AMON: An Open Source Architecture for Online Monitoring, Statistical Analysis and Forensics of Multi-gigabit Streams

Michael Kallitsis, Stilian Stoev, Shrijita Bhattacharya, George Michailidis
mgkallit@merit.edu, sstoev@umich.edu, shrijita@umich.edu, gmichail@ufl.edu

Abstract

The Internet, as a global system of interconnected networks, carries an extensive array of information resources and services. Key requirements include good quality-of-service and protection of the infrastructure from nefarious activity (e.g. distributed denial of service—DDoS—attacks). Network monitoring is essential to network engineering, capacity planning and prevention / mitigation of threats. We develop an open source architecture, AMON (All-packet MONitor), for online monitoring and analysis of multi-gigabit network streams. It leverages the high-performance packet monitor PF_RING and is readily deployable on commodity hardware. AMON examines *all packets*, partitions traffic into sub-streams by using rapid hashing and computes certain real-time data products. The resulting data structures provide views of the intensity *and* connectivity structure of network traffic *at the time-scale of routing*. The proposed *integrated* framework includes modules for the *identification* of heavy-hitters as well as for *visualization* and statistical *detection at the time-of-onset* of high impact events such as DDoS. This allows operators to quickly visualize and diagnose attacks, and limit offline and time-consuming post-mortem analysis. We demonstrate our system in the context of real-world attack incidents, and validate it against state-of-the-art alternatives. AMON has been deployed and is currently processing 10Gbps+ live Internet traffic at Merit Network. It is extensible and allows the addition of further statistical and filtering modules for real-time forensics.

Index Terms

Network monitoring, detection, identification, visualization, PF_RING, gigabit streams, commodity hardware, data products, algorithms, statistics, heavy tails, extreme value distribution, network attacks.

I. Introduction

Motivation and background: The Internet has become a vital resource to business, governments and society, worldwide. It has thrived and grown under diverse conditions and technologies with

little to no centralized regulation. Its fundamental design principles have successfully ensured its robustness and broad accessibility. However, these same principles do not provide centralized management and/or monitoring of the *entire network*. Therefore, understanding broad based patterns such as traffic loads and thus adequacy of capacity and quality-of-service, composition of network traffic, adoption of new protocols and applications, are challenging tasks. Such traffic characterization problems and the corresponding network engineering, management and capacity planning solutions made necessary the analysis of large volumes of data, and also gave rise to statistical techniques to handle them, such as streaming algorithms [1], [2], sketches [3]–[5], tomography [6], [7] and analysis of heavy tails and long range dependence [8], [9].

In addition, it enables numerous vulnerabilities and security threats, both to the infrastructure and to its user base. For example, malicious activities such as *distributed denial of service* (DDoS) attacks are relatively easy to implement and rather hard to prevent, since best practices like origin IP anti-spoofing (e.g., BCP38 recommendation [10]) are not universally deployed by network operators. Their timely detection at appropriate short time-scales (e.g. in seconds) requires processing vast amounts of meta-data (e.g., NetFlow) distributed throughout the entire network, thus making it a challenging task. Further, the non-centrally controlled diverse hardware and software network infrastructure, allows many additional vulnerabilities open to exploitation by adversaries. A recent example features the exploitation of misconfigured NTP (network time protocol) servers, that led to one of the largest DDoS attacks ever recorded [11]. In such *reflection and amplification* attacks [12], multiple small requests are sent to several mis-configured NTP servers (or other UDP-based services), which inflict transmissions of large data amounts to targeted hosts. Thus, the intended victims get overwhelmed with traffic and temporarily disabled. Volumetric DDoS attacks are just one possible scenario; low-volume DDoS activities that rely on traffic sparseness to avoid detection are also of concern. It is important to be able to defend against the *DDoS threat model* and detect the onset of such potentially unpredictable attacks in order to adequately secure the network, e.g., by filtering (blocking) traffic or deploying security patches to network gear.

The key to addressing these diverse topics is the availability of *adequate data* coupled with *advanced monitoring and analysis tools* and the corresponding *software infrastructure*. The vast

volume of network traffic streams makes collection, storage and processing of all traffic data infeasible. Therefore, the focus has been on the information available in packet headers, such as source and destination addresses, application ports, protocol, payload size, etc. While such type of meta-data is more manageable, its rate of occurrence is still very fast. For example, storing packet header information (say initial 96 bytes of an Ethernet frame) from a 10 GE link at Merit[1] at a rate of 1.8 Mpps (million packets per second) requires 1.7 GB per 10 seconds (equivalently, around 15 TB per single day). The industry has developed tools such as NetFlow and others (sFlow, etc.), which effectively compress the packet meta data by grouping them into flows. NetFlow-alike traffic sampling functionality is available on many network elements. This compression mechanism, however, creates an intermediate step, which introduces a delay in the access to traffic meta-data (in addition to distorting its structure).

Even if one has access to raw data on packet headers or NetFlow, its high acquisition rate makes online analysis of this information often a formidable challenge. Many conventional statistical methods and algorithms are not sequential in nature and require access to large batches of data spanning several minutes to hours. Thus, possible DDoS attacks or changes in the network traffic patterns will be detected with offline analysis several minutes after their onset. The time-scales of such analyses are not desirable, if the goal is to prevent large-scale network outages. Note that specialized, albeit very expensive, appliances exist (e.g., Arbor Networks' PeakFlow), but in real-world settings are configured to receive heavily sampled Flow data (i.e., sampling rates of 1:1000 or more). Hence, low-volume or short-term attacks may elude detection. Further, such tools require *a priori* knowledge of baseline traffic patterns.

These challenges motivate us to develop new software and algorithmic infrastructure for harvesting and monitoring network traffic data *at the time-scale of routing* (i.e., at wire-speed), guided by the following principles: (a) examine *all* packets at the monitoring host; (b) develop memory efficient data structures and statistical summaries that can be computed and retained

[1]Merit Network, Inc. operates Michigan's research and education network. It is an Internet service provider that serves a population of nearly 1 million users. Merit is the largest IP network in Michigan, and its network includes a wide range of link types that include link speeds from T1 through 100Gbps. The network backbone consists of a 100G fiber ring, which passes through the major cities of Michigan, as well as Chicago.

at the time-scale of routing; (c) easy to build and deploy using *commodity, inexpensive, off-the-shelf* hardware; (d) the resulting data products should be available to be communicated and shared *in real-time* to centralized monitoring stations for further forensics, and (e) the monitoring architecture should allow for interactive filtering *in real-time*.

Related Work: Over the past 15 years, many practical tools have been developed for intrusion detection. For example, *Snort* (see, `snort.org`), *Suricata* (`suricata-ids.org`) and *Bro* (`bro.org`) are popular tools that rely on *signature-based* methods to examine traffic data for *known* malicious patterns. Nevertheless, recent malware often manage to evade pattern matching detection by becoming *polymorphic* (i.e., existing in various forms via encryption). The proposed work aims to complement existing tools by adopting instead a *behavioral-based* approach.

