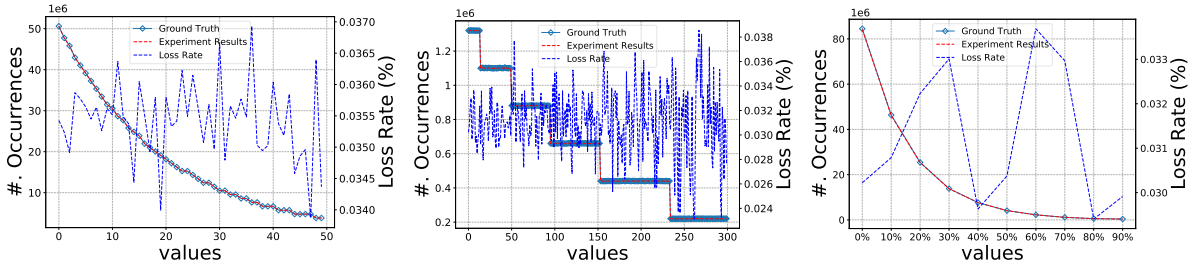


Figure 3.9: INT Processing Latency

3.6.4 Latency

Fig. 3.9 shows the processing latency for INT flows through the different components of the packet processing pipeline. There are three major operations: updating the INT and flow information in the primary hash table H_P; eviction of a flow entry (including INT meta-data) to the general ring buffer R_G when required; the notification of a flow's INT information to the INT event ring buffer to deliver the information to the host. The eviction to R_G takes the least amount of time. The averaging of the INT metric when writing the entry into the ring R_G takes less time than updating the complete path's information, which includes up to 20 measurements (5 switches and 4 measurements each). On the other hand, the first step of updating the INT and flow information in H_P is to write the latest INT metrics received for each of the INT transit node, which takes additional processing. The latency for notification of the INT event into R_I depends on the event notification threshold and the resulting rate. A higher event notification rate has a significant impact on the latency because of the need to have a lock on R_I. We configure the experiment to have around 0.4 million INT events sent to R_I per second. A lock is necessary to ensure correctness and to avoid a race among the different threads of 52 PMEs that can concurrently access and update the ring buffer. Nonetheless, we can observe that the median latency within the sNIC for the three operations of INT processing and event notification are less than 5μs and a maximum of 25μs which is orders of magnitude lower than processing INT on the host CPU.

3.6.5 Accuracy



(a) End-to-end Hop Latency (b) Queue Occupancy at Switch 1 (c) Link Utilization at Switch 1

Figure 3.10: Accuracy in collecting INT metric distribution. Loss rate = 1−(#Observed sNIC Occurrences)/(#Occurrences in trace).

Fig. 3.10 shows the histogram of the metrics collected across the INT transit nodes and their accuracy relative to the ground truth. Collecting each metric requires extracting the INT metrics for each switch in the path. Our experiments consider three metrics (*i.e.,* end-to-end hop latency, queue occupancy and link utilization for the first switch in the path) with the notification rate configured to be around 0.4 Million INT events/sec. Each metric value is randomly generated according to an exponential distribution. Data for Fig. 3.10(a) is collected by summing hop latency across all switches, while for Fig. 3.10(b) and (c), we consider data from the first switch. Fig. 3.10 shows that the inaccuracy at the host for each of the metrics reported to the host is less than 0.04% at the full line rate. This is significantly better than what has been observed with other efforts reported in the literature. We are currently designing methods to robustly identify the bottleneck link in the network (and in the path of individual flows). Flows with the highest end-end latency can be identified, with the host using exported statistics to determine the switch in the path of those flows with the highest queue occupancy/link utilization.

3.7 Related Work

Recent works that support a wide range a queries such as Sonata [75], NetQRE[76], OpenSOC[77] Gigascope [78], Omnimon[79], PINT[80], and BeauCoup[81] exploit powerful programmable

switches and substantial host CPU processing. However, these systems do not conduct protocol-level inspection and thus cannot detect a wide range of stealthy attacks. Marple [82] performs queries entirely on a programmable switch at line-rate, and takes advantage of an off-chip backing store to handle the evictions from the switch data structure (implemented as a hash table with 8-buckets per row) with large memory (above 2^{19} cache slots). FlowRadar [83], uses a BloomFilter and an invertible Bloom lookup table to encode per-flow counters on the switch. However, it is hard to obtain accurate packet counts using approximate methods in a small switch memory [84]. Omnimon[79] conducts network-wide measurement at full accuracy, but it is also limited for detecting low-rate attacks because of P4Switch limitations. *OINT* can also complement OMNIMON to provide a more comprehensive network-centric monitoring solution. Turboflow [85] implements a linear probing hash table and is able using a Barefoot Tofino switch (sNIC option also provided) to store every packet by having microflow records (mFRs) that are continuously evicted, resulting in a very high eviction rate ,thereby increasing the load on the host substantially. *OINT* instead partitions the aggregation function between sNIC and host and stores flow records in the host without loss. Trumpet [86] and Pathdump [87] offload query processing to end-hosts but not on a switch or sNIC, limiting its packet processing capacity.

On the other hand, Sketch-based solutions like Elastic [84], MVSketch [88], Univmon [89], sketchlearn [90], NitroSketch [91] seek to support line-rate telemetry while focusing on identifying heavy hitters and heavy change accurately, while being less focused on low rate or low volume events. *OINT* on the other hand seeks to ensure all the flows are tracked, so that we can support both volumetric analysis and specialized monitoring tasks including slow-rate attack detection and prevention. The sNIC data structure in *OINT* automatically adapts to the packet arrival rate to ensure loss-free, near line-rate processing.

A large number of works [7, 8, 9, 10, 11, 12] have addressed different aspects of processing and collection of INT packets. Here, we focus primarily on INT monitors. IntMon [13] implements an INT monitoring service on the Open Networking Operating System (ONOS). However, it achieves very low processing rates and high cpu utilization. IntCollector [14] also uses UDP encapsulation

for INT packets and the monitor reports INT change events based on a predefined threshold. However the INT packet processing and event detection are implemented on the host CPU, splitting INT packet processing into a fast path (accelerated by an eXpress Data Path (XDP)) and a slow path for exporting and inserting INT events into a database. Because of the packet processing being done on the host CPU, performance is limited. The work in [15] is closest to ours. They implement the INT packet processing and INT event detection using the sNIC P4 pipeline. But, they only report simple threshold-crossing INT events to the stream processor (running on the host CPU) using the kernel bypass technique AF XDP. However, the use of P4 pipeline restricts the per-flow state information to simple registers and counters only, and does not give us the ability to maintain complex per flow state that are required by most server-based networking applications [16]. Also, additional miscellaneous functions such as timeouts *etc.* are not easily implementable using P4 [16]. By using callable C functions and P4, we design a highly efficient INT monitoring platform that not only supports notification of INT events, but also exports the basic INT telemetry report for every INT packet.

4. OPTIMIZED IN-BAND NETWORK TELEMETRY

4.1 Overview

4.1.1 Network Internal State Monitoring

Network internal elements such as routers and links work in a collaborative manner to maintain quality and efficiency of network operations. Understanding the behavior of network internal elements in time can help network operators on diagnosing internal falls and alleviating degradation of network performance [92, 93, 94, 95, 96]. The naive way of collecting network internal statistics is directly accessing network internal elements. For example, commonly used approaches like SNMP [97], RMON [98] retrieve information from routers by answering infrequent polling requests. However, growing size and complexity of nowadays networks limits the performance of these approaches. Another way of monitoring the network internal state is through active probing, which sends probing packets over the network to conduct analysis (*e.g.,* inference of delay, bandwidth or network topology), but there are some concerns of the probing packets such as the bandwidth consumption, interface with normal traffic [99] and they cannot provide custom-specified measurements (*e.g.,* queue occupancy, tx utilization, *etc.*) for each individual switch over the network.

The advanced development in In-band Network Telemetry (INT) [100] allows network operators to collect network internal statistics without introducing new traffic or interfere with routers. The intuition of INT is to have each INT-enabled switch embedded switch-specific measurements inside passing packets until reach the sink where those INT measurement being extracted and processed. The main concern about the INT is bandwidth consumption by the packets overhead introduced by INT (*e.g.,* minimal overhead around 2.8 % of a 1000 Byte MTU [101]) as each switch would append its own measurements. A degradation of 25 % to 60% bandwidth is observed when INT packets transmitted on a 10G nic with different number of hops and switch measurements [102]. Furthermore, there is around 3% increase in the switch processing time by enabling INT on

a 10G interface [103, 101].

In this paper, we researched on optimizing the INT to provide a bandwidth-efficient network internal state monitoring where we allow network operator to specify the bandwidth budget on transmitting INT flows. We argue that the proposed monitoring mechanism can help achieve a ideal monitoring granularity by predicting a specific set of flows to cover the network.

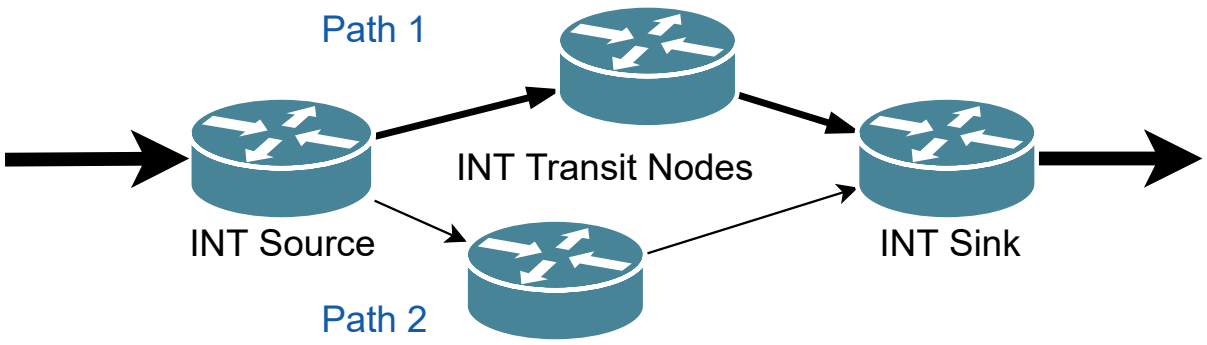


Figure 4.1: Sampling approach at INT source.

4.1.2 Bandwidth Efficient INT

Much work in reducing the bandwidth consumption of INT has focused on the sampling methodology [104, 105, 106, 107]. Sampling the traffic to have a small portion of INT-enabled flows help reduce the bandwidth used for INT, however, it loses information on Non-INT flows and most importantly it can not guarantee a ideal monitoring granularity, which can be crucial for identifying anomalies. For example, some paths might be assigned more packets to carry INT metrics as shown in figure 4.1 where the number of INT packets in path 1 is larger than path 2. The state-of-art work of *PINT* [101] designed a new mechanism combined the idea of traditional INT with a probabilistic data structure to distribute switch information among different packets and decode statistics at the end, which limits packet overhead as low as one bit. But it relies on fairly long lived flows to decode switches information and has high complexity. Optimizations [108, 109] approaches were proposed to address the bandwidth consumption by the INT overhead.

However, the dynamic network states tracking or monitoring granularity are limited.

In our study of bandwidth efficient INT, we aim to provide a consistent network monitoring over all network interfaces by selected INT flows under bandwidth limitation without losing information of non-INT flows. The designed monitoring platform is able to keep tracking of network internal measurements (*e.g.,* queue occupancy, link utilization, *etc.*) and provide real time inference on subset of flows for monitoring after estimating flow paths.

4.1.3 Problem Specification and Contributions

The main challenge of this paper is *how to infer a subset of flows as INT flows for a consistent (*i.e., *at desired granularity) monitoring under bandwidth constrains without introducing packet overhead.* To find such set of flows requires comprehensive knowledge including the distribution of network flows over time and the routes of each flow. However, those paths of network flows over a network are not prior knowledge to the sink but flow statistics such as the flow 5-tuple and flow size are visible at the sink. The common approach for collecting intermediate routers information is active probing (*e.g.,* send ICMP packets), but those probing packets cannot reveal the routers along the paths of normal network flows. The INT technique can tell such information by carrying the switch ID in packets traversing through, however, it requires to install the path information for every packet, which affect the throughput. Assume a 5 hops single path from source to the sink, the INT would introduce minimal of 36 bytes (*i.e.,* 4B shim header + 12B metadata header + 4B switch ID $\times$ 5 hops + any encapsulation) for each packet. Therefore, a more efficient design is required to collect the path information without introducing any overheads affecting the throughput.

Another concern on the main challenge is *how to select a subset of flows as INT flows* after we collected the path information. The selected flows should collect network internal statistics in a consistent way (*e.g.,* report measurements every second), which requires the monitoring platform at the sink to find such *healthy persistent flows.* Additionally, the list of selected flows should be monitored and updated if any of those selective flows become inconsistent or no longer exists. Therefore, we shall explored the way of learning flow patterns to extract the subset of flows out from the normal traffic to carry INT metadata of network internal elements under the desired

bandwidth budget.

We shall leverage the overall design of *OINT*, which involved the coordination between monitoring sink and the intermediate routers over the network to provide a bandwidth-efficient monitoring for network internal states. Our contributions are as follows:

- We first analyze the traffic dynamics using realistic data traces to understand the duration and distributions of flows and show the potential efficiency provided by long-lived flows in section 4.2.
- We design the *OINT* monitoring platform for tracking the flow paths without introducing packet overheads and the *OINT* is capable of inferring an optimal set of flows served as INT flows for a consistent monitoring under the bandwidth budget in section 4.3.
- We provide the implementation details of *OINT* over programmable switches to achieve both bandwidth-efficient and memory-efficient with high packet processing rate.
- We evaluate the *OINT* monitoring platform for the performance (*i.e.,* coverage rate, bandwidth, memory *etc.*) of INT flows.