There exists a noteworthy amount of literature in the area of statistical, behavior-based anomaly detection. Standard techniques that seek 'change detection' points in traffic time series include exponential smoothing [13] or other more general time-series techniques [14], [15]. More recent methods employ wavelet-based tools [16] or subspace reduction methodologies based on principal component analysis [17], [18]. Such methods lack the capability of identifying the actual 'heavy-hitters' and, most importantly, suffer from the 'dimensionality curse' (i.e., having multi-dimensional features to monitor) and/or are inadequate for *online* realization on fast, multi-gigabit streams. Hence, there has been a lot of activity in the theoretical computer science community on designing and studying efficient algorithms for data streams that aim to alleviate the high dimensionality and high 'velocity' constraints. Many summary data structures (i.e., sketches) have been developed to address the challenging problems of *identification* of heavy-hitters or frequent items in a stream [19]–[25], anomaly *detection* in high-dimensional regimes [3], [24], [26], [27], compressed sensing and estimation of frequency moments [4], [28]–[33], community mining [34], etc. (see [35] and references therein).

In reality, the mere access to fast data streams involves formidable technical challenges. Many of the more sophisticated sketch-based algorithms (e.g., [4], [26], [28], [33]) tackle the multi-dimensional aspect of the problem, but implementing them on multi-gigabit streams is rather challenging or often impossible without the use of specialized hardware (e.g., FPGA). In addition, few frameworks (e.g., [24], [26]) take a holistic approach to develop methods

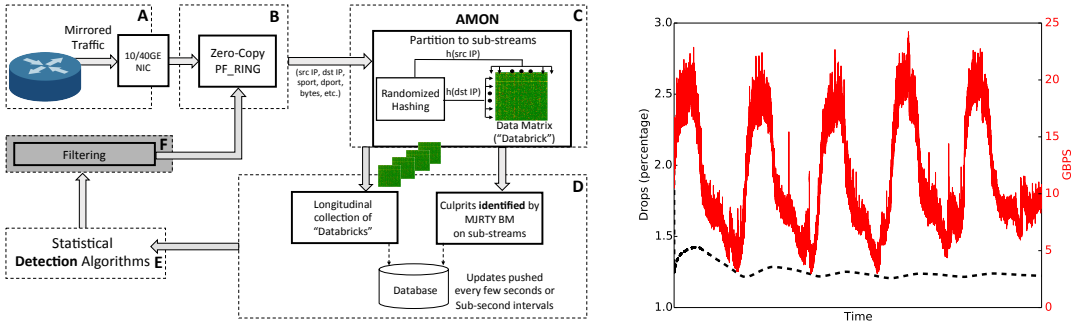
TABLE I: Characterization of previous work (representative sample).

	Identification	Detection	Real-time Visual.	High-Dim. Data	Real-time Streams	Interactive Zoom-in	Attack Classification
AMON	**yes**	**yes**	**yes**	**yes**	**yes**	**yes** (ongoing work)	**no**
CGT [19]	yes	no	no	yes	yes	no	no
Defeat [26]	yes	yes	no	yes	no	no	yes
[24]	yes	yes	no	yes	yes	no	no
[18]	no	yes	no	no	no	no	yes
[27]	no	yes	no	no	no[2]	no	no

that address the problems of change detection and identification together. Nevertheless, we acknowledge the presence of considerable previous work on the topics of network monitoring, troubleshooting and intrusion discovery. At the same time, our open-source platform offers a novel extendible framework that *couples together the important problems of detection, identification and visualization of aberrant behavior in multi-gigabit streams*. We propose *new* algorithms that are a direct consequence of the data products generated by our framework. Furthermore, there are relatively few tools that could allow network engineers to interactively examine 10Gbps+ traffic streams on the time scale of routing. This motivates the approach we have adopted in this paper, which focuses on leveraging several recent advances in high-speed packet capture to provide tools that are easy and inexpensive to deploy, examine every packet at the interface, and provide simple statistics of the 'signal' that allow network engineers to (a) visualize structural aspects of traffic, (b) detect changes in intensity or structure of traffic sub-flows, (c) potentially filter and zoom-in on anomalous IP address ranges identified automatically and (d) identify ranges of exact IP addresses associated with anomalous events. Table I highlights our contributions and identifies differences with previous literature.

Major Contributions: Our first contribution is the design and implementation of a software monitoring framework, referred to as *AMON (All-packet MONitor)*, working reliably over 10Gbps+ links. This framework is based on PF_RING [36] in zero-copy mode which efficiently delivers packets to the monitoring application by bypassing the OS kernel. We implement hash-based traffic summaries, which simply randomly assign source and destination pairs to bins providing an aggregate but essentially instantaneous picture of the traffic, which can be used to *diagnose*

[2]As noted in [27], it is limited to relatively low rates in an attempt to not overload the device and affect forwarding actions.



(a) High-level architecture of AMON.

(b) Software performance.

Fig. 1: The proposed framework. *Left:* AMON's data products comprise the input of identification, statistical detection and visualization modules introduced in this paper. *Right:* Performance at rates exceeding 20Gbps; minute drop rates recorded.

and visualize changes in structure and intensity.

Our second contribution is a suite of statistical tools for automatic *detection* of significant changes in the structure and intensity of traffic. The accompanying instances of Boyer-Moore majority vote algorithm [37] can be leveraged to *identify* precise IP addresses associated with attacks; this is an important contribution as well. We illustrate the new data acquisition framework with several views of the resulting data structures (hash-binned arrays), referred as *'databricks'*. We show (with real data from Merit!) that even basic visualization tools and algorithms *applied to the right type of data* can help instantaneously identify distributed attacks, which do not contribute to large traffic volume (see 'SSDP' and 'Tor' case studies, Section V-D).

This paper is organized as follows: Section II introduces our monitoring architecture, including the *data products* (Figure 2) and our software prototype; Section III introduces AMON's identification component; Section IV discusses our statistical methodology for automatic detection. Section V evaluates our software and algorithms on a rich set of real-world Internet data, including *four* real DDoS case studies, and compares against successful and robust state-of-the-art methods for detection and identification [19], [26].

II. Data and Software Infrastructure

An overview of the proposed architecture is portrayed in Figure 1a. The monitoring application is installed on a machine that receives raw packets in a streaming fashion. In our prototype at Merit Network, the monitoring probe receives traffic via a passive traffic mirror (using a SPAN—switched port analyzer—setting) configured on a network switch. Packets are then

efficiently delivered at 10Gbps+ rates to the monitoring module via PF_RING ZC. Subsequently, all packets are processed in a streaming fashion for constructing, via efficient hashing, a data matrix (i.e., the databrick depicted at Figure 2) *and* a separate matrix containing the most active source-destination flows identified via our extension of the Boyer-Moore algorithm (Section III). Periodically, these data products are shipped to a database for storage, further analysis and dashboard-based visualizations. These data are analyzed through various detection algorithms described in Section IV. Flows flagged by the detection module can be extracted for further analysis by the corresponding filtering mode that is currently under development.

A. *Data products via pseudo-random hash functions*

Internet traffic monitored at a network interface can be viewed as a stream of items $(\omega_n, v_n),\ n = 1, 2, \ldots$ (see e.g. [2]). The $\omega_n \in \Omega$ are the *keys* and v_n are the updates (e.g., payload) of the stream signal. For example, the set of keys could be all IPv4 addresses ($\Omega = \{0,1\}^{32}$); IPv6 addresses; pairs of source-destination IP addresses ($\Omega = \{0,1\}^{64}$); may include source and destination ports, etc; while payloads could be bytes, packets, distinct ports, etc. Since it is not feasible to store and manipulate the *entire signal* when monitoring 10Gbps+ links, we employ *hashing* to compress the *domain* of the incoming stream keys into a smaller set. Collisions are allowed and, in fact, expected, but the hash function is chosen so that it *spreads out* the set of observed keys approximately uniformly.

Consider for example the set of IPv4 addresses $\{0,1\}^{32}$ and let $h : \{0,1\}^{32} \to \{1, \ldots, m\}$ be a hash function (see [25], [38], [39]) that uniformly spreads the addresses over the interval $\{1, \ldots, m\}$. Upon observing key (s, d) of the source and destination of a packet, we compute the hashes $i := h(d)$ and $j := h(s)$ and update the *data matrix* $X = \{X(i,j)\}_{m \times m}$ as $X(i,j) := X(i,j) + v$. This matrix constitutes the first data output of our architecture and is depicted in Fig. 2. It is emitted at periodic intervals (e.g., 1 or 10 seconds) to a centralized database for online as well as further downstream analysis, and reinitialized. The row- and column-sums of this matrix yield the destination- and source-indexed hash-binned arrays, also depicted on the figure. These *data products* are used as inputs for the detection and visualization algorithms described below.

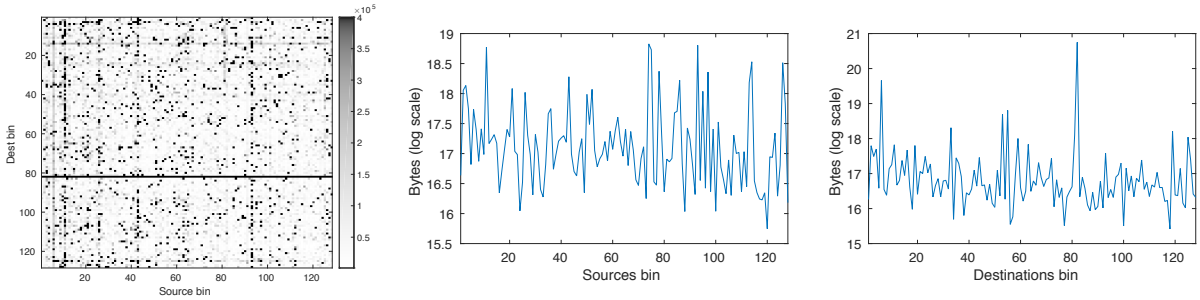


Fig. 2: Our data products. These data arrays, generated online by our PF_RING-based software, are used as the basic input structures for our detection algorithms. *Left:* The 'databrick' matrix during the 'Library' attack (see Section V); the apparent horizontal stripe (at dest 'bin' 82) signifies traffic from multiple sources to a single destination (victim). *Middle:* View of sources array, constructed by a matrix column-sum. *Right:* Destinations array; observe that 'bin' 82 stands out (notice the log scale).

B. Software implementation: PF_RING-based Monitoring

The AMON monitoring application is powered by PF_RING, a high performance packet capture network socket. Modern hardware advances in CPU speeds and architecture, memory bandwidth and I/O buses have shifted the bottlenecks in multi-gigabit packet reception into the software stack [40], [41]. PF_RING avoids unnecessary memory copies between the operating system layers, and hence the length of the packet journey between the network interface (NIC) and the monitoring application is shortened. Consequently, the number of CPU cycles spent for transferring packets from their NIC entry point to the application is significantly reduced. This leads to optimal memory bandwidth utilization [40], [41], and therefore to extremely efficient packet processing speeds (see Figure 1b).

Our system takes advantage of the zero-copy framework that PF_RING offers. In this mode, the monitoring application reads packets directly from the network interface, i.e., both the OS kernel and the PF_RING module are bypassed. As a result, efficient monitoring is now achievable with commodity, off-the-shelf hardware. For example, all experiments in this study were conducted using NIC cards costing below 800 USD. Although alternative fast packet processing schemes exist [41], PF_RING was selected due to its robustness, proved efficiency and broad versatility.

III. Identification: the hash-thinned Boyer–Moore algorithm

The proposed architecture periodically emits a list of heavy activity stream elements that can be used for traffic engineering purposes, accounting and security forensics. When an alert

is raised, operators can readily examine these 'heavy-hitters'. Our identification algorithm is based on the *Boyer-Moore* (BM) majority vote algorithm [37], and the idea of stream thinning for creating sub-streams described below. The so-named MJRTY Boyer-Moore algorithm [37] can identify *exactly* the majority element—the element whose volume is at least 50% of the total—in a stream, if one exists. It solves the problem in time *linear* in the length of the input sequence and *constant* memory. We first define the identification problem at hand.

Problem Addressed 1. (IDENTIFICATION OF HEAVY-HITTERS) Given an input stream (ω_n, v_n), identify the top-K most frequent items. The frequency of key ω is the sum of its updates v.

Next, we describe the original Boyer-Moore algorithm with an analogy to the *one-dimensional* random walk on the line of non-negative integers. A variable $count$ is initialized to 0 (i.e., the origin) and a candidate variable *cand* is reserved for use. Once a new key arrives, we check to see if *count* is 0. If it is, that IP is set to be the new candidate *cand* and we move *count* one step-up, i.e. $count = 1$. Otherwise, if the IP is the same as *cand*, then *cand* remains unchanged and *count* is incremented, and, if not, *count* moves one step-down (decremented). We then proceed to the next IP and repeat the procedure. Provably, when all IPs are read, *cand* will hold the one with majority, if majority exists.

Our extension of the MJRTY Boyer-Moore method applied to each sub-stream is outlined in Figure 3. It returns up to m 'heavy-hitter' items present in a stream of keys, taking values in $\{\omega_1, \ldots, \omega_N\}$, by 'thinning' the original stream $\mathcal{S}$ into m sub-streams. In the *update* operation, upon observing a new stream item (ω, v), we compute the sub-stream index $s := h_1(\omega)$ using hash function h_1. In essence, we run m independent realizations of the Boyer-Moore algorithm described above, one for each sub-stream. Arrays *count* and $cand$ hold the algorithm state for all sub-streams, i.e., $cand[s]$ holds the majority candidate for s. Array $count$ is updated accordingly with the value of v as lines 9, 11 and 16 depict. The auxiliary $flag$ for each sub-stream s can help track whether a majority is indeed underlying into that sub-stream; at the start of the monitoring period the $flag$ corresponding to s is set, and as long as $count[s]$ remains non-negative (i.e., $cand[s]$ needs no updates), the $flag$ never resets. A $flag$ that remains 'on'

Input: Number heavy-hitters: K, $K \leq m$
Input: hash function $h_1 : [N] \rightarrow [m]$
Input: hash function $h_2 : [N] \rightarrow [m']$, $m' = 256$

$count[i] = 0, \ \ i \in [m]$
$cand[i] = -1, \ \ i \in [m]$
$flag[i] = 1, \ \ i \in [m]$
$P_{bm}[i, j] = 0, \ \ i \in [m], j \in [m']$.