4.2 Traffic Dynamics

Characterizing network traffic is crucial for strategic decision-making over network such as anomaly detection [110, 111, 112], resource consumption [113], load balancing [114, 115]. Learning traffic patterns helps on finding heavy / small flows and their duration time (*i.e.,* short-lived or long-lived flows), which can be further used for scheduling the INT. Compared to the sampling techniques of INT, learning traffic in advance has better control over desired monitoring granularity and bandwidth consumption. The ideal scenario for monitoring network state is to have a minimum set of *persistent* INT flows to carry INT metrics for routers in the paths as those persistent flows considered to be long-lived flows, which can provide consistent measurements to the sink (*i.e.,* data center). Most flows lasts under a few hundred millisecond and are small (*i.e.,* $\leq 10KB$) in a data center [116]. While a significant short-lived flows contributed to network traffic, long-lived flows (*e.g.,* last than one minute) are also observed but more slower than short-lived flows

and account for around 2% of total flows [94]. Short-lived flows are not feasible for designing a efficient INT because it would require the network operators to update the monitoring flow as INT flows frequently to guarantee consistent measurements. Long-lived flows, on the other hand, have the capability of providing a consistent monitoring as a result of long duration. Therefore, we shall focus on how to select a set of fairly long-live flows as INT flows to carry switch metrics.

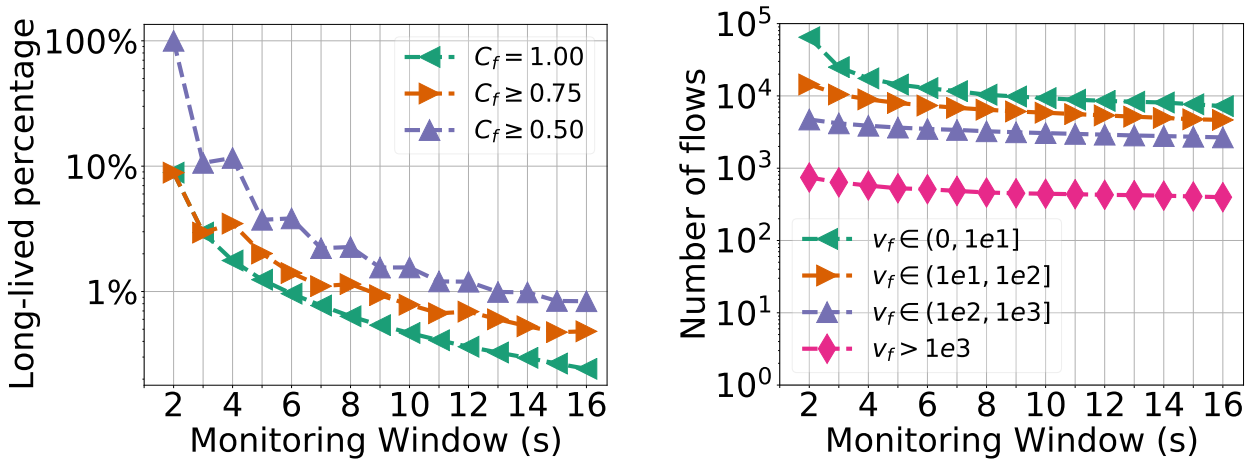
We studied several network traffic traces to understand the traffic dynamics for helping optimize the bandwidth consumed by INT overheads. A 10G testbed we setup on replaying the CAIDA 2019 traces [70] provides around 1.5 $Mpps$, while a 40G testbed gives around 6 $Mpps$. We use the 40G testbed with one second monitoring granularity to illustrate the characteristics of long-lived flows in the realistic data traces. Let C_f denote the coverage rate of a flow f,

$$C_f = \frac{1}{T} \sum_{t \in T} X_t \tag{4.1}$$

where the X_t is the incidence vector over the monitoring window T (*i.e.,* $X_t = 1$ if the flow f found in tth second over T and zero otherwise). Let $N(C_f \geq \alpha)$ denote the number of flows with coverage rate greater than α where $0 \leq \alpha \leq 1$ and let s_f denote the size of a flow f captured in a monitoring interval. Figure 4.2(a) shows the portion of long-lived flows (*i.e.,* $\frac{N(C_f \geq \alpha)}{N(C_f \geq 0)}$ where $\alpha \in \{0.5, 0.75, 1\}$) as the monitoring window T increased. The portion of long-lived flows is decaying overtime because we have a large number of new flows observed. However, the number of long-lived flows is steady overtime as shown in the figure 4.2(b) where the $C_f = 1.0$. Most of the long-lived flows are small size v_f (*i.e.,* $v_f \leq 1e3$) and less than a thousand flows have size larger than 1e3 packets. Although the fraction of long-lived flows with respect to the total number of flows is decreasing overtime, the number of long-lived flows shows a steady trend*.

Those stable long-lived flows can potentially provide a fine-grained INT analysis over networks with advantages of persistent and small sizes. Much research work has proposed algorithms on how to detecting long-lived flows (*i.e.,* persistent flows) [94, 117, 118, 119, 120, 121, 122] either by host

*The distribution analysis with different coverage rate is included in the appendix and they also showed a similar trend as figure 4.2(b)



(a) Long-lived fraction

(b) Long-lived distribution

Figure 4.2: Analysis of long-lived flows.

streaming analysis or offline analysis over packet traces. The intuition on tracking of persistent items is identifying network anomalies such as malicious host associated with a botnet [120], click fraud [123], periodic patterns (*i.e.,* port scan [124, 125], low-rate attacks [24, 110, 26, 126, 127]). However, healthy persistent flows are also frequently found inside network traces with long duration (*i.e.,* days or weeks). For example, application-level keep-alives, time synchronization and multicast control [94]. Therefore, the monitoring platform should be able to provide a candidate list of flows and filter out potentially malicious long-lived flows. Assume we select one healthy long-lived flow $\hat{f}$ with coverage rate $C_f = 1.0$ and average flow size $Avg(v_{\hat{f}})$ around 1000, then the corresponding flow path can be continuously monitored by enabling $\hat{f}$ as INT flow. The minimal additional bandwidth consumption of $\hat{f}$ by carrying INT metrics, on the other hand, is only around $Avg(v_{\hat{f}}) \times 20B \times 8b/B = 160kbps$ (*i.e.,* 16B INT header + 4B switch metrics (ID, hop latency, queue occupancy, tx utilization) $\times$ 1 hops) on a 1 hop path. Compared to the original INT approach that enable each flow as INT flow, which required around $960Mbps$ on a 40G environment ($\sim 6Mpps$), the bandwidth reduction is nearly 99.98%.

Analyzing the traffic patterns gives us privilege on control the amount of INT flows. However, commodity network switches often have limitations on processing overhead in order to sustain

high-rate network traffic. Hence, we shall focus on design the *OINT* to provide a practical implementation on the INT sink (*e.g.,* data center) to track a list of healthy persistent flows for covering the whole network by passively inferring flow path of each flow.

4.3 The *OINT* Monitoring Platform

4.3.1 The *OINT* Overview

The design goals of the *OINT* monitoring platform are: 1) Inferring the flow path without additional packet overhead, 2) Predicting a set of flows as INT flows that last desired monitoring intervals, 3) Providing bandwidth-efficient INT solutions for a network and memory-efficient solutions for INT sink. In this section, we describe how the *OINT* monitoring platform achieve those design goals.

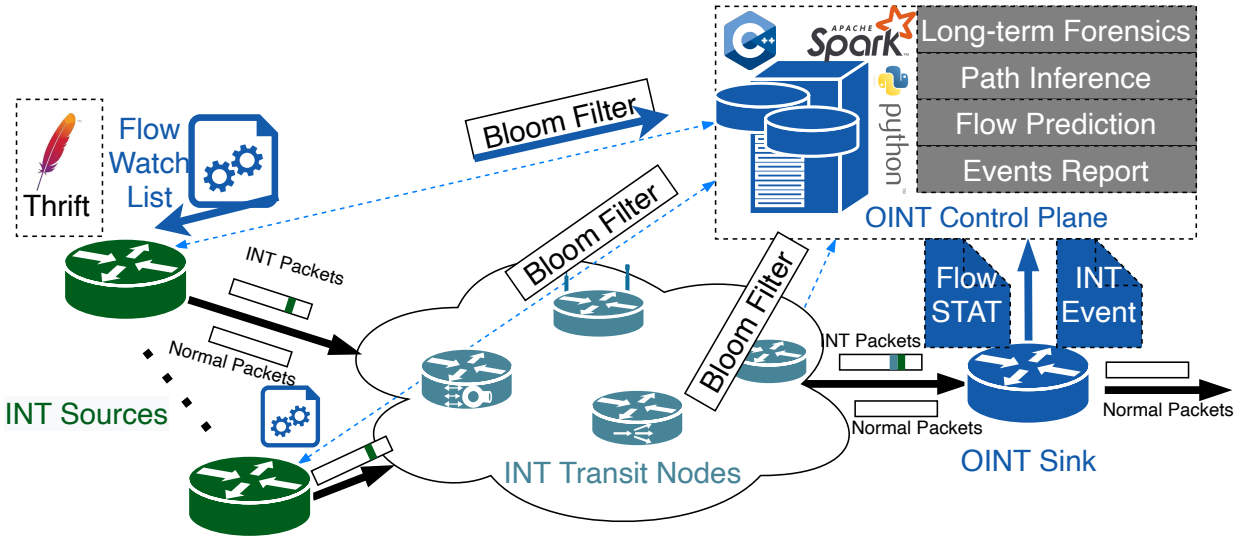


Figure 4.3: The *OINT* Monitoring Platform.

The *OINT* is illustrated in the figure 4.3. The monitoring platform collects network traffic at the *OINT* sink where the aggregation operations of flows and a few INT analysis functions implemented. The *OINT* sink reports the stored flow statistics to the *OINT* control plane for predicting a set of flows to write into INT sources where packets start to carry switch measurements

if the keys (*i.e.,* 5-tuple) of packets exist in the flow watch list [128]. As the reported statistics from the *OINT* only contains information about the flow without the path information (before we assign any set of flows to INT source to carry path information or other switch metrics), we implemented bloom filters for each INT source and INT transit node to store the key of each packets, then the *OINT* control plane would query the information inside each INT source or transit node and compare with the flow statistics collected from the *OINT* sink to predict a set of flows under a bandwidth budget.

The *OINT* functions The *OINT* monitoring platform supports a wide range of (INT-specific) network monitoring tasks, which falls into the following category.

- **Flow paths changing events.** The *OINT* is able to detect if any paths of flows are changed. One aspect of the selected INT flows is to cover each interfaces of network, and the *OINT* would revise the selected flows if paths changing events happened.

- **Threshold-crossing events of switch metric.** The *OINT* is able to report threshold-crossing events such as hop latency across a predefined threshold to the network operator for further analysis.

- **Heavy change events of switch metric.** The *OINT* is able to detect heavy change metrics by comparing measurement received in the current monitoring interval to the one in the last monitoring interval.

- **Distribution analysis of switch metrics.** The *OINT* also provides distribution analysis over collected INT metrics by the collaborations between the programmable switch and the monitoring host.

- **Persistent flows tracking and reporting.** The *OINT* can keep tracking of persistent flows overtime and generate reports to the network operators.

4.3.2 Traffic Collection at The *OINT* Sink

The *OINT* sink serves as the end point of INT flows from multiple INT sources placed in a network and detaches individual switch measurements along the path from a packet. The purpose of traffic collection at the *OINT* sink is to provide packet processing functions (*e.g.,* aggregation of flows of interests) as much as possible before we sent any reports to the host. Correctly design the data structure at the *OINT* sink can support high rate packet processing and reduce the communication overheads between the *OINT* sink and the monitoring host. The data structure should be able to track and aggregate potential long-lived flows at each monitoring interval. For the short-lived flows, we should leverage the remaining capabilities of the INT sink to provide statistics (*e.g.,* lossless or summarized) to the monitoring host. Additionally, the data structure should be able to aggregate the high frequency flows (*i.e.,* heavy flows) to avoid processing overhead at the *OINT* sink and communication overhead to the host (*e.g.,* repeated reports). Therefore, we design a data structure shown in figure 4.4, which consists of a traffic aggregation unit and a report unit, to track flows with long duration and report traffic statistics (*e.g.,* small flows, INT measurements, *etc.*).

Traffic aggregation unit. We designed the traffic aggregation unit to be a hash table for accommodating the incoming normal flows (*i.e.,* non-INT flows). However, a linear hash table with one bucket cannot provide sufficient aggregation with limited number of rows (*e.g.,* 2^{16}) since dynamic network flows (*e.g.,* Millions of flow) would cause hash collision frequently[†]. Hence, we designed the hash table H_T in the traffic aggregation unit with multiple buckets per row to accommodate the hash collision and aggregate the high frequency flows potentially. In order to keep tracking of the long-lived flows, we installed the LRU policy on each row of the hash table so that we evict the least recently used flow out when a hash collision happened. The LRU policy helps to separate out the short-lived flows from the traffic and also aggregate flows with high frequency. Instead of overwriting those hash collisions, we simply forward hash collisions to the report unit where we generate reports of flow statistics to the host.

[†]The probability of hash collision for a hash table is $1 - exp(-\frac{n(n-1)}{2r})$ with total number of flows n and number of rows r of hash table.