(a) Initialization

$P_{est}[s] = \max_{j \in [m']} P_{bm}[s, j], \ \ s \in [m]$
Initialize $\mathcal{O} = \emptyset$
for r=1 **to** K **do**
  Find $o = \text{Argmax}_{j \in [m] \setminus \mathcal{O}} P_{est}[j]$
  $\mathcal{O} = \mathcal{O} \cup o$ #Exclude for next iteration
  Output ω_r^* =cand[o]
  Output $P_{est}[o]$
end for

(b) Query operation

[Thin] Compute buckets $s = h_1(\omega)$ and $j = h_2(\omega)$.
if $cand[s] == -1$ **then**
  $cand$[s] = ω, $count$[s] = v, P_{bm}[s,j] = v
else
  if $cand[s] == \omega$ **then**
    P_{bm}[s,j] = P_{bm}[s,j] + v, $count$[s] = $count$[s] + v
  else
    if $count[s] > 0$ **then**
      $count[s]$ = $count[s]$ - v
      if $count[s] < 0$ **then**
        $cand$[s] = ω, $count$[s] = -$count$[s] # reset cand
        $flag$[s] = 0 # reset flag
      end if
      P_{bm}[s,j] = P_{bm}[s,j] + v
    else
      $cand$[s] = ω, $count$[s] = v # reset cand
      $flag$[s] = 0 # reset flag
      P_{bm}[s,j] = P_{bm}[s,j] + v
    end if
  end if
end if

(c) Update operation for stream item (ω, v)

Fig. 3: Identification algorithm: Hash-thinned MJRTY Boyer-Moore.

guarantees the presence of a majority item[3]. An estimate of the frequency of each hitter can be obtained via the $m \times m'$ data structure P_{bm}. Through the use of an independent hash function h_2, this sketch structure keeps a hash array of size m' for each sub-stream s, and gets updated with the arrival of each stream element.

When a *query* operation is performed (see Figure 3), we retrieve the m candidates. An estimation of the volume of a candidate 'hot' item s is recovered by looking at the maximum value of sub-array $P_{bm}[s]$. The m candidates are ranked according to these estimates, and an approximation of the set of top-K hitters is identified.

IV. Statistical Methods for Anomaly Detection

This section introduces three new detection methods. We start with a data exploration and model validation discussion that characterizes our data products; all methods leverage AMON's data. The first method is based on estimating the *number of 'heavy hitters'* at each time point;

[3]As an example, during the real-time 5-day experiment with the Chicago traffic (Figure 1b), an average fraction of at least 85.41% of all sub-streams (with 0.11% standard deviation) contained a majority element (for bytes).

this estimate represents a *monitoring statistic* that one can track, and time points with heavy hitter activity can be flagged as anomalies. Next, a method derived by modeling the distribution of the *relative volume* of the heaviest bins in the source and destination, 1-dimensional, hash-binned arrays is introduced. This section concludes with a technique for discovering *structural changes* in traffic, a method chiefly suitable for seemingly innocuous, low volume aberrant behavior. Henceforth, the problem of anomaly detection is formulated as follows.

Problem Addressed 2. (DETECTION) Given an input traffic stream, find the time points when the baseline probability distribution of an appropriate monitoring statistic seems inadequate.

A. Statistics, model validation and data exploration

The successful detection of statistically significant anomalies in the derived hash-binned traffic arrays depends on the adequacy of the model employed. We undertook an extensive empirical analysis of long hash-binned arrays and found that heavy-tails are ubiquitous. Figure 4 (left panel, top figure) shows a time series of the linearized hash-binned array of outgoing (Source) traffic at Merit Network for the period 17:30-18:30 EST on July 22, 2015. Observe the consistent presence of extreme peaks in the data, some of which may in fact be due to an attack event (see the 'Library' case study, Section V). By zooming-in on a short (seemingly calm) 3-minute period—bottom right—we observe that the extreme peaks, although of lower magnitude, persist.

Heavy-tailed power law distributions are suitable statistical models for data exhibiting such characteristics. Power laws are ubiquitous in computer network traffic measurements. It is well known and documented that file-sizes, web-pages, Ethernet traffic, etc. exhibit power-law tails [42], [43]. Specifically, let $X = X_t(i)$ denote, for example, the amount of traffic registered in a given hash-array bin i. Then, a parsimonious model for its tail is as follows:

$$\mathbb{P}(X > x) \sim c/x^{\alpha}, \quad \text{as } x \to \infty, \tag{1}$$

where '$\sim$' means that the ratio of the left- to the right-hand side converges to 1 and where $\alpha > 0$ and $c > 0$ are constants. The smaller the exponent α, the heavier the tail of the distribution, and the greater the frequency of extreme values. In particular, if $\alpha < 2$, then the variance of X does not exist and if $\alpha < 1$, then the mean of this model is infinite.

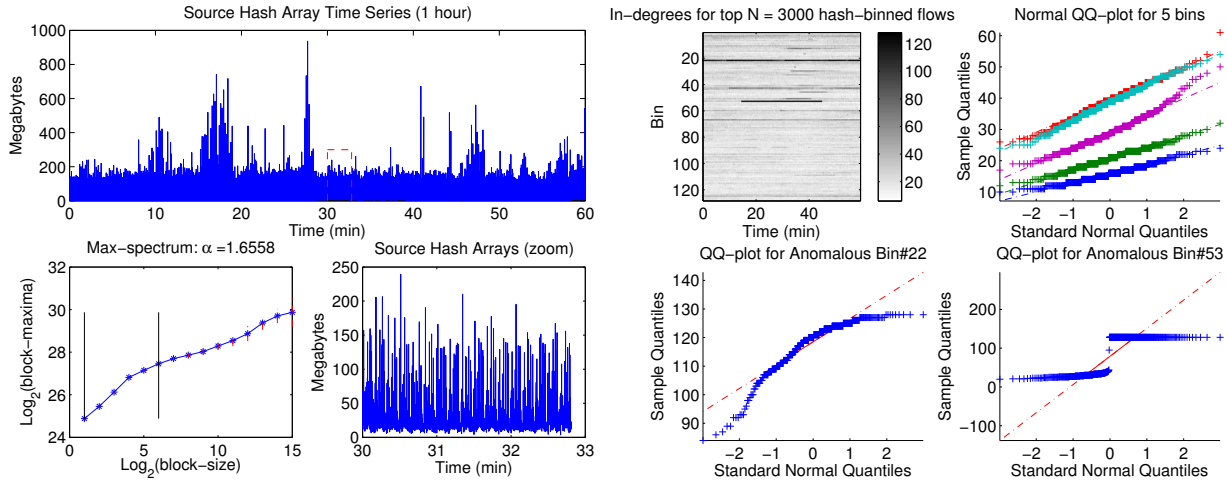


Fig. 4: *Left panel:* Time-series of Source hash-binned arrays (Top) and its zoomed-in version (Bottom-right), computed over 10-second windows. The *max-spectrum* of the entire time series is plotted on the bottom-left. Merit Network: 17:30-18:30 EST, July 22, 2015. *Right panel:* Merit Network 16:00-17:00 EST, Aug 1, 2015 – the 'Tor' event in Section V-D. (Top-left) Ingress connectivity for the top $N = 3000$ hash-binned flows per 10-second windows over 1-hour. (Top-right) QQ-plots demonstrating accuracy of the Normal approximation of typical in-degree distributions. (Bottom plots) QQ-plots for anomalous bins.

Figure 4 (left panel, bottom) shows the *max-spectrum* of the entire 1-hour long time series of hash-binned source traffic array. The *max-spectrum* is a plot of the mean log-block-maxima versus the log-block-sizes of the data. A *linear* trend indicates the presence of power-law tails as in (1), while the slope provides a consistent estimate of $1/\alpha$. Thus, steeper max-spectra correspond to lower values of α and heavier tailed distributions generating more extreme values. A useful feature of the max-spectrum plot is its ability to examine various log-block-sizes (scales), thus enabling simultaneous examination of the power-law behavior in the data at various time-scales [44]. As it can be seen, the power-law behavior (linearity in the spectrum) extends over a wide range of time-scales from seconds to hours. The time-scale relevant to our studies is a few seconds, which yields estimates of $\alpha \approx 1.6$, obtained by fitting a line over the range of scales ($\log_2$-block-sizes) 1 through 6. Over intermediate time-scales (a few minutes) the exponent α raises to about 2.5. The simple power-law models are no longer sufficient to capture the distribution over the largest time-scales (hours), where complex intermittent non-stationarity and diurnal trends dominate.

Alternatively, Figure 5 shows the complimentary cumulative distribution function $x \mapsto \mathbb{P}(X >$

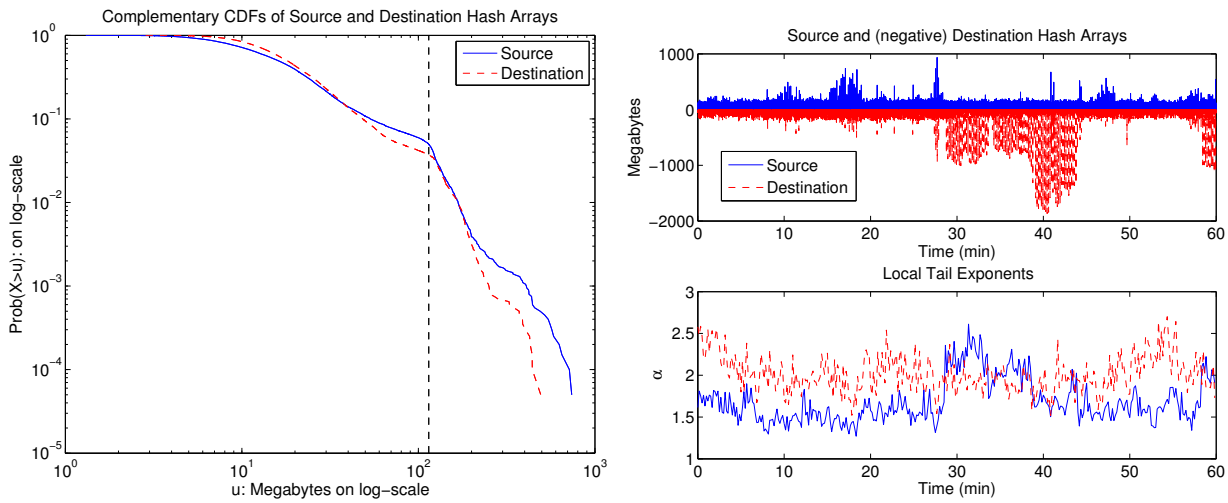


Fig. 5: *Left:* Complementary CDF $u \mapsto \mathbb{P}(X > u)$ on log-scale for Source and Destination traffic hash-arrays. *Right:* Time series and tail exponents of Source and Destination traffic hash-array computed over 10-second windows. Merit Network: July 22, 2015, 17:30-18:30 EST.

x) on a log-scale for both Source and Destination traffic arrays. Linear scaling on this plot corresponds to power-law behavior as in (1) and the slope of the linear fit yields an estimate of $-\alpha$. Even though one cannot clearly talk about time-scales here, similarly to the max-spectrum plot, one sees two regimes of power-law scaling. The heavy-tail behavior is relatively more severe for the range of smaller values corresponding, on average, to shorter time-scales. Our focus is on very short time scales of a few seconds to a minute. Our analysis shows that over such time-scales, the power-law model captures the essence of the distribution. Figure 5 shows tail exponents for both Source and Destination hash arrays $X_t(i),\ i = 1, \ldots, m$ as a function of time t. Observe the persistent heavy-tailed nature of the data throughout the entire period of time. Note that the Source (outgoing) traffic is slightly heavier-tailed (lower exponents α) than the Destination (incoming). Note also that the estimators of the tail exponent are rather robust to large-volume fluctuations, e.g. in the Destination time-series. This is another important feature of the max-spectrum, which will play a role in the successful detection of such anomalous event, described in the following sections.

B. Detection of Heavy Hitters

In this section, we describe a methodology (named as 'Fréchet method') for aberrant behavior discovery, based on monitoring the number of hash-bins involving heavy traffic, henceforth referred as *heavy hitters*. The precise definition of a heavy hitter is rather subtle; depending on the traffic context, a given flow (e.g. video transmission) may be perceived as a heavy hitter in light traffic conditions, while it may be in fact a 'typical' event in normal traffic conditions. Here, we adopt a statistical perspective, where we flag hash-bins as heavy hitters, if their signal exceeds a given quantile of a *baseline probability distribution*, i.e., we view heavy hitters as 'outliers'. In order to be adaptive to changing traffic conditions, we shall dynamically and robustly estimate the baseline probability model from the data. Our notion of heavy hitters depends on the probability associated with the quantile threshold. This tuning parameter may be set depending on how sensitive we would like to be to 'alarms'.

Let $X_t = \{X_t(i)\}_{i=1}^m,\ t = 1, 2, \cdots$ be sequence of hash-binned traffic arrays. In the case of the *source IP* signal, for example, $X_t(i)$ corresponds to the column-sum of the databrick matrix and represents the number of bytes originating from all source IPs ω hashed to bin i, i.e. $h(\omega) = i$ over the time-window t. Figure 2 (middle) shows an example of such array. The process of hashing effectively randomizes traffic flows in different bins, and therefore, the entries $X_t(i),\ i = 1, \ldots, m$ may be reasonably assumed to be statistically independent and identically distributed (i.i.d.).

Our goal is to identify and flag the presence of abnormally large (heavy) traffic. One way that this manifests itself is through abnormally large values of $X_t(i)$s, for *some* i's. To this end, we consider the sample maximum of the hash-array:

$$D_m(X_t) := \max_{i=1,\ldots,m} X_t(i). \tag{2}$$

We shall identify a bin $i \in \{1, \ldots, m\}$ as a heavy hitter, if its value is large, relative to an *asymptotic approximation* to the distribution of the sample maximum.

Proposition 1. *Let $X(i),\ i = 1, \ldots, m$ be i.i.d. random variables with heavy tails as in* (1). *Then, as $m \to \infty$, we have that*

$$\frac{1}{m^{1/\alpha}} D_m(X) \equiv \frac{1}{m^{1/\alpha}} \max_{i=1,\ldots,m} X(i) \xrightarrow{d} c^{1/\alpha} Z_\alpha, \tag{3}$$

where $\mathbb{P}(Z_\alpha \leq x) = e^{-1/x^\alpha}$ *has the standard* α*-Fréchet distribution and* c *is the asymptotic parameter in* (1).

This result is a simple consequence of Theorem 3.3.7, p. 131 in [45]. For completeness, its proof is given in the Appendix.