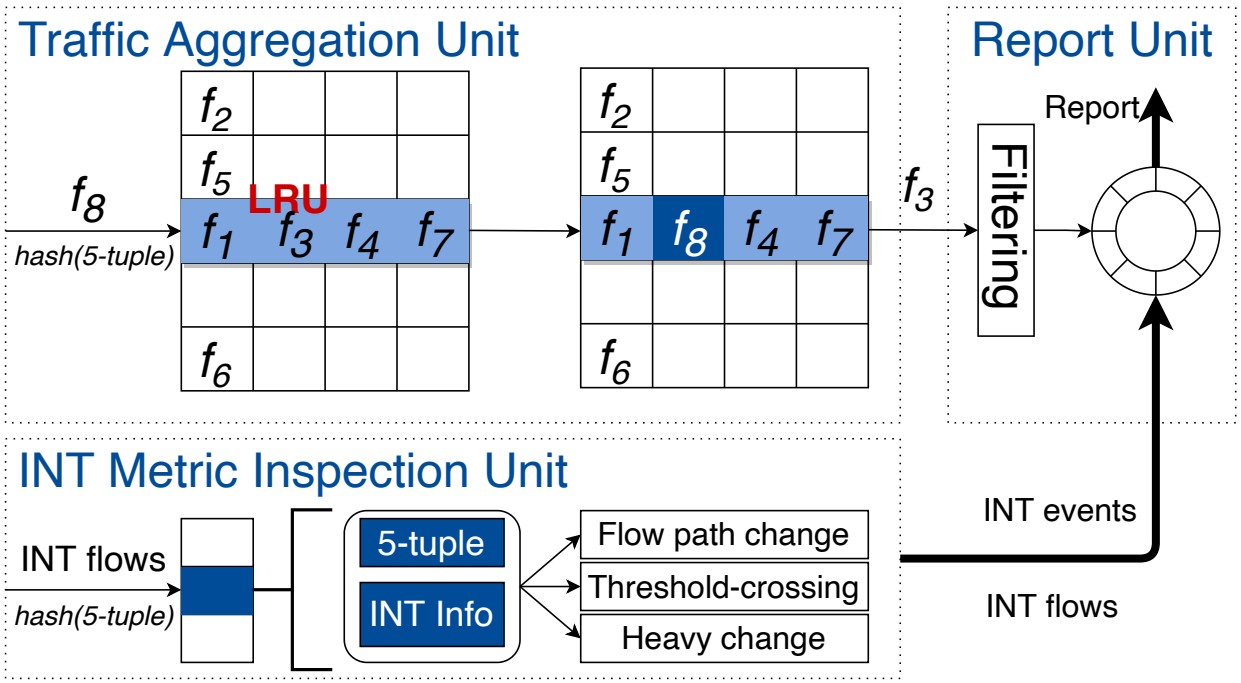
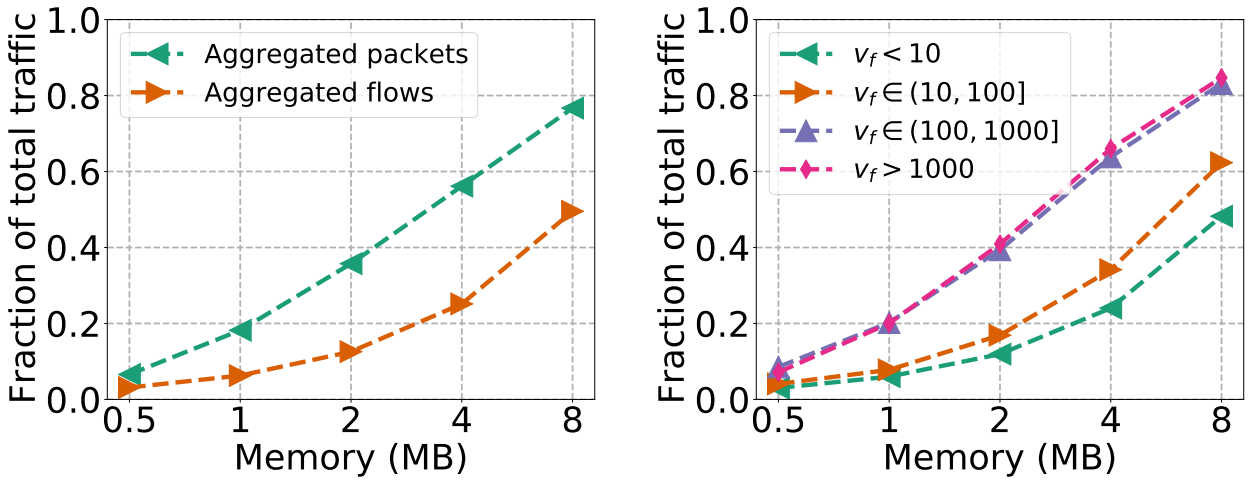


Figure 4.4: The *OINT* sink data structure.

For example, a new flow f_8 arrives at the *OINT* sink as shown in the figure 4.4 and hashed (*i.e.,* flow 5-tuple) into the third row of the hash table but it cannot find a empty slot as all available buckets are taken. As the new flow iterates those buckets, we also record the LRU bucket, which is located in the second column of corresponding hash index. Therefore, the LRU flow f_3, which is replaced by the new flow f_8, is evicted to the report unit for further processing.

We designed our hash table H_T with 4 buckets each row based on our experiments analysis[‡], and the corresponding performance of hash table aggregation using CAIDA traces [70] is shown in the figure 4.5. Larger memory certainly has capabilities to accommodate more network flows than small memory allocation on the hash table H_T as shown in the figure 4.5(a). Figure 4.5(b) shows the flow distribution compared with the total traffic received, and the fraction of total traffic calculated the amount of flows with size v_f (*i.e.,* $v_f < 10$) found in the *OINT* sink divided by the total traffic with size v_f. The *OINT* sink is able to aggregate large flow with small flows being evicted out from the hash table.

[‡]We provided the reports of selecting number of buckets in the appendix



(a) Flow aggregation

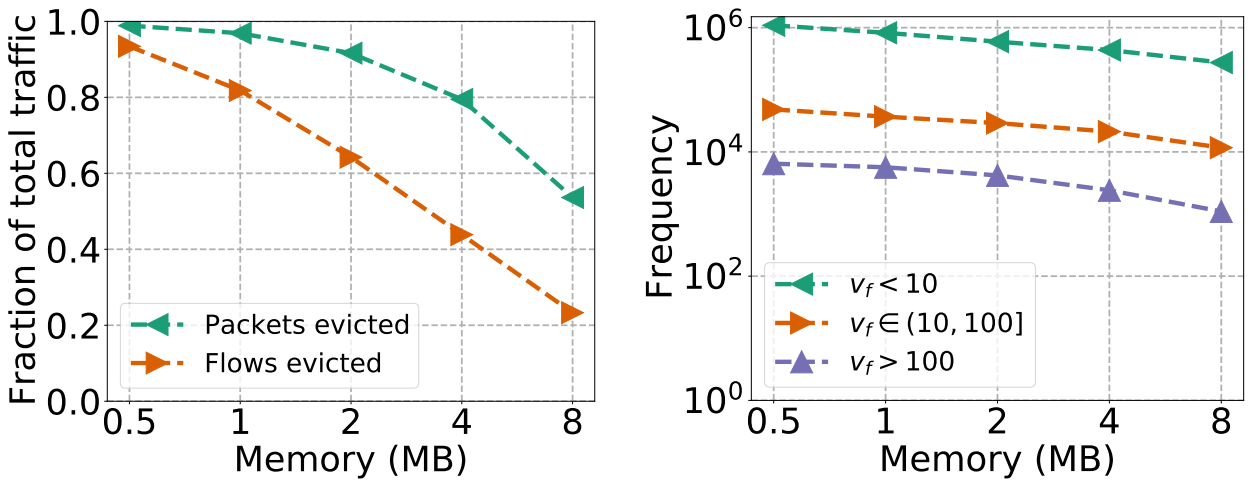
(b) Flow distribution

Figure 4.5: The *OINT* sink traffic aggregation unit analysis. Total traffic: CAIDA 2019 trace received in one monitoring interval (*i.e.,* 1s).

Report unit. The first function of the report unit, as we can see from the figure 4.2b, is to accommodate the evictions from the traffic aggregation unit. The report unit contains a filtering stage and a storage implemented by rings to communicate with the monitoring host. Ideally, we can report each eviction to host for a lossless tracking of all incoming network traffic, which is possible as we expected frequently visited flows would be captured inside the hash table with short-lived flows being evicted to the report unit. However, high volume of evictions is expected under high packet arrival rate or small memory allocation on the hash table as shown in the figure 4.6(a). Hence, the filtering stage help to filter out the small flows and only write fairly large flows into the rings as the small flows come with a small duration, which do not contribute much on selecting long-lived flows.

We leverage the amount of evictions generated by the traffic aggregation unit with different memory allocation and the results are shown in the figure 4.6 (b). Although the small memory generate more evictions than the large memory, the distribution of flows with large size (*i.e.,* $v_f > 100$) being evicted are similar. The filter stage is to apply a threshold on flow size (*i.e.,* $v_f > 100$) to maintain a stable eviction rate to the host when experience high packet rate or small

memory allocation, which help us control communication overhead between the *OINT* sink and the monitoring host.



(a) Total eviction

(b) Flow distribution

Figure 4.6: The *OINT* sink traffic eviction analysis. Total traffic: CAIDA 2019 trace received in one monitoring interval (*i.e.,* 1s).

INT metric inspection unit. We redirect the INT flows into a separate processing pipeline separated from the traffic aggregation unit as we expect the amount of INT flows are very small, which is the optimization purpose of our monitoring platform and we shall discuss in more details in section 4.3.4. We designed a hash table H_I with one bucket to store the INT metric collected over entire network and evict entry out if any hash collision happened. As the number of INT flows is limited (*i.e.,* proportion to the number of distinct paths over a network), we also expect the number of evictions is small. Similar to the evictions at the traffic aggregation unit, the INT evictions would be wrote into the rings. Other than the evictions of INT flows, the INT metric inspection unit would also generate INT events (*i.e.,* flow path change, threshold-crossing, heavy change metrics) to the rings.

Each incoming INT packet will first try to find an corresponding entry in the hash table with the same flow 5-tuple to update or create a new entry if it cannot find any. We first compare the

current INT metrics of switches along the flow path to the metrics embedded in the incoming INT packet if it can find an match in the hash table, and report any suspicious switch metrics (*i.e.,* heavy change, threshold-crossing) or path change of the flow to the monitoring host through report unit. For continuously monitoring over a certain flow path to get distribution of all switches metrics (*e.g.,* due to multiple heavy change or threshold-crossing events along the path), the *OINT* monitoring host would update the p4 match-action table (MAT) of the *OINT* sink to directly mirror the INT flow to the host and the original INT flow sent to the egress pipeline after extracting INT information.

4.3.3 Network-wide *OINT*

One design goal of the *OINT* is to be able to infer the flow paths of those flows reported by the *OINT* sink without introducing additional packet overhead in the network. In order to achieve this, we design a cooperative way among network switches and the *OINT* to recover the routing of each flow reported from the *OINT* sink by allocating a small amount of memory for a bloom filter inside each INT source / transit node. The purpose of the bloom filter at each INT-enabled switch is to record flows traversing through by setting the corresponding hash bits to 1.

4.3.3.1 Data operations.

At the very beginning of the monitoring, each INT source and INT transit node will initialize a bloom filter with width m and the number of hash functions k. For each packet traversing through the INT-enabled switch, those k hash functions will help to set the corresponding hash index to 1 [129]. As the traffic collected at the INT sink does not contain trace information, we use the report unit of the *OINT* to forward the flow statistics to the host where we conduct path recovery. The host would compare the flow statistics with the bloom filters metrics collected over the network to provide the path information for flows of interests (*i.e.,* flows with high coverage rate defined in equation 4.1). Let $\mathcal{S} = \{s_1, \ldots, s_n\}$ denote the set of switches of a network and let $\mathcal{S}_f$ denote the set of switches along the path of flow f such that $f \in \mathcal{F}$ where $\mathcal{F}$ is the set of flow candidates. Any flow paths from the INT source to the INT sink can be uniquely determined by $\mathcal{S}_f$ such that

$\mathcal{S}_f \subset \mathcal{S}$. Therefore, the *OINT* monitoring host should provide solutions on finding the smallest set of flows such that $\cup_{f \in \mathcal{F}} \mathcal{S}_f = \mathcal{S}$ to eliminate unnecessary flows (*i.e.,* overlap paths, large flow sizes) and minimize network burden for introducing INT flows.

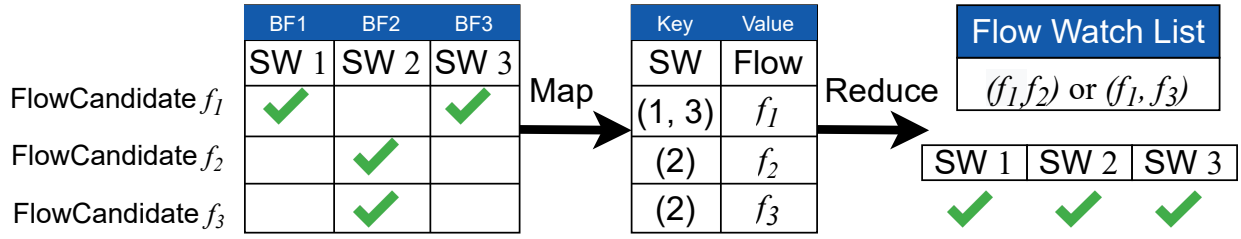


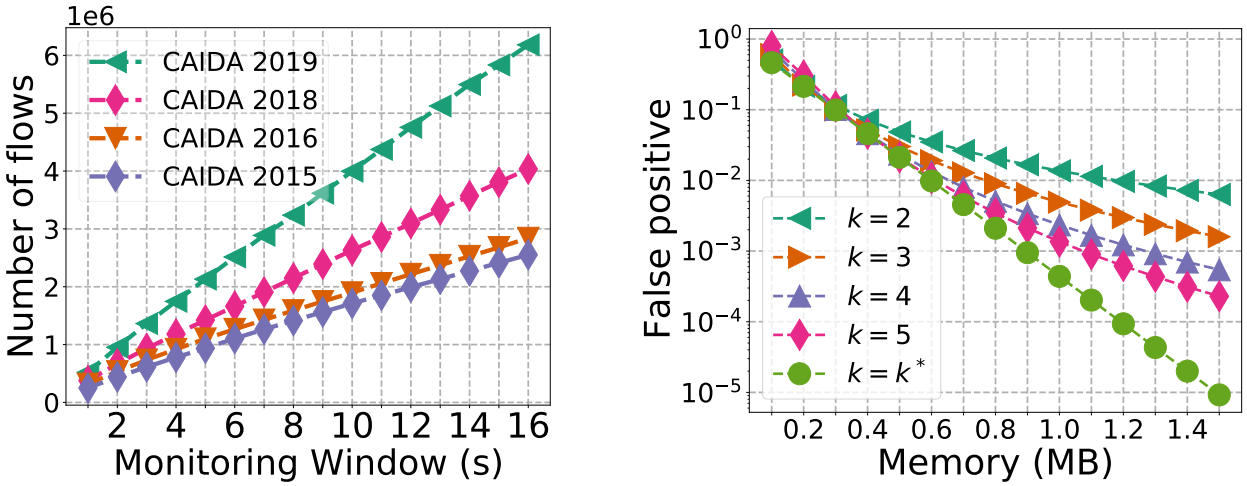
Figure 4.7: Flow watch list generation.