Relation (3) suggests that for relatively large values of m, we can use the limit α-Fréchet distribution to calibrate the detection of heavy hitters. Specifically, given a sensitivity level p_0 (e.g. equal to 0.95 or 0.99), we flag the bin i as a *heavy hitter*, if

$$X_t(i) \geq T_{p_0}(m, \alpha, c) := m^{1/\alpha} c^{1/\alpha} \Phi_\alpha^{-1}(p_0) = \Big(\frac{c}{\log(1/p_0))}\Big)^{1/\alpha}, \tag{4}$$

where $\Phi_\alpha^{-1}(p) = (\log(1/p))^{-1/\alpha},\ p \in (0,1)$ is the inverse of the standard α-Fréchet cumulative distribution function $\Phi_\alpha(x) = e^{-1/x^\alpha},\ x > 0$. This way, in practice, under normal traffic conditions, the probability of flagging any bin in time-slot t as a heavy hitter is no greater than $(1 - p_0)$. The rate of potential false alarms, may be controlled and reduced by judiciously increasing the level p_0. On the other hand, the presence of abnormally large bins relative to the reference distribution will be flagged if their values exceed the threshold $T_{p_0}(m, \alpha, c)$.

To be able to use this methodology, one should estimate the key parameters α and c appearing in formula (4). The recently proposed max-spectrum method in [44] is particularly well-suited to this task. It is easy to tune, robust to outliers, computationally efficient, and it provides estimates of both the scale parameter c and the tail exponent α. This methodology is summarized in the formal algorithm (Algorithm 1).

Remark 1. The hash-array is obtained from the PF_RING-based methodology at the time scale of one array per several seconds. For the traffic conditions in the Merit Network (e.g. rates of 10Gbps), we found that time-windows of 10 seconds provide sufficiently well-populated bins that lend themselves to reasonable heavy-hitter detection. In this setting, we output estimates of heavy hitters every 10 seconds. For greater traffic rates, hash-binned arrays are populated faster and our methodology can be applied at an even shorter, sub-second time-scale.

Remark 2. Proposition 1 is an asymptotic result. In our experiments with real traffic data, we found the approximation based on the Fréchet distribution to be reasonably accurate for m as low as 128 and durations about 10 seconds.

Algorithm 1: Fréchet method

Input: Stream of hash-arrays $X_t = \{X_t(i)\}_{i=1}^m$; probability level $p_0 \in (0,1)$; smoothing coefficient $\lambda \in (0,1)$.

Output: Stream of significant heavy-hitter bins $\mathcal{H}_t \subset \{1,\dots,m\}$ and their counts $k_t = |\mathcal{H}_t|$.

for each stream item X_t **do**
  Estimate the tail exponent $\hat{\alpha} := \alpha(X_t)$ and scale coefficient $\hat{c} := c(X_t)$ from the sample $X_t = \{X_t(i)\}_{i=1}^m$ based on the *max-spectrum*.
  if $(t = 1)$ **then**
    Set $\alpha_t := \hat{\alpha}$ and $c_t := \hat{c}$
  else
    Perform EWMA smoothing: $\alpha_t := \lambda\hat{\alpha} + (1-\lambda)\alpha_{t-1}$ and $c_t := \lambda\hat{c} + (1-\lambda)c_{t-1}$.
  end if
  Compute the significance threshold $T_t := T_{p_0}(m, \alpha_t, c_t)$ using (4).
  Estimate the set of heavy hitter bins $\mathcal{H}_t$ at window t as $\mathcal{H}_t := \Big\{ i \in \{1,\dots,m\} \,:\, X_t(i) \geq T_t \Big\}$.
  return $\mathcal{H}_t$ and $k_t := |\mathcal{H}_t|$.
end for

Algorithm 2: Relative volume

Input: Stream of hash-arrays $X_t = \{X_t(i)\}_{i=1}^m$; probability level $p_0 \in (0,1)$; candidate value $k \in \{1,\dots,m\}$ (preferably $\ll m$); smoothing parameter $\lambda \in (0,1)$.

Output: Binary stream of alarm-flags $f_t \in \{0,1\}$.

for each stream item X_t **do**
  Estimate the tail exponent $\hat{\alpha} := \alpha(X_t)$ from the sample $X_t = \{X_t(i)\}_{i=1}^m$.
  if $(t = 1)$ **then**
    Set $\alpha_t := \hat{\alpha}$
  else
    Perform EWMA smoothing: $\alpha_t := \lambda\hat{\alpha} + (1-\lambda)\alpha_{t-1}$.
  end if
  Compute the relative volume of of the top–k bins $V_t(k)$ as in (5).
  Using Monte Carlo simulations, compute numerically the significance threshold $q_t = q_t(p_0; k, \alpha_t, m)$, such that
$$\mathbb{P}(W_{\alpha_t}(k,m) \leq q_t) \approx p_0.$$
  return $f_t := \mathbb{I}\{V_t(k) > q_t\}$, i.e., flag $V_t(k)$ as significantly large (at level p_0) if $V_t(k) > q_t$.
end for

C. Detection via Relative Volume

Alternatively, one can detect high-impact events by monitoring the volume of the top-hitters relative to the total traffic. As before, suppose that $X_t = \{X_t(i)\}_{i=1}^m$ is a hash-binned array of traffic volume (in bytes) computed over a given time window.

Sort the bins in decreasing order, so that $X_t(i_1) \geq \cdots \geq X_t(i_k) \geq \cdots \geq X_t(i_m) \geq 0$. Fix a $k \in \{1,\dots,m\}$ and consider the relative volume of traffic contributed by the top-k bins:

$$V_t(k) := \frac{\sum_{j=1}^k X_t(i_j)}{\sum_{j=1}^m X_t(i_j)}. \tag{5}$$

Note that the indices of top-k bins can change from one time-window to the next.

We aim to identify when $V_t(k)$ is 'significantly' large. For example, if $k = 1$, one would like to know if the top bin suddenly carries a very large proportion of the traffic relative to the rest.

This could indicate an anomaly in the network. As in the previous section, we will measure significance relative to a baseline probability model, which is dynamically estimated from the data. The ubiquitous heavy-tailed nature of the $X_t(i)$'s will play a key role.

Let now $X(i),\ i = 1, \ldots, m$ be i.i.d. non-negative random variables representing a generic hash-binned traffic array. As argued in the previous section, in a wide range of regimes, the distribution of the $X(i)$'s is heavy tailed, and they may be assumed independent because of the pseudo-randomization due to hashing. Thus, as in (1), we shall assume that $\overline{F}(x) \equiv 1 - F(x) = \mathbb{P}(X(1) > x) \sim c/x^{\alpha}$, for some $c > 0$ and $\alpha > 0$. It is well-known that if the distribution function F is continuous, then $U(i) := \overline{F}(X(i)),\ i = 1, \ldots, m$ are i.i.d. Uniform$(0, 1)$. Therefore, the Rényi representation for the joint distribution of the *order statistics* (p. 189 in [45]), implies

$$\Big(X(i_k)\Big)_{k=1}^{m} \stackrel{d}{=} \Big(\overline{F}^{-1}\Big(\frac{\Gamma_k}{\Gamma_{m+1}}\Big)\Big)_{k=1}^{m}. \tag{6}$$

This yields the following result about the distribution of the relative volume.

Proposition 2. **(i)** *Under the above assumptions, we have*

$$\{V(k;m),\ k = 1, \ldots, m\} \stackrel{d}{=} \Big\{\frac{\sum_{j=1}^{k} \overline{F}^{-1}(\Gamma_j/\Gamma_{m+1})}{\sum_{j=1}^{m} \overline{F}^{-1}(\Gamma_j/\Gamma_{m+1})},\ k = 1, \ldots, m\Big\}. \tag{7}$$

(ii) *Under* (1)*, for fixed* $1 \le k < \ell$*, we have, as* $m \to \infty$,

$$\frac{V(k;m)}{V(\ell;m)} \xrightarrow{d} W_{\alpha}(k,\ell) := \frac{\sum_{j=1}^{k} \Gamma_j^{-1/\alpha}}{\sum_{j=1}^{\ell} \Gamma_j^{-1/\alpha}}. \tag{8}$$

The proof is given in the Appendix. Recall that our goal is to test whether $V(k;m)$ is significantly large. The asymptotic result in (8) suggests that the distribution of the statistic $W_{\alpha}(k,\ell)$ can be used as a baseline model. Note however that it quantifies the magnitude of $V(k,m)$ relative to $V(\ell,m)$ for some fixed ℓ. In practice, in the context of network traffic hash-binned arrays we studied, it turns out that $V(\ell,m) \approx 1$ for moderately large values of ℓ. Therefore, the denominator in the left-hand side of (8) can be taken as 1. Further, to be slightly conservative, one can take $\ell = m$. We therefore obtain the distributional approximation $V(k;m) \stackrel{d}{\approx} W_{\alpha}(k,m) := \frac{\sum_{j=1}^{k} \Gamma_j^{-1/\alpha}}{\sum_{j=1}^{m} \Gamma_j^{-1/\alpha}}$. Note that this approximation is in fact valid exactly, if $\overline{F}(x) = c/x^{\alpha},\ x \ge c^{1/\alpha}$, i.e. under the Pareto model. This discussion leads to Algorithm 2.

Remark 3. In scenarios where the Pareto approximation is not as accurate, one can adapt the above algorithm by considering $\ell < m$ and test the contribution of ratios of volumes $V(k;m)/V(\ell;m)$, relative to the baseline distribution of $W_\alpha(k,\ell)$. As indicated above, for simplicity, and to be slightly conservative in practice, we use $\ell = m$, which worked rather well.

The significance threshold q_t in Algorithm 2 may fluctuate substantially in time, since the tail exponent α_t does (see, e.g. Figure 5 in [46]). This natural *adaptivity* property allows us to dynamically calibrate to the changing statistical properties of the stream. It is important, however, to be also *robust* to sudden changes of regime due to the onset of anomalies, i.e., we should not adapt to the anomalies we are trying to detect. Such robustness can be achieved and tuned by the smoothing parameter λ. The smaller the value of λ, the closer the α_t to past values α_s, $s \leq t$. Some degree of smoothing can also improve estimation accuracy through *borrowing strength* from the past. In practice, we found that $\lambda \approx 0.5$ works well in our conditions.

If the type of anomalies considered *persist* over several windows of time Δ, one can substantially decrease the false alarm rate by considering *control charts*. This leads to a slight modification of Algorithm 2. Following [47], one can consider the p-values, $p_t := \mathbb{P}(V_t(k) > W_{\alpha_t}(k,m))$, and then apply an EWMA on the *z-scores*: $z_t := \lambda_p \Phi^{-1}(1-p_t) + (1-\lambda_p) z_{t-1}$, for another weight $\lambda_p \in (0,1)$. Then, under baseline conditions, z_t follows the Normal distribution with zero mean and variance $\sigma_z^2 = \lambda_p/(2-\lambda_p)$. Thus, classical process control methodology suggests to raise an alarm if $z_t/\sigma_z > L$, for a given level parameter $L > 0$ [13]. The pair of parameters (λ_p, L) can be tuned so as to ensure detection of persisting anomalies, while minimizing false alarms. Section V includes studious sensitivity analyses of these calibrations controls.

D. Community Detection

Consider now the two-dimensional matrix $X_t = \{X_t(i,j)\}_{i,j=1}^m$ of updates, obtained for a certain time t. The technique proposed next aims at detecting changes in the *community structure* of the network flows. To this end, focus on the top N bins of $X_t(i,j),\ i,j = 1,\ldots,m$, which represent an aggregate summary of the top origin-destination flows in the network.

Let $A_t = (a_t(i,j))_{m\times m}$ be a binary matrix, such that $a_t(i,j) = 1$ if and only if bin (i,j) belongs to the set of top N items in the array X_t. One can view A_t as an adjacency matrix of an oriented graph G_t, which is a type of a histogram of the underlying (rather sparse) graph

of flows from a given sIP to a dIP that are active over the time-window of interest. Changes in the connectivity of G_t indicate changes in the community structure of the traffic flows. For example, in the event of a DDoS or other distributed attacks, a given destination IP ω_0 is flooded with substantial amount of traffic from multiple source IPs. If a large number of sources are involved, then this will likely result in a horizontal strip of relatively large values in the two-dimensional hash-binned array. The location of the strip will be $i_0 := h(\omega_0)$—the index of the bin where the target destination IP ω_0 is hashed (see, e.g. Figures 7 and 8 for visualizing the 'Tor', 'SSH-scanning' and 'SSDP' attack events).

One way to formally and automatically detect such features is to focus on the graph with adjacency matrix A_t. In this event, the matrix A_t will have a relatively larger number of 1s in the i_0th row and, correspondingly, the in-degree of node i_0 will be large. We propose a statistical method for quickly identifying *significant* peaks in the in-degrees (or out-degrees). This method, combined with the information from the Boyer-Moore MJRTY instances associated with the bins involved can lead to an almost instantaneous identification of possible targets as well as (potential) culprits of malicious activities.

Focus on ingress connectivity, i.e., let $I_t(i) := \sum_{j=1}^{m} a_t(i,j), \quad i = 1, \ldots, m$ be the in-degree associated with node i for the oriented graph G_t. Our goal is to flag statistically significant peaks of $I_t(i)$. As argued in Sections IV-B and IV-C, hashing ensures randomization and hence $I_t(i),\ i = 1, \ldots, m$ can be reasonably assumed to be independent. In contrast with the previous sections, however, the distribution of the counts $I_t(i)$ are no longer heavy-tailed but rather well-approximated by a Normal distribution. For a fixed i, thanks to the randomization induced by hashing, one can view $a_t(i,j)$'s as independent in j. Hence, for relatively large m, as well as N and ultimately traffic rate, the CLT ensures that centered and normalized integer counts $I_t(i)$ can be modeled by the Normal distribution. Indeed, Figure 4 (right panel, top-right plot) shows Normal quantile-quantile plots of $I_t(i),\ t = 1, \ldots, T$ for 5 typical (non-anomalous) bins i. The linearity in the plots indicates agreement with the Normal distribution. The heatmap therein (top-left) shows the entire array $(I_t(i))_{m \times T}$ of in-degrees computed over 10-second time windows over the duration of 1 hour. We focused on the top $N = 3000$ flows. The bottom plots in this figure show the QQ-plots corresponding to anomalous bins with high in-degree

corresponding to the higher intensity lines in the top-left plot.