For example, a scenario of three INT-enabled switches of a network is illustrated in the figure 4.7 and we have $\mathcal{S} = \{s_1, s_2, s_3\}$. We assume that the host generate three flow candidates after analyzing the flow statistics reported from the *OINT* sink. The host first try to see if each flow candidate found in each bloom filter (BF) collected from the network switch. In this example, flow candidate f_1 is found in switch 1 and switch 3 with $\mathcal{S}_1 = \{s_1, s_3\}$, and $\mathcal{S}_2 = \mathcal{S}_3 = \{s_2\}$. As flow paths are uniquely defined by a set of switch IDs, we map each flow to a new key (*i.e.,* s_i) and store the 5-tuple as the value of the key. Then, the host conduct a reduce operation on each group to provide optimal solutions. In this case, the set of flow ($\{f_1, f_2\}$ or $\{f_1, f_3\}$) has the optimal solution to cover the entire network.

4.3.3.2 Accuracy analysis.

The error of a bloom filter is related to the number of hash functions k, the filter width m (*i.e.,* bits) and the number of inserted flows n. For a single switch implemented a bloom filter with parameter (m, k) and assume n flows have already set corresponding bits to 1 using k hash functions, the probability of false positives ε (*i.e.,* a new flow found in the bloom filter) is $(1 - e^{-kn/m})^k$ with the optimal choice of $k^* = \frac{m}{n} ln2$ [129, 130]. We start by analyzing the number of distinct flows we expect to handle and the amount of memory allocation for the bloom filter to

achieve a small false positive rate. Figure 4.8(a) shows the amount of distinct flows observed in a 40G setup environment as monitoring window increased. Figure 4.8(b) shows the false positive with different memory and the number of hash functions, which conducted by using n equals to the number of flows in the first monitoring interval of CAIDA 2019 trace. For $k = 3$, we need around $0.8MB$ memory for a bloom filter to reach 0.01 false positive.



(a) Distinct flows

(b) False positive of BF

Figure 4.8: Bloom filter (BF) resource consumption.

We now extend the single switch / single bloom filter case to a large network with larger number of INT-enabled switches with bloom filters implemented. Network operators can placed one or more INT sources (*i.e.,* start point of embedding INT metrics into network packets) with distinct instructions in the network for different monitoring purposes. For each INT sources, there are one or more flow paths with the *OINT* sink node as the end point (*i.e.,* extracting the INT header and INT metrics). Let $\mathcal{P} = \{\mathcal{S}_1, \ldots, \mathcal{S}_{|\mathcal{P}|}\}$ denote the collection set of distinct flow paths from INT sources to the *OINT* sink such that at least one switch is different in any two elements of $\mathcal{P}$. Let $\overline{\mathcal{S}_{i,j}}$ denote as the complement of a path collection $\mathcal{S}_i$ from $\mathcal{S}_j$ such that $\overline{\mathcal{S}_{i,j}}$ contains switches from $\mathcal{S}_j$ that are not in $\mathcal{S}_i$. The *OINT* aims to use a set of flows to cover the entire network constructed

by the paths collection $\mathcal{P}$, and we hereby provide analysis on the error of selecting the optimal set of flows candidates (*i.e.,* the selected optimal flows cannot cover the entire network).

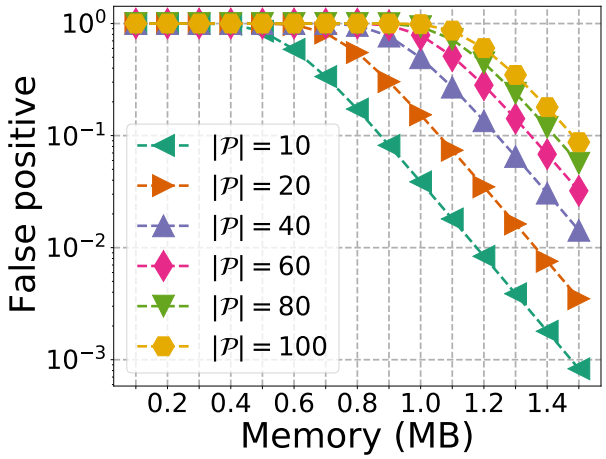
Theorem 8. *The probability of false positive γ on choosing the minimal set of flows to cover a network with path collection $\mathcal{P}$ is $1 - \prod_i \prod_j (1 - \varepsilon^{|\overline{\mathcal{S}_{i,j}}|})$ where $i, j \in \{1, \ldots, |\mathcal{P}|\}$ and $i \neq j$.*

Theorem 8 provides an approximation error analysis over a network with path collection $\mathcal{P}$ from the INT source to the *OINT* sink and the comprehensive analysis and proof are provided in the appendix. The worst scenario is each distinct path has only one switch, which makes $\overline{\mathcal{S}_{i,j}} = 1$ for any $i, j \in \{1, \ldots, |\mathcal{P}|\}$ and $i \neq j$, and the error is

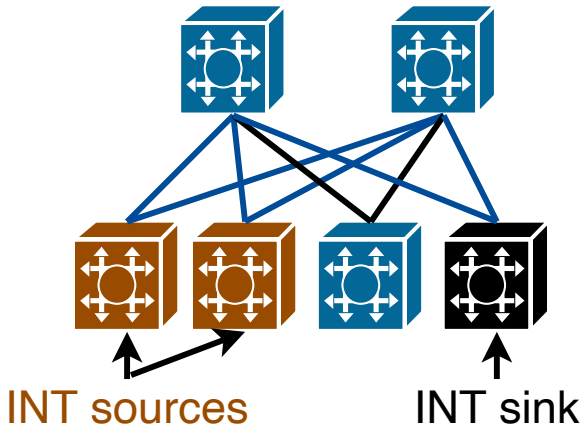
$$\gamma_{max} = 1 - (1 - \varepsilon)^{|\mathcal{P}|(|\mathcal{P}|-1)} \tag{4.2}$$

In this case, we cannot cover the network with selected flows if any of them were found in the bloom filters implemented at switches outside their flow paths. In general, we would expect the distinction between any two paths is larger than 1, which help reduce the error γ as $\overline{\mathcal{S}_{i,j}}$ increased.

In order to quantify the worst case error with different network parameters (*i.e.,* $\mathcal{P}$, memory allocation of switches), we vary the switch memory for bloom filter and $|\mathcal{P}|$, which is shown in the figure 4.9(a) with ε calculated under optimal k^*. Larger $|\mathcal{P}|$ certainly contributes to larger error as we can infer from the equation (4.2) also. In order to have a reasonable false positive error, for example under 0.1, the memory required for distinct flow paths $|\mathcal{P}| = 100$ is 1.5 MB. However, a smaller false positive should be expected as 1) the number of distinct flow path $|\mathcal{P}|$ is generally small since the number of INT-capable switches is limited in current networks; and 2) the path distinction $\overline{\mathcal{S}_{i,j}}$ is larger, which makes $\varepsilon^{|\overline{\mathcal{S}_{i,j}}|}$ smaller. For example, a simple 2-Tier data center architecture is illustrated in the figure 4.9(b) and we have 2 INT sources (*i.e.,* brown switches) and 1 INT sink (*i.e.,* black switch) with 4 distinct flow paths (*i.e.,* blue connections). The path distinction $\varepsilon^{|\overline{\mathcal{S}_{i,j}}|}$ are either 1 or 2 depends on how many different switches between two flow paths, thus the error $\gamma = 1 - (1 - \varepsilon)^4 (1 - \varepsilon^2)^2$ according to theorem 8, which is around 0.04 for $\varepsilon = 0.01$.



(a) False positive γ_{max}



(b) 2-Tier DC

Figure 4.9: The *OINT* error analysis.

4.3.4 Bandwidth-Efficient *OINT*

4.3.4.1 Flow candidates selection

After collecting the flow statistics from the *OINT* sink, we need to provide an optimal set of flow candidates to generate flow watch list, which will be pushed into each INT source for generating INT flows. Ideally, the set of flow candidates should provide a consistent monitoring overtime, which needs those candidates have considerably long duration than other flows. Hence, the intuition on selecting the optimal set of flows is that those flows have high coverage rate C_f (defined in equation (4.1)) in the current monitoring window and thence can provide high coverage rate in the future. As we can see from the traffic analysis in the section 4.2, the number of long-lived flows are stable over time and some of them can be selected to provide a consistent monitoring across the network. Therefore, we start by formulating the optimization problem of selecting an optimal set of flow to cover the network and then designed an efficient algorithm on to implement on the *OINT* monitoring host.

Let $\mathcal{F}^{(*)}$ denote any sets of flows that can cover each path of a network and we have $\mathcal{F}^{(*)} \subseteq \mathcal{F}$ where $\mathcal{F}$ is the set of total flows. Let $C_{\mathcal{F}^{(*)}}$ denote the coverage rate of a corresponding set of flows similar as equation (4.1) but $X_t = 1$ when any flows in the set found in monitoring interval t and

let C^* denote a desired coverage rate.

$$\begin{aligned} \min_{\mathcal{F}^{(*)}} \quad & |\mathcal{F}^{(*)}| \\ \text{s.t.} \quad & \sum_{f \in \mathcal{F}^{(*)}} v_f * a_{INT} \leq B \\ & a_{INT} \leq MTU \\ & C_{\mathcal{F}^{(*)}} \geq C^* \end{aligned} \tag{4.3}$$

where B is the bandwidth budget of the flow path, which can be defined by the network operator in advance, and a_{INT} is the average packet size of the INT flow f. For each flow in the selected set, $v_f * a_{INT}$ describes the average amount of bytes needed for a flow to carry INT metrics over the network. The naive way of finding such a optimal set of flows would try every combinations of flows received, which need $2^{|\mathcal{F}|}$ operations, thus an efficient method is required. As the monitoring host received each individual flow report from the *OINT* sink at each monitoring interval, we maintain a data structure to help quickly select the optimal set of flows.

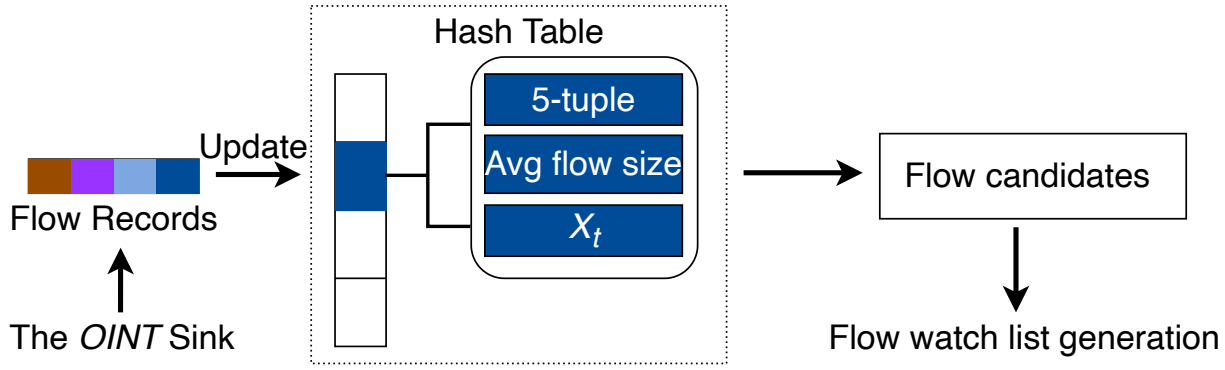
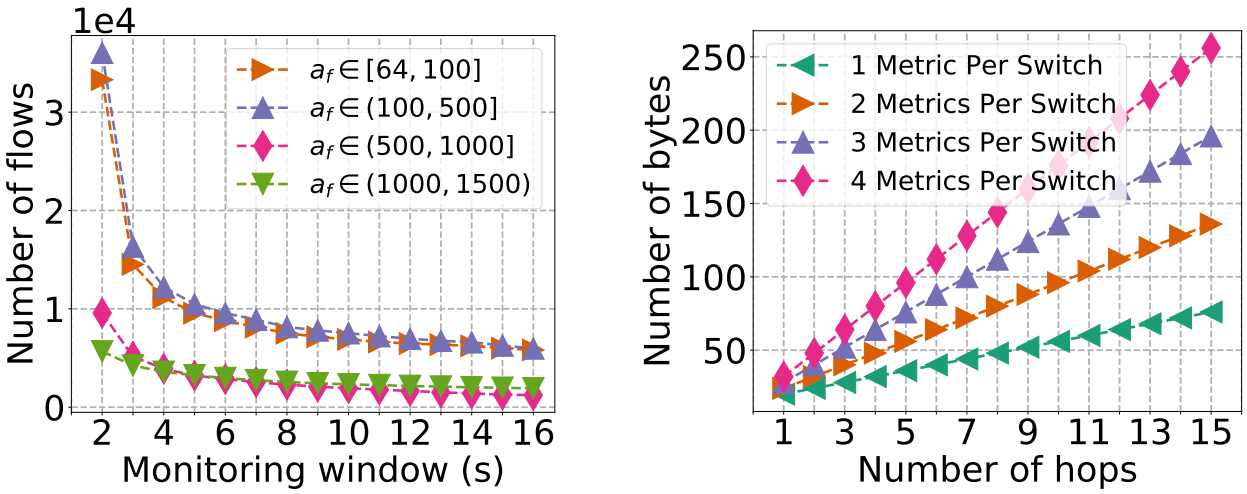


Figure 4.10: Optimal set selection of the *OINT*

As illustrated in the figure 4.10[§], a hash table is allocated for each flow received from the *OINT* sink and each entry of the hash table contains the flow 5-tuple, the average flow size over current

[§]We provided the full algorithm in the appendix

monitoring window T and the incidence vector indicates if a flow is present in each time interval. During the update process of each flow record, we directly insert a flow into a flow candidates list if this flow has already satisfy the constrains (*i.e.,* average flow size, coverage rate, MTU). At the end of a monitoring window (*i.e.,* received records of the last monitoring interval from the *OINT* sink), we examine those flow candidates as we illustrated in the figure 4.7 to select the optimal set of flows to cover the network. Instead of discarding each flow record at the end of each monitoring window, we push those flow records into a Redis [74] database for long-term forensics (*i.e.,* report long duration of flows (*i.e.,* days or weeks) to network operators). The monitoring window size is predefined, which we set to be 10 (s) as we can obtain enough number of long-lived flows without taking two much space on the monitoring host (*i.e.,* figure 4.2b). The *OINT* keeps monitoring for each monitoring window and updates the flow watch list if anything changed such as previous selected flows become inconsistent or flow paths are changed. In the process of flow watch list generation, we also filter out flows in the blacklist provided by the network operator after identifying unauthorized flows (*e.g.,* anomalies).



(a) Packet size distribution

(b) Long-lived fraction

Figure 4.11: The long-lived flows ($C_f = 1$) analysis.

Besides the advantage of long duration of INT flows, they should also have the capacity for

the additional INT headers and INT metrics. We provided an analysis over the average packet size distribution of the long-live flows ($C_f = 1$) in the figure 4.11(a). As we can see from the figure, a large portion of long-lived flows has average packet size between 64 bytes to 500 bytes with small portion ($\sim 20\%$) of long-lived flows has larger packet size over 500 bytes. In order to collect the INT metrics along the path for a packet, we illustrate how many bytes required for a packet in the figure 4.11b. We set the maximum number of hops to be 15, which can be set by the operator in the *Remaining hop count* field of INT header. We also vary the different number of switch measurements from 1 to 4 such as hop latency, queue occupancy, link utilization, switch ID, which are the most important switch metrics to collect. The largest space required to carry INT data is around 250B (*i.e.,* 15 hops and 4 metrics per switch), which can be embedded inside around 80% of long-lived flows.

4.4 Optimizations

We provide a few optimizations over the *OINT* monitoring platform to accelerate the flow candidates selection and also reduce the communication overheads at the same time.

4.4.1 Optimal set selection

The *OINT* monitoring host keeps tracking of the average flow size of each flow until reach the end of a monitoring window T for predicting a set of a optimal long-lived flows. However, there might be small long-lived flows (*i.e.,* $v_f \leq 100$) as we can see from the figure 4.2b where the number of small long-lived flows are around 1e4. For those long-lived flows, we cannot provide a fine-grained network monitoring as the average measurement time of the flow path is larger than 10 ms during which the network state can change dramatically. In order to select a reasonable flow candidates list and also help reduce the size of flow candidates, we bounded the average size of flow at each monitoring interval.

Let $1/\Delta$ denote a predefined monitoring granularity and thence Δ is the smallest flow size to provide such monitoring granularity. As the flows size fluctuated overtime, we use Δ as the threshold on average size of flows. The *OINT* monitoring host applies a check when update flow

records (*i.e.,* average size, X_t) at each monitoring interval and only forwards flows satisfying the granularity to the flow candidates list. For example, as illustrated in the figure 4.12 with $\Delta = 250$ (*i.e.,* 4 ms monitoring granularity), the monitoring host is updating the flow records at the monitoring interval $t = 3s$. Size of flow f_1 does not meet the threshold $v_{f_1} < \Delta/2 = 250$ but the second flow (*i.e.,* f_2) does. If both of the flows are identified as long-lived flows (*e.g.,* $C_f \geq 0.75$, then we add f_2 to the flow candidates list while f_1 is directly forwarded to long-lived flow report. Generally, we can apply $\Delta = 100$ to eliminate a large amount small size flows.

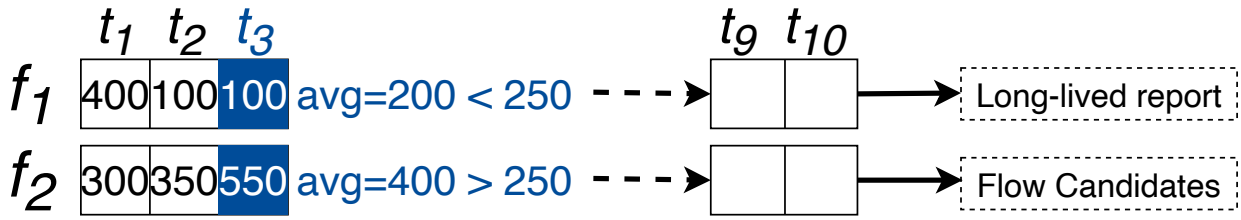


Figure 4.12: Optimization on flow candidates selection with monitoring window $T = 10s$ and average threshold $\Delta = 500$.

4.4.2 Flow watch list insertion

The generation process of flow watch list depends on decoding the flow paths of each flow candidates. For a large network, communication overheads between the *OINT* and each INT source and INT transit node can be linearly increased for retrieving bloom filters information, which can also reduce the available network bandwidth. Therefore, we propose two optimized approaches on updating the flow watch list in each INT source.

Directly configure. After the flow candidates were selected, we first match those flows with the blacklist provided by network operators to filter out potential malicious flows, then we directly insert all flow candidates to each INT source. The advantage of this approach is that we reduce the resource consumption on retrieving bloom filters and generating flow watch list and also the computation time. By having all flow candidates installed in the INT sources, we reach the same

goal (*i.e.,* network coverage) as the customize flow watch list for each INT source with slightly higher bandwidth consumption than the original approach since we might have overlapped flows for monitoring a flow path.

Hybrid. The *OINT* only communicates with limited number of network switches (*i.e.,* INT sources only) to transfer bloom filter information. This approach enable the *OINT* to customized select flow watch list for each INT source, which is the origins of INT flows. In this way, we can quickly generate a flow watch list for each INT source without consuming too much bandwidth to cover the network. The latency on updating flow watch list is shown in the figure 4.13(a) and we only need around 100 ms to update flow watch list with a thousand flows.

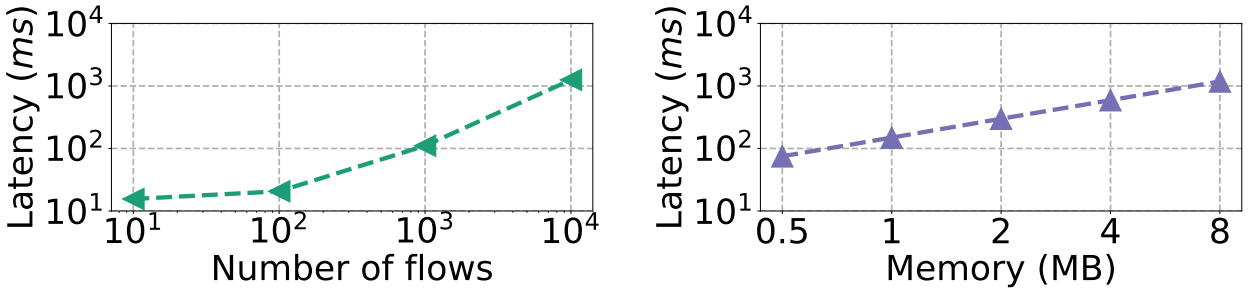
For those approaches, the *OINT* can always enable a feedback mechanism to dynamically update the flow watch list. The feedback comes from the enabled INT-flows, which collect the flow paths information (*i.e.,* switch ID, *etc.*) and can further help reduce the size of flow watch list by eliminating non-essential INT flows (*i.e.,* overlapped).

4.4.3 The *OINT* sink hash table compression

The hash table H_T at the *OINT* sink stored the potential long-lived flows. At the end of each monitoring interval, we send flow records in the H_T to the *OINT* monitoring host. However, larger hash table on the *OINT* sink generally require longer time on updating the hash table on the host. As illustrated in the figure 4.13(b), the amount of time linearly increased by the number of flow records. Therefore, we propose an optimization over the flow records of hash table sent to the *OINT* monitoring host, which is similar to the filter stage at the report unit. We limit the flow size of flows stored in the H_T to send an update, which only contains flow records with size larger than a threshold (*e.g.,* 100) while smaller flows would remain in the H_T until eviction happen.

4.5 Implementation

In this section, we discuss the hardware implementation details of the *OINT* monitoring platform over programmable switch provided by Netronome [131] and also provide solution for general p4 pipeline implementations. More implementation information can be found in the appendix



(a) Flow watch list

(b) The hash table H_T

Figure 4.13: Latency analysis.

(*e.g.,* source code, embedding INT metrics).

4.5.1 INT encapsulation and parsing

There are many possible ways to encapsulate the INT information into packets such as over TCP/UDP and VXLAN GRE [128]. We embrace an encapsulation by introducing an additional UDP layer to indicate the presence of INT metrics between L3 and L4 layer as illustrate in the figure 4.14. If the original packet is an UDP packet, then we utilize the original UDP header by changing the destination port to indicate an INT header present after the L4 layer (*i.e.,* UDP/TCP header). The INT header includes the INT shim header (*i.e.,* carry information such as original UDP destination port and INT metric length) and the INT metadata header (*i.e.,* includes parameters such as remaining hop count and instructions bits). The INT metadata, which store the per switch measurements, follows the L4 layer. More details can be found in the INT specification [128].

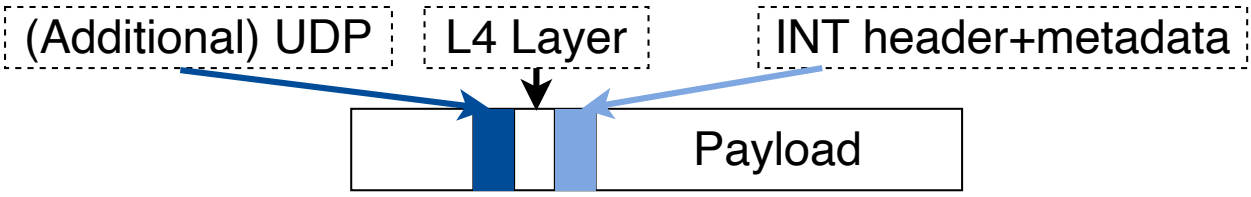


Figure 4.14: INT packets encapsulation.

In order to processing INT packets with uncertain number of INT sources and INT instructions (*i.e.,* different measurement metrics), the *OINT* sink should provide a correct INT parsing. Hence, we utilize the length and instruction bit map stored in the INT metadata header to parse the INT metadata, which is stored in the p4 header stack. At the egress pipeline, we remove the INT information of INT packets and then forward normal packets.

4.5.2 Hardware implementation of the *OINT*

The programmable switches (*i.e.,* 10G, 40G) we used to implement the *OINT* are the smartNICs from the Netronome [131]. Instead of p4 processing capabilities, the smartNICs also provide C functions for complex packet processing in parallel, which has advantages in reducing packet process latency and supporting complicated flow state tracking, which can be difficult in the p4-only packet processing. We first discuss the implementation over the Netronome smartNIC, and then we provide solutions for general programmable switch that process packets in a standard p4 pipeline.

Netronome smartNIC. The p4 pipeline we designed is for packet header processing (L1 $\sim$ L4 layers) and INT header / metatdata extraction. After processing the packet header, we invoke an additional match-action table, which enable the complex C functions to process network (INT) packets. The callable C functions have two stages, which are for fast packet update (*i.e.,* find the same flow and increase the counter) and eviction to report unit respectively. Each entry of the hash table H_T in the *OINT* sink consists of flow 5-tuple (16B), packet count (4B), timestamp (8B) and current monitoring interval (4B), which is 32B in total. Similar designed is used for INT metric inspection unit, which is a small hash table (*i.e.,* 0.6 MB) with one bucket each row but each entry has an additional INT buffer storing INT metrics. Currently, we support maximum 8 hops with 4 measurements per switch for a flow path to store in the INT buffer (128B). For flows with larger hops, we directly forward the INT metrics to the *OINT* monitoring host since there are limited INT flows installed by INT sources. At the end of each monitoring window (*i.e.,* 10s), we utilize the direct memory access (DMA) engine on the smartNIC to transmit flow records (*i.e.,* stored in the communication packets) to the monitoring host, which only dedicate 1 thread for accommodating

flow records.

Flow watch list generation. Upon receiving flow records from the *OINT* sink, the monitoring host keeps updating the flow entry (*i.e.,* average flow size and incidence vector) and report any INT events (*i.e.,* path change, switch metrics analysis) to the network operators. The MapReduce operations for the flow candidates are implemented using Apache spark [132]. For the configurations of INT sources, we utilized the Apache Thrift APIs [133] to install flow watch list into each INT source, which monitors the incoming flows and embeds the INT instructions and metrics into flows that match with any records in the flow watch list.

4.6 Evaluations

In this section, we evaluate the *OINT* monitoring platform using realistic setup. The evaluation focus on the performance of the *OINT* with selected flow candidates such as the coverage rate, flow sizes and resource consumption, *etc.* .

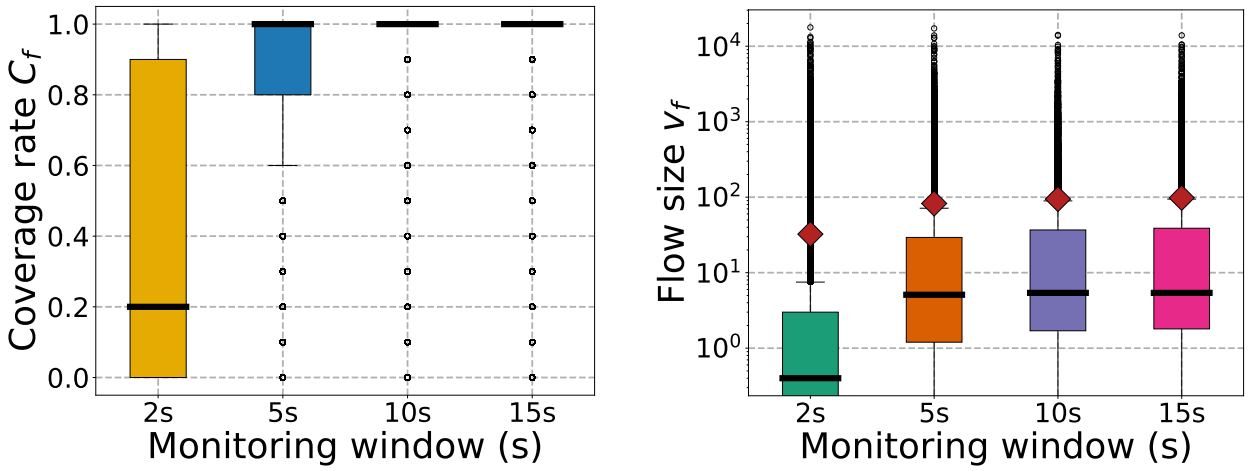
4.6.1 Evaluation setup

Hardware. The testbed on evaluating the *OINT* consist of Netronome Agilio LX 2×10 GbE sNICs and 2×40 GbE sNICs with 8GB DDR3 memory, which support p4 pipeline and additional callable C functions packet processing, and a Linux server with 10 Intel Xeon 2.20GHz CPU cores.
Traces. The realistic traces we used are from four years CAIDA [70] traces (*i.e.,* 2015, 2016, 2018, 2019).

4.6.2 Monitoring window sensitivity

We first evaluate the basic version of the *OINT*, which only consists of the traffic aggregation unit (without compression), report unit (without filter stage) and INT metric inspection unit. We setup the C^*, which is the coverage rate constrain on flow candidates, with $C^* = 1.0$ (*i.e.,* flows that exist in every monitoring interval), and assume that the bandwidth budget can support any generated flow candidates, which generated by the monitoring host at the end of the monitoring window (then the *OINT* starts the next monitoring window).

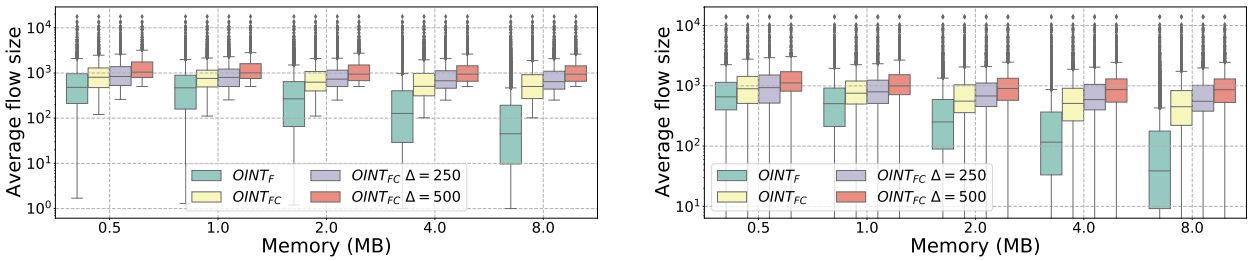
We first test the performance of coverage rate in the future with different monitoring window



(a) Coverage rate

(b) Flow size

Figure 4.15: Future 10s performance

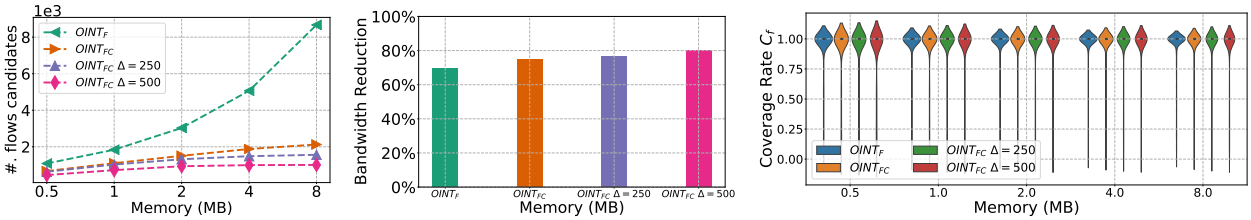


(a) Selected flow size distribution

(b) Future flow size distribution

Figure 4.16: Flow size distribution analysis. (a) the distribution of average flow size of flow candidates selected with monitoring window $T = 10s$. (b) The distribution of average flow size selected flows observed in the future 10s.

size $T \in \{2, 5, 10, 15\}$. As illustrated in the figure 4.15(a), small monitoring windows (*i.e.,* 2s, 5s) gives higher portion of low coverage flows in the future, which reduce the probability of finding high coverage flows since flow watch list is generated by the set of flow candidates. However, the performance of monitoring windows $T = 10s, 15s$ are similar, but $T = 10s$ is more sensitive since it can provide faster update on flow watch list than $T = 15s$, and the monitoring window T can be gradually increased after the flow watch list is stable. The figure 4.15(b) shows the flow size distribution in the future 10s with different monitoring window. The smaller monitoring window



(a) #. flow candidates (b) Bandwidth reduction (c) Futuer coverage rate C_f

Figure 4.17: The *OINT* performance and resource consumption. (a) #. flow candidates with memory configurations. (b) Minimum bandwidth reduction compared with original INT. (c) C_f provided by the *OINT* in the future 10s.

size also has more small flows, which cannot provide a fine-grained monitoring (*e.g.,* $T = 2$ has around only 40 telemetry pps in average (red dots)). Therefore, a $T = 10s$ monitoring window size is chosen at the start for the *OINT*.

4.6.3 The *OINT* performance and resource consumption

4.6.3.1 flow size distribution

With selected flow candidates as the INT flows after the current monitoring window, we collected the INT flows statistics (*i.e.,* 5-tuple, packet count, *etc.*) in the next monitoring window to see if those INT flows provide a fine-grained monitoring with different configurations of the *OINT*. We divide the optimizations into three categories as follows.

- ***OINT***$_F$. The *OINT* with filter stage enabled.
- ***OINT***$_{FC}$. Filter stage and hash table compression enabled.
- ***OINT***$_{FC}\Delta = \delta$. *OINT*$_{FC}$ with optimization on selecting flow candidates at monitoring host.

The figure 4.16(a) shows the flow size distribution of flow candidates selected under different *OINT* configurations. The average flow size of a flow is calculated by the mean of flow size observed over all monitoring intervals. As we can see from the figure 4.16(a), with small optimization over the *OINT*, we captured a wide range of flow sizes including small flows (*i.e.,* $v_f \leq 100$), while restricted sets of flow candidates are selected with optimization increased (*i.e.,* hash table

compression filtered small flows and monitoring host further reduces the flow candidates size), which is what we expected as the goal is to select flows with ideal monitoring rate (*i.e.,* sufficient flow size). Although the larger memory (*i.e.,* 8 MB) allocated for the *OINT* sink is capable of capturing many small flows, the performance on larger flow sizes is similar to the small memory allocation. As we can also see from the figure 4.17(a) that the number of flow candidates is similar with more optimizations (*i.e.,* *OINT*$_{FC}$, *OINT*$_{FC}\Delta$) enabled under different memory allocation for the *OINT* sink, which filtered out the potential small flows. Figure 4.16(b) shows the flow size distribution of flow candidates performed in the future 10s, and we observed that those flow candidates also provided expected flow sizes (*i.e.,* under different optimization configurations) in the future 10s.

4.6.3.2 Coverage rate C_f

The figure 4.17(c) provides the future coverage rate C_f of the flow candidates under different optimizations. We described the coverage rate distribution provided by flow candidates in the future 10s in figure 4.17(c) by kernel density estimation [134]. The majority ($\sim 86\%$) of flows candidates provides consistent network monitoring $C_f = 1.0$ while a small portion ($\sim 0.9\%$) of flows has $C_f = 0$, which can be eliminated in the next monitoring window by updating the flow watch list. By having such set of flow candidates monitoring the network, we can collect a consistent monitoring as the original INT approach (*i.e.,* enabling all packets as INT packets).

4.6.3.3 Bandwidth consumption

However, the *OINT* reduce the bandwidth consumption by the INT header and INT metadata dramatically compared the the original INT. The figure 4.17(b) shows the amount of bandwidth reduction by enabling different optimizations over *OINT*. With filter stage enabled at the report unit, we can achieve minimum (*i.e.,* as flows in flow watch list is subset of the flow candidates, and here we assume we directly write all flow candidates into each INT source) around 70% bandwidth reduction on the INT information compared with the original approach. With more restricted constrains such as the $\Delta = 500$, we filtered out more small flows, the minimum bandwidth reduction

can achieve around 80%.

4.6.3.4 Threads / cores usage

The flow statistics (*i.e.,* normal packets, INT packets or events) are sent to the monitoring host by DMA supported by a high transmission rate (*i.e.,* multiple 10 $Gbps$ based on PCIe bandwidth). The monitoring host allocate 1 core for processing the incoming flow statistics from the *OINT* sink and another core for writing records into Redis database [74].

4.7 Limitations

Coverage of short-lived flows. The *OINT* aims to provide a fine-grained network monitoring with healthy long-live flows tracked by the *OINT* sink and the monitoring host. The process of generating flow watch lists for each INT source by the set of flow candidates might not achieve 100% coverage rate of a given network if the traffic on one of the paths does not contain any long-lived flows satisfied the coverage rate constrain (*e.g.,* $C_f = 1.0$). However, this problem can be potentially solved by the feedback mechanism of the *OINT*, which continuously keeps tracking of the INT flow statistics to see the coverage by inspecting collected switch information and increases the number of flow candidates by lower the coverage rate to find the set of flows traversing the uncovered paths.

Path change of normal traffic. Another limitations of the *OINT* is the path change events of normal traffic. Although the *OINT* is capable of tracking INT flow path change events by inspecting the collected switch metrics, it cannot conduct such path change events for the normal traffic, which do not have switch metrics embedded inside. However, the *OINT* can specifically monitor a small set of flows of interest by using the similar method of path recovery, which inspects the bloom filter of each INT-enabled switches, to check if path change happened for any of those flows.

4.8 Related Work

Traditional INT, which enables all packets traversing through the INT sources as INT packets, has been studied by many researchers [105, 135, 136, 137, 138, 108, 139, 140, 141, 142]. INT

monitoring architectures [139, 143, 102] present the design of processing INT flows and mechanism on reporting INT events. IntCollector [139] handled INT information with slow and fast path, which are responsible for processing INT packets and storing INT report respectively using software NIC and host CPU. Similar to IntCollector, Hyun *et al.*[144] presented an INT management system over IntColletor with the management system controls heterogeneous INT-enabled devices through a common interface. Vestin *et al.*[143] designed an INT monitoring platform using Netronome smartNIC p4 pipeline, which supports simple threshold-crossing INT events to the stream processor using the kernel bypass technique AF XDP. With C funtions provided by smartNIC, the processing latency on INT packets and a wide range of INT events can be further reduced and more flexibilities can be provided [102].

However, the overall bandwidth consumed by the additional INT information is the main drawback of the traditional INT approach [101, 100], thus optimizations over original INT are further investigated by researchers [108, 101, 145, 109]. Sampling approaches [104, 105, 106, 107] aim to select subset of network traffic as INT packets to carry network measurements to reduce the bandwidth consumption. INT-filter [146] reduced unnecessary uploaded INT telemetry data by historical prediction. Marques *et al.*[108] studied the optimization problems over bandwidth by minimizing the number of active (*i.e.,* INT capable) telemetry flows and number of telemetry items carried by INT flows in order to cover each interface of a network. However, those heuristic solutions are based on assumptions that network flows are stable overtime and measurements on different paths are the same, which might be different in a realistic network setup. Different from the original INT approach using existing traffic, Castro *et al.*[109] used active probing packets to minimize the number of probing cycles in order to cover the whole network and PINT framework [101] designed a probabilistic data structure to encode switch measurements into network packets and decode at the sink, which might have a coarse monitoring granularity and high complexity.

The proposed *OINT* monitoring platform provides a solution for fine-grained monitoring with high network coverage and small bandwidth consumption. The *OINT* uses bloom filters for path recovery of flow candidates to install flow watch list at each INT source. The efficient data struc-

ture of bloom filter has been applied on many network applications [130, 147, 148] such as routing mechanism [149, 150], flow statistics estimation and tracking [117, 151], *etc.* . Chen *et al.*[117] developed a two-stage bloom filters mechanism to track long duration flows with offline evaluations but it might require large space to store flow records, which is difficult to implement in commodity switches. The long-lived flow tracking mechanism of the *OINT* leverages the capabilities of *OINT* sink and monitoring host to achieve low latency and high packet processing rate.

5. SUMMARY AND CONCLUSIONS

This research work provides methods on monitoring network internal statistics with a network tomography way of monitoring and a novel idea of In-band Network Telemetry. Both ways of monitoring do not involve explicitly install additional monitoring hardware inside networks and also bypass the complexity of directly accessing point connections of networks.

Mutiscale analysis using Discrete Wavelet Transforms enable us to characterize the complex properties of network traffic by extracting the details across frequencies and provides valuable statistics in differentiating the normal patterns and the anomalies. We proposed an unbiased estimator on the multi-scale energy of internal links based on the tomography of end-to-end measurements. From the model simulations of the canonical two leaf tree, we showed the how the estimator of the path intersection is constructed and generalized to more complex networks with potentially asymmetric routing. We also explored how to localize the anomalies targeting at internal links use the estimator from the RTT measurements. In a simple experimental demonstration we showed how this approach may be applied to detecting the network attacks on internal network links through changes or signatures in the inferred energy spectrum of metric values, in our case link packet delay.

Most recent proposed In-band Network Telemetry enables collect network internal states completely within data plane. Efficiently collecting INT traffic statistics at an INT sink, in a loss-free manner and generating notifications without impacting throughput is crucial for an INT network monitoring platform. To strike a balance between having more complex INT metric collection and maintaining a high throughput, we proposed a smartNIC-based host as an INT sink and monitoring platform. Unlike using a P4 switch as an INT sink, the sNIC is able to perform INT-packet processing at high rates as well as the complex statistics collection tasks in a loss-free manner. The packet processing pipeline in the sNIC combines the packet header extraction using P4, and callable C functions running on micro-engines to achieve the high performance we desire. While INT traffic is aggregated on a large hash table on the sNIC, the INT events are exported to a set of ring

buffers retrieved by the monitoring host. The history of INT events and aggregated INT metrics are stored using in-memory database in the monitoring host. Our evaluations show that the monitoring platform achieves full line-rate throughput through the sNIC, high event notification rates, and high accuracy for traffic statistics collection. The latency incurred by our INT-sink and monitoring platform is quite low–a few μsecs–so the platform can function as a 'bump-in-the-wire'.

In order to reduce the bandwidth consumption of the INT, the *OINT* provides a way of inferring flow paths of subset of network traffic by having bloom filters installed inside each INT-enabled switches over a network, which does not introduce any packet overhead compared with original INT approach (*i.e.,* record switch IDs along a flow path). With several optimizations (*i.e.,* hash table compression, flow candidates reduction) enabled, the *OINT* can also reduced the communication overheads between the INT sink and the monitoring host. We showed that, by keeping tracking of the potential long-lived flows, we are able to provide a fine-grained (*e.g.,* 100 telemetry items per second) network monitoring for each network internal elements solution without consuming too much network available bandwidth. Compared to the original approach, the *OINT* is able to reduce the bandwidth consumption by at least 80%.

REFERENCES

[1] D. Harrington, R. Presuhn, and B. Wijnen, "An architecture for describing snmp management frameworks," 04 1999.

[2] S. Waldbusser, R. Cole, C. Kalbfleisch, and D. Romascanu, "Rfc3577: Introduction to the remote monitoring (rmon) family of mib modules," 2003.

[3] B. Claise, "Cisco systems netflow services export version 9," RFC 3954, RFC Editor, October 2004. `http://www.rfc-editor.org/rfc/rfc3954.txt`.

[4] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, p. 69–74, Mar. 2008.

[5] N. Duffield, J. Horowitz, F. Lo Presti, and D. Towsley, "Network delay tomography from end-to-end unicast measurements," vol. 2170, pp. 576–595, 09 2001.

[6] N. G. Duffield and F. Lo Presti, "Network tomography from measured end-to-end delay covariance," *IEEE/ACM Transactions on Networking*, vol. 12, no. 6, pp. 978–992, 2004.

[7] S. Tang, D. Li, B. Niu, J. Peng, and Z. Zhu, "Sel-INT: A runtime-programmable selective in-band network telemetry system," *IEEE Transactions on Network and Service Management*, vol. PP, 11 2019.

[8] Y. Kim, D. Suh, and S. Pack, "Selective in-band network telemetry for overhead reduction," pp. 1–3, 10 2018.

[9] J. Marques, M. Caggiani Luizelli, R. Iraja Tavares da Costa Filho, and L. Gaspary, "An optimization-based approach for efficient network monitoring using in-band network telemetry," *Journal of Internet Services and Applications*, vol. 10, 12 2019.

[10] D. Suh, S. Jang, S. Hanb, S. Pack, and X. Wang, "Flexible sampling-based in-band network telemetry in programmable data plane," *ICT Express*, vol. 6, 09 2019.

[11] J. Liang, J. Bi, Y. Zhou, and C. Zhang, "In-band network function telemetry," pp. 42–44, 08 2018.

[12] D. Bh, A. Kassler, J. Vestin, M. A. Khoshkholghi, and J. Taheri, "IntOpt: In-band network telemetry optimization for nfv service chain monitoring," pp. 1–7, 05 2019.

[13] N. Tu, J. Hyun, and J. Hong, "Towards onos-based sdn monitoring using in-band network telemetry," pp. 76–81, 09 2017.

[14] N. Van Tu, J. Hyun, G. Y. Kim, J.-H. Yoo, and J. W.-K. Hong, "Intcollector: A high-performance collector for in-band network telemetry," in *2018 14th International Conference on Network and Service Management (CNSM)*, pp. 10–18, IEEE, 2018.

[15] J. Vestin, A. Kassler, D. Bh, K.-j. Grinnemo, J.-O. Andersson, and G. Pongracz, "Programmable event detection for in-band network telemetry," 09 2019.

[16] "P4 data plane programming for server-based networking applications."

[17] A. Gilbert, "Multiscale analysis and data networks," *Applied and Computational Harmonic Analysis*, vol. 10, no. 3, pp. 185 – 202, 2001.

[18] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of ethernet traffic (extended version)," *IEEE/ACM Trans. Netw.*, vol. 2, pp. 1–15, Feb. 1994.

[19] A. Feldmann, A. C. Gilbert, and W. Willinger, "Data networks as cascades: Investigating the multifractal nature of internet wan traffic," pp. 42–55, 1998.

[20] C.-T. Huang, S. Thareja, and Y.-J. Shin, "Wavelet-based real time detection of network traffic anomalies," *2006 Securecomm and Workshops*, pp. 1–7, 2006.

[21] A. Dainotti, A. Pescapè, and G. Ventre, "A cascade architecture for dos attacks detection based on the wavelet transform," *Journal of Computer Security*, vol. 17, pp. 945–968, 2009.

[22] F. H. V. Teles and L. L. Ling, "Adaptive wavelet-based multifractal model applied to the effective bandwidth estimation of network traffic flows," *IET Communications*, vol. 3, pp. 906–919, 2009.

[23] R. Mathew and V. Katkar, "Survey of low rate dos attack detection mechanisms," in *Proceedings of the International Conference & Workshop on Emerging Trends in Technology*, ICWET '11, (New York, NY, USA), p. 955–958, Association for Computing Machinery, 2011.

[24] E. Cambiaso, G. Papaleo, and M. Aiello, "Slowcomm: Design, development and performance evaluation of a new slow dos attack," *Journal of Information Security and Applications*, vol. 35, pp. 23–31, 08 2017.

[25] A. Kuzmanovic and E. W. Knightly, "Low-rate tcp-targeted denial of service attacks: The shrew vs. the mice and elephants," in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, (New York, NY, USA), p. 75–86, Association for Computing Machinery, 2003.

[26] G. Maciá-Fernández, J. Díaz-Verdejo, P. García-Teodoro, and F. Toro-Negro, "Lordas: A low-rate dos attack against application servers," vol. 5141, pp. 197–209, 10 2007.

[27] "Ns3."

[28] I. Daubechies and C. Heil, "Ten lectures on wavelets," *Computers in Physics*, vol. 6, pp. 697–, 11 1992.

[29] N. G. Duffield and F. L. Presti, "Network tomography from measured end-to-end delay covariance," *IEEE/ACM Trans. Netw.*, vol. 12, no. 6, pp. 978–992, 2004.

[30] G. Berkolaiko, N. Duffield, M. Ettehad, and K. Manousakis, "Graph reconstruction from path correlation data." 2018.

[31] D. Percival, "On estimation of the wavelet variance," *Biometrika*, vol. 82, 01 2012.

[32] "Internet2."

[33] T. C. U. I. R. . T. Dataset, 2019.

[34] M. Luckie, "Scamper: A scalable and extensible packet prober for active measurement of the internet," pp. 239–245, 01 2010.

[35] M. J. Coates and R. D. Nowak, "Sequential monte carlo inference of internal delays in non-stationary data networks," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 366–376, 2002.

[36] N. Duffield, F. Lo Presti, V. Paxson, and D. Towsley, "Network loss tomography using striped unicast probes," *IEEE/ACM Transactions on Networking*, vol. 14, no. 4, pp. 697–710, 2006.

[37] Y. Feng, S. Panda, S. G. Kulkarni, K. K. Ramakrishnan, and N. Duffield, "A smartnic-accelerated monitoring platform for in-band network telemetry," in *2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN*, pp. 1–6, 2020.

[38] B. Yu, J. Cao, D. Davis, and S. V. Wiel, "Time-varying network tomography: router link data," in *2000 IEEE International Symposium on Information Theory (Cat. No.00CH37060)*, pp. 79–, 2000.

[39] R. Caceres, N. Duffield, J. Horowitz, and D. F. Towsley, "Multicast based inference of network internal loss characteristics," *IEEE Transactions on Information Theory*, vol. 45, no. 7, pp. 2462–2480, Nov. 1999.

[40] N. G. Duffield, J. Horowitz, F. L. Presti, and D. Towsley, "Multicast topology inference from measured end-to-end loss," *IEEE Transactions on Information Theory*, vol. 48, pp. 26–45, Jan 2002.

[41] N. G. Duffield, J. Horowitz, F. L. Presti, and D. F. Towsley, "Explicit loss inference in multicast tomography," *IEEE Transactions on Information Theory*, vol. 52, no. 8, pp. 3852–3855, 2006.

[42] Yolanda Tsang, M. Coates, and R. D. Nowak, "Network delay tomography," *IEEE Transactions on Signal Processing*, vol. 51, no. 8, pp. 2125–2136, 2003.

[43] V. Arya, N. G. Duffield, and D. Veitch, "Multicast inference of temporal loss characteristics," *Perform. Eval.*, vol. 64, no. 9-12, pp. 1169–1180, 2007.

[44] V. Arya, N. G. Duffield, and D. Veitch, "Temporal delay tomography," in *INFOCOM 2008. 27th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 13-18 April 2008, Phoenix, AZ, USA*, pp. 276–280, 2008.

[45] N. G. Duffield, "Network tomography of binary network performance characteristics," *IEEE Transactions on Information Theory*, vol. 52, no. 12, pp. 5373–5388, 2006.

[46] V. Arya and D. Veitch, "Sparsity without the complexity: Loss localisation using tree measurements," in *NETWORKING 2012* (R. Bestak, L. Kencl, L. E. Li, J. Widmer, and H. Yin, eds.), (Berlin, Heidelberg), pp. 289–303, Springer Berlin Heidelberg, 2012.

[47] H. X. Nguyen and P. Thiran, "Network loss inference with second order statistics of end-to-end flows," in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, (New York, NY, USA), pp. 227–240, ACM, 2007.

[48] C. Callegari, M. Pagano, S. Giordano, and T. Pepe, "Combining wavelet analysis and information theory for network anomaly detection," in *Proceedings of the 4th International Symposium on Applied Sciences in Biomedical and Communication Technologies*, ISABEL '11, (New York, NY, USA), Association for Computing Machinery, 2011.

[49] J. Gao, G. Hu, X. Yao, and R. K. C. Chang, "Anomaly detection of network traffic based on wavelet packet," in *2006 Asia-Pacific Conference on Communications*, pp. 1–5, 2006.

[50] S. S. Kim and A. L. N. Reddy, "Statistical techniques for detecting traffic anomalies through packet header data," *IEEE/ACM Trans. Netw.*, vol. 16, p. 562–575, June 2008.

[51] L. Li and G. Lee, "Ddos attack detection and wavelets," vol. 28, pp. 421– 427, 11 2003.

[52] A. Dainotti, A. Pescapè, and G. Ventre, "Wavelet-based detection of dos attacks.," 11 2006.

[53] A. Lakhina, M. Crovella, and C. Diot, "Diagnosing network-wide traffic anomalies," *SIGCOMM Comput. Commun. Rev.*, vol. 34, p. 219–230, Aug. 2004.

[54] A. Lakhina, M. Crovella, and C. Diot, "Mining anomalies using traffic feature distributions," in *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '05, (New York, NY, USA), p. 217–228, Association for Computing Machinery, 2005.

[55] B.-S. Liu, Y.-J. Li, Y.-P. Hou, and X.-S. Sui, "The identification and correction of outlier based on wavelet transform of traffic flow," pp. 1498 – 1503, 12 2007.

[56] W. Lu and A. Ghorbani, "Network anomaly detection based on wavelet analysis," *EURASIP J. Adv. Sig. Proc.*, vol. 2009, 01 2009.

[57] A. Hussain, J. Heidemann, and C. Papadopoulos, "A framework for classifying denial of service attacks," in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, (New York, NY, USA), p. 99–110, Association for Computing Machinery, 2003.

[58] P. Barford, J. Kline, D. Plonka, and A. Ron, "A signal analysis of network traffic anomalies," *Proceedings of the 2nd Internet Measurement Workshop (IMW 2002)*, 09 2002.

[59] X. Luo and R. Chang, "On a new class of pulsing denial-of-service attacks and the defense.," 01 2005.

[60] Z. Wu, R. Hu, and M. Yue, "Flow-oriented detection of low-rate denial of service attacks[j]," *International Journal of Communication Systems*, vol. 29, 04 2014.

[61] Y. Chen, K. Hwang, and Y.-K. Kwok, "Collaborative defense against periodic shrew ddos attacks in frequency domain," *ACM Transactions on Information and System Security*, vol. 3, 01 2005.

[62] M. Yue, L. Liu, Z. Wu, and M. Wang, "Identifying ldos attack traffic based on wavelet energy spectrum and combined neural network," *International Journal of Communication Systems*, vol. 31, 10 2017.

[63] "In-band Network Telemetry (INT)."

[64] W. Stallings, *SNMP, SNMP v2, SNMP v3, and RMON 1 and 2 (Third Edition)*. Reading, Mass.: Addison-Wesley, 1999.

[65] "P4 language specification."

[66] Y. Tokusashi, H. T. Dang, F. Pedone, R. Soulé, and N. Zilberman, "The case for in-network computing on demand," in *Proceedings of the Fourteenth EuroSys Conference 2019*, EuroSys '19, (New York, NY, USA), Association for Computing Machinery, 2019.

[67] J. Hyun, N. Tu, J.-H. Yoo, and J. Hong, "Real-time and fine-grained network monitoring using in-band network telemetry," *International Journal of Network Management*, vol. 29, p. e2080, 10 2019.

[68] "In-band network telemetry (int) dataplane specification v2.0."

[69] T. Zhang, L. Linguaglossa, M. Gallo, P. Giaccone, and D. Rossi, "Flowatcher-dpdk: Lightweight line-rate flow-level monitoring in software," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 1143–1156, 2019.

[70] "The CAIDA anonymized internet traces." `http://www.caida.org/data/passive/passive_dataset.xml`.

[71] "Netronome NFP-4000 Flow Processor."

[72] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, (New York, NY, USA), pp. 267–280, ACM, 2010.

[73] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "MoonGen: A Scriptable High-Speed Packet Generator," in *Internet Measurement Conference 2015 (IMC'15)*, (Tokyo, Japan), Oct. 2015.

[74] "Redis."

[75] A. Gupta, R. Harrison, M. Canini, N. Feamster, J. Rexford, and W. Willinger, "Sonata: Query-driven streaming network telemetry," in *Proceedings of the 2018 Conference of the*

ACM Special Interest Group on Data Communication, SIGCOMM '18, (New York, NY, USA), pp. 357–371, ACM, 2018.

[76] Y. Yuan, D. Lin, A. Mishra, S. Marwaha, R. Alur, and B. T. Loo, "Quantitative network monitoring with netqre," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, (New York, NY, USA), p. 99–112, Association for Computing Machinery, 2017.

[77] "Opensoc."

[78] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk, "Gigascope: A stream database for network applications," in *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, (New York, NY, USA), p. 647–651, Association for Computing Machinery, 2003.

[79] Q. Huang, H. Sun, P. P. C. Lee, W. Bai, F. Zhu, and Y. Bao, "Omnimon: Re-architecting network telemetry with resource efficiency and full accuracy," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, (New York, NY, USA), p. 404–421, Association for Computing Machinery, 2020.

[80] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "Pint: Probabilistic in-band network telemetry," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, (New York, NY, USA), p. 662–680, Association for Computing Machinery, 2020.

[81] X. Chen, S. Landau-Feibish, M. Braverman, and J. Rexford, "Beaucoup: Answering many network traffic queries, one memory update at a time," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, (New York, NY, USA), p. 226–239, Association for Computing Machinery, 2020.

[82] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, and C. Kim, "Language-directed hardware design for network performance monitoring," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, (New York, NY, USA), p. 85–98, Association for Computing Machinery, 2017.

[83] Y. Li, R. Miao, C. Kim, and M. Yu, "Flowradar: A better netflow for data centers," in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, (Santa Clara, CA), pp. 311–324, USENIX Association, Mar. 2016.

[84] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, "Elastic sketch: Adaptive and fast network-wide measurements," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, (New York, NY, USA), pp. 561–575, ACM, 2018.

[85] J. Sonchack, A. J. Aviv, E. Keller, and J. M. Smith, "Turboflow: Information rich flow record generation on commodity switches," in *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, (New York, NY, USA), pp. 11:1–11:16, ACM, 2018.

[86] M. Moshref, M. Yu, R. Govindan, and A. Vahdat, "Trumpet: Timely and precise triggers in data centers," in *Proceedings of the 2016 ACM SIGCOMM Conference*, SIGCOMM '16, (New York, NY, USA), p. 129–143, Association for Computing Machinery, 2016.

[87] P. Tammana, R. Agarwal, and M. Lee, "Simplifying datacenter network debugging with pathdump," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, (Savannah, GA), pp. 233–248, USENIX Association, Nov. 2016.

[88] L. Tang, Q. Huang, and P. P. C. Lee, "Mv-sketch: A fast and compact invertible sketch for heavy flow detection in network data streams," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pp. 2026–2034, April 2019.

[89] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with univmon," in *Proceedings of the 2016 ACM*

SIGCOMM Conference, SIGCOMM '16, (New York, NY, USA), pp. 101–114, ACM, 2016.

[90] Q. Huang, P. Lee, and Y. Bao, "Sketchlearn: relieving user burdens in approximate measurement with automated statistical inference," pp. 576–590, 08 2018.

[91] Z. Liu, R. Ben-Basat, G. Einziger, Y. Kassner, V. Braverman, R. Friedman, and V. Sekar, "Nitrosketch: Robust and general sketch-based monitoring in software switches," in *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, (New York, NY, USA), pp. 334–350, ACM, 2019.

[92] C. Guo, L. Yuan, D. Xiang, Y. Dang, R. Huang, D. Maltz, Z. Liu, V. Wang, B. Pang, H. Chen, Z.-W. Lin, and V. Kurien, "Pingmesh: A large-scale system for data center network latency measurement and analysis," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, (New York, NY, USA), p. 139–152, Association for Computing Machinery, 2015.

[93] V. Jeyakumar, M. Alizadeh, Y. Geng, C. Kim, and D. Mazières, "Millions of little minions: Using packets for low latency network programming and visibility," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, (New York, NY, USA), p. 3–14, Association for Computing Machinery, 2014.

[94] L. Quan and J. Heidemann, "On the characteristics and reasons of long-lived internet flows," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, (New York, NY, USA), p. 444–450, Association for Computing Machinery, 2010.

[95] Y. Meidan, M. Bohadana, A. Shabtai, J. D. Guarnizo, M. Ochoa, N. O. Tippenhauer, and Y. Elovici, "Profiliot: A machine learning approach for iot device identification based on network traffic analysis," in *Proceedings of the Symposium on Applied Computing*, SAC '17, (New York, NY, USA), p. 506–509, Association for Computing Machinery, 2017.

[96] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 92–99, 2010.

[97] D. Harrington, R. Presuhn, and B. Wijnen, "Rfc3411: An architecture for describing simple network management protocol (snmp) management frameworks," 2002.

[98] S. Waldbusser, R. Cole, C. Kalbfleisch, and D. Romascanu, "Rfc3577: Introduction to the remote monitoring (rmon) family of mib modules," 2003.

[99] K. G. Anagnostakis, S. Ioannidis, S. Miltchev, M. Greenwald, J. M. Smith, and J. Ioannidis, "Efficient packet monitoring for network management," in *NOMS 2002. IEEE/IFIP Network Operations and Management Symposium. ' Management Solutions for the New Communications World'(Cat. No.02CH37327)*, pp. 423–436, 2002.

[100] L. Tan, W. Su, W. Zhang, J. Lv, Z. Zhang, J. Miao, X. Liu, and N. Li, "In-band network telemetry: A survey," *Computer Networks*, vol. 186, p. 107763, 2021.

[101] R. Ben Basat, S. Ramanathan, Y. Li, G. Antichi, M. Yu, and M. Mitzenmacher, "Pint: Probabilistic in-band network telemetry," in *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, (New York, NY, USA), p. 662–680, Association for Computing Machinery, 2020.

[102] Y. Feng, S. Panda, S. G. Kulkarni, K. K. Ramakrishnan, and N. Duffield, "A smartnic-accelerated monitoring platform for in-band network telemetry," in *2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN*, pp. 1–6, 2020.

[103] R. Oudin, G. Antichi, C. Rotsos, A. W. Moore, and S. Uhlig, "Oflops-sume and the art of switch characterization," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 12, pp. 2612–2620, 2018.

[104] D. Suh, S. Jang, S. Han, S. Pack, and X. Wang, "Flexible sampling-based in-band network telemetry in programmable data plane," *ICT Express*, vol. 6, no. 1, pp. 62–65, 2020.

[105] S. Tang, D. Li, B. Niu, J. Peng, and Z. Zhu, "Sel-INT: A runtime-programmable selective in-band network telemetry system," *IEEE Transactions on Network and Service Management*, vol. PP, 11 2019.

[106] S. Tang, J. Kong, B. Niu, and Z. Zhu, "Programmable multilayer int: An enabler for ai-assisted network automation," *IEEE Communications Magazine*, vol. 58, no. 1, pp. 26–32, 2020.

[107] T. Pan, E. Song, C. Jia, W. Cao, T. Huang, and B. Liu, "Lightweight network-wide telemetry without explicitly using probe packets," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 1354–1355, IEEE, 2020.

[108] J. Marques, M. Caggiani Luizelli, R. Iraja Tavares da Costa Filho, and L. Gaspary, "An optimization-based approach for efficient network monitoring using in-band network telemetry," *Journal of Internet Services and Applications*, vol. 10, 12 2019.

[109] A. G. Castro, A. F. Lorenzon, F. D. Rossi, R. I. Da Costa Filho, F. M. Ramos, C. E. Rothenberg, and M. C. Luizelli, "Near-optimal probing planning for in-band network telemetry," *IEEE Communications Letters*, 2021.

[110] A. Kuzmanovic and E. W. Knightly, "Low-rate tcp-targeted denial of service attacks: The shrew vs. the mice and elephants," in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '03, (New York, NY, USA), p. 75–86, Association for Computing Machinery, 2003.

[111] P. Barford, J. Kline, D. Plonka, and A. Ron, "A signal analysis of network traffic anomalies," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pp. 71–82, 2002.

[112] Y. Gu, A. McCallum, and D. Towsley, "Detecting anomalies in network traffic using maximum entropy estimation," in *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pp. 32–32, 2005.

[113] C. Estan, S. Savage, and G. Varghese, "Automatically inferring patterns of resource consumption in network traffic," in *Proceedings of the 2003 Conference on Applications, Tech-*

nologies, Architectures, and Protocols for Computer Communications, SIGCOMM '03, (New York, NY, USA), p. 137–148, Association for Computing Machinery, 2003.

[114] A. Shaikh, J. Rexford, and K. G. Shin, "Load-sensitive routing of long-lived ip flows," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '99, (New York, NY, USA), p. 215–226, Association for Computing Machinery, 1999.

[115] S. Ghorbani, Z. Yang, P. B. Godfrey, Y. Ganjali, and A. Firoozshahian, "Drill: Micro load balancing for low-latency data center networks," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, (New York, NY, USA), p. 225–238, Association for Computing Machinery, 2017.

[116] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, (New York, NY, USA), p. 267–280, Association for Computing Machinery, 2010.

[117] A. Chen, Y. Jin, J. Cao, and L. E. Li, "Tracking long duration flows in network traffic," in *2010 Proceedings IEEE INFOCOM*, pp. 1–5, 2010.

[118] S. Singh and S. Tirthapura, "Monitoring persistent items in the union of distributed streams," *Journal of Parallel and Distributed Computing*, vol. 74, 11 2014.

[119] M. Enriquez, "Identifying temporally persistent flows in the terminal airspace via spectral clustering," 10 2013.

[120] F. Giroire, J. Chandrashekar, N. Taft, E. Schooler, and D. Papagiannaki, "Exploiting temporal persistence to detect covert botnet channels," vol. 5758, pp. 326–345, 09 2009.

[121] H. Dai, M. Shahzad, A. Liu, M. Li, Y. Zhong, and G. Chen, "Identifying and estimating persistent items in data streams," *IEEE/ACM Transactions on Networking*, vol. PP, pp. 1–14, 09 2018.

[122] B. Lahiri, S. Tirthapura, and J. Chandrashekar, "Space-efficient tracking of persistent items in a massive data stream," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 7, no. 1, pp. 70–92, 2014.

[123] L. Zhang and Y. Guan, "Detecting click fraud in pay-per-click streams of online advertising networks," in *2008 The 28th International Conference on Distributed Computing Systems*, pp. 77–84, IEEE, 2008.

[124] J. Gadge and A. A. Patil, "Port scan detection," in *2008 16th IEEE International Conference on Networks*, pp. 1–6, IEEE, 2008.

[125] S. Staniford, J. A. Hoagland, and J. M. McAlerney, "Practical automated detection of stealthy portscans," *Journal of Computer Security*, vol. 10, no. 1-2, pp. 105–136, 2002.

[126] X. Luo, R. K. Chang, *et al.*, "On a new class of pulsing denial-of-service attacks and the defense.," in *NDSS*, 2005.

[127] R. Mathew and V. Katkar, "Survey of low rate dos attack detection mechanisms," in *Proceedings of the International Conference & Workshop on Emerging Trends in Technology*, ICWET '11, (New York, NY, USA), p. 955–958, Association for Computing Machinery, 2011.

[128] "Telemetry report format specification v1.0." `https://github.com/p4lang/p4-applications/blob/master/docs/telemetry_report_v1_0.pdf`, 2018.

[129] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, p. 422–426, July 1970.

[130] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet mathematics*, vol. 1, no. 4, pp. 485–509, 2004.

[131] "Netronome smartnics specification." `https://www.netronome.com/products/agilio-cx/`.

[132] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin, A. Ghodsi, J. Gonzalez, S. Shenker, and I. Stoica, "Apache spark: A unified engine for big data processing," *Commun. ACM*, vol. 59, p. 56–65, Oct. 2016.

[133] "Apache thrift."

[134] M. Waskom and the seaborn development team, "mwaskom/seaborn," Sept. 2020.

[135] Y. Kim, D. Suh, and S. Pack, "Selective in-band network telemetry for overhead reduction," pp. 1–3, 10 2018.

[136] D. Bh, A. Kassler, J. Vestin, M. A. Khoshkholghi, and J. Taheri, "IntOpt: In-band network telemetry optimization for nfv service chain monitoring," pp. 1–7, 05 2019.

[137] J. Liang, J. Bi, Y. Zhou, and C. Zhang, "In-band network function telemetry," pp. 42–44, 08 2018.

[138] D. Suh, S. Jang, S. Hanb, S. Pack, and X. Wang, "Flexible sampling-based in-band network telemetry in programmable data plane," *ICT Express*, vol. 6, 09 2019.

[139] N. Tu, J. Hyun, and J. Hong, "Towards onos-based sdn monitoring using in-band network telemetry," pp. 76–81, 09 2017.

[140] N. Van Tu, J. Hyun, G. Y. Kim, J.-H. Yoo, and J. W.-K. Hong, "Intcollector: A high-performance collector for in-band network telemetry," in *2018 14th International Conference on Network and Service Management (CNSM)*, pp. 10–18, IEEE, 2018.

[141] A. Gulenko, M. Wallschläger, and O. Kao, "A practical implementation of in-band network telemetry in open vswitch," in *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*, pp. 1–4, 2018.

[142] C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, "In-band network telemetry via programmable dataplanes," in *ACM SIGCOMM*, vol. 15, 2015.

[143] J. Vestin, A. Kassler, D. Bh, K.-j. Grinnemo, J.-O. Andersson, and G. Pongracz, "Programmable event detection for in-band network telemetry," 09 2019.

[144] J. Hyun, N. Van Tu, J.-H. Yoo, and J. W.-K. Hong, "Real-time and fine-grained network monitoring using in-band network telemetry," *International Journal of Network Management*, vol. 29, no. 6, p. e2080, 2019.

[145] T. Pan, E. Song, Z. Bian, X. Lin, X. Peng, J. Zhang, T. Huang, B. Liu, and Y. Liu, "Int-path: Towards optimal path planning for in-band network-wide telemetry," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 487–495, IEEE, 2019.

[146] E. Song, T. Pan, C. Jia, W. Cao, J. Zhang, T. Huang, and Y. Liu, "Int-filter: Mitigating data collection overhead for high-resolution in-band network telemetry," in *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pp. 1–6, IEEE, 2020.

[147] S. Geravand and M. Ahmadi, "Bloom filter applications in network security: A state-of-the-art survey," *Computer Networks*, vol. 57, no. 18, pp. 4047–4064, 2013.

[148] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood, "Fast hash table lookup using extended bloom filter: an aid to network processing," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 4, pp. 181–192, 2005.

[149] A. Marandi, V. Hofer, M. Gasparyan, T. Braun, and N. Thomos, "Bloom filter-based routing for dominating set-based service-centric networks," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9, IEEE, 2020.

[150] W. Gao, J. Nguyen, Y. Wu, W. G. Hatcher, and W. Yu, "A bloom filter-based dual-layer routing scheme in large-scale mobile networks," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–9, IEEE, 2017.

[151] A. Kumar, J. Xu, and J. Wang, "Space-code bloom filter for efficient per-flow traffic measurement," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 12, pp. 2327–2339, 2006.