Given the above discussion, in the baseline regime, we shall assume that $I_t(i),\ i = 1, \ldots, m$ are independent $\mathcal{N}(\mu_t, \sigma_t^2)$. Then for $D_t := \max_{i=1,\ldots,m} I_t(i)$, by the independence of the $I_t(i)$'s, we obtain $\mathbb{P}(D_t \leq x) = \Phi\left(\frac{x-\mu_t}{\sigma_t}\right)^m$, where Φ is the standard normal CDF. Fix a probability level p_0 (e.g. 0.99), and consider the *significance threshold* $u_t(p_0) \equiv u(p_0, m, \mu_t, \sigma_t) := \mu_t + \sigma_t \times p_0^{1/m}$. Thus, in the baseline regime, all in-degrees $I_t(i),\ i = 1, \ldots, m$ lie below $u_t(p_0)$ with probability p_0. As in Section IV-B, we shall flag all bins i, for which $I_t(i)$ exceeds $u_t(p_0)$ as *anomalous*. The detection algorithm is analogous to Algorithm 1, except that now one should estimate the parameters μ_t and σ_t. This can be similarly done using an EWMA of the empirical means and standard deviations of the samples $I_t(i),\ i = 1, \ldots, m$. We omit the details to avoid repetition.

This method is illustrated in Section V, where it is successfully employed in mining seemingly harmless events characterized by high node-connectivity (e.g., the 'SSDP' and 'Tor' cases). These events are harder to detect via the methods described in Sections IV-B and IV-C.

Remark 4. Observe that the access to various cloud services and resources can have similar characteristics, where multiple source IPs communicate with a single destination IP (server). Such servers, however, are typically well-known and can be *a priori* filtered out. An alternative application of this methodology is to track the up-surge of users to a particular service, such as Twitter, Facebook or Google, for example. Such up-surges, not necessarily due to malicious activity, may be of interest to network engineers or researchers.

V. Performance Evaluation

A. Software performance

The excellent performance of PF_RING is well documented [40], [41]; this section focuses instead on our monitoring application. We perform measurements in situations where AMON is deployed in the field, and under heavy stress-testing with a traffic generator appliance.

Figure 1b (right) illustrates our software capabilities when monitoring traffic at Merit's main peering point in Chicago. Our setting involves a passive monitor (i.e., packet tap) receiving traffic from four SPAN 10GE ports. The mirrored traffic includes both ingress and egress network traffic, and a 5-day snapshot of aggregate volume is shown in Figure 1b. Note that traffic rates are well above 20Gbps; however, AMON monitoring traffic *from all four 10GE ports*

simultaneously, experienced minimal packet drops (below 1.5%). Further, the amount of physical memory required by our application was only around 40MB, something expected from the low space complexity of the Boyer-Moore algorithm [39].

To shed more light into this, we undertook performance tests using a traffic generator with 40 byte payload packets (i.e., sending at the minimum frame of 64 bytes). At wire-speed of 10Gbps we measured throughput that exceeded 12 Mpps (million packets per second). This corresponds to a drop rate of 18%; testing with payloads of size 96 and above showed zero loss at wire speeds. We conjecture that the bottleneck seems to be the buffer size of the NIC card we used, and not PF_RING. In particular, the Intel card we tested has buffers of size 4096, and hence packet drops seem to be inevitable at these rates.We are in the process of conducting tests on cards with larger buffers in order to verify our hypothesis.

B. Identification accuracy

Next, we demonstrate the identification accuracy of MJRTY Boyer-Moore; we perform comparisons against Combinatorial Group Testing (CGT) [19]. We utilize an hour-long NetFlow dataset, collected at Merit, with 92 million flows and an aggregate volume of 447 GBytes and around 580 million packets. Both methods are evaluated against the ground truth (i.e., exact recovery of top-K hitters). All methods report their answers every 100,000 NetFlow records; Table II illustrates the average proportion of identified heavy hitters among the top-K and the standard error (in parenthesis). The chosen data contain a low-volume DDoS attack attributed to the Simple Service Discovery Protocol (SSDP); see Figure 8 (left).

The CGT method [19] is a probabilistic technique, based on the ideas of 'group testing'. It aims at finding the elements whose volume is at least $1/(k+1)$ of the total; this is a relaxed version of the top-K hitters problem. The authors provide performance guarantees with respect to accuracy, space and time. It is suited for high-speed streaming data; indeed, besides its offline evaluation on accuracy, *we have implemented the method in the AMON framework and verified its time and space efficiency.* Its online realization demonstrated results similar to Figure 1b. For the results of Table II we sought the top source IPs per interval. The tuning parameters for CGT include the hash-table size W, and the number of groups T (in all experiments, $T = 2$; increasing T improves accuracy but worsens the efficiency on real-data). The granularity unit b

TABLE II: Identification; comparison with Combinatorial Group Testing (CGT) [19].

Top-K hitters	BM (m=512)	BM (m=1024)	CGT (k=500,W=1024)	CGT (k=1000,W=2048)	CGT (k=2000,W=4096)
K=10 (packets)	$0.99_{(0.04)}$	$0.99_{(0.03)}$	$0.91_{(0.10)}$	$0.91_{(0.10)}$	$0.89_{(0.09)}$
K=10 (bytes)	$0.99_{(0.03)}$	$0.99_{(0.03)}$	$0.79_{(0.14)}$	$0.85_{(0.12)}$	$0.86_{(0.11)}$
K=50 (packets)	$0.94_{(0.04)}$	$0.98_{(0.02)}$	$0.90_{(0.07)}$	$0.90_{(0.05)}$	$0.90_{(0.04)}$
K=50 (bytes)	$0.96_{(0.03)}$	$0.98_{(0.02)}$	$0.80_{(0.11)}$	$0.89_{(0.06)}$	$0.90_{(0.05)}$
K=100 (packets)	$0.85_{(0.05)}$	$0.94_{(0.03)}$	$0.60_{(0.11)}$	$0.89_{(0.05)}$	$0.90_{(0.03)}$
K=100 (bytes)	$0.90_{(0.03)}$	$0.96_{(0.02)}$	$0.58_{(0.09)}$	$0.86_{(0.09)}$	$0.92_{(0.04)}$
K=200 (packets)	$0.71_{(0.05)}$	$0.87_{(0.03)}$	$0.30_{(0.06)}$	$0.76_{(0.12)}$	$0.91_{(0.03)}$
K=200 (bytes)	$0.77_{(0.04)}$	$0.90_{(0.02)}$	$0.29_{(0.05)}$	$0.48_{(0.08)}$	$0.78_{(0.13)}$

(see [19], Sec. 3.3) is set to $b = 8$ for better efficiency, at the expense of space in memory.

The MJRTY BM is regulated with the number of sub-streams, m. Note that, regarding space utilization, hashing with size $m = 1024$ corresponds to the CGT case $W = 1024$. In all cases, we used $W \geq 2k$, per Lemma 3.3 in [19]. The results of Table II showcase that MJRTY BM is highly accurate in finding the most frequent elements of the stream. It also often outperforms its competitor. CGT's performance can increase with elevated values of W and T at expense of space and, most importantly, time. However, MJRTY BM can increase its accuracy too by stretching m. Finally, we note that CGT, being a probabilistic algorithm, may output IP elements that are not present in the stream (due to hash collisions). Conversely, MJRTY BM is not susceptible to this.

C. Detection accuracy

We shed light into the detection accuracy of our methods by considering real-world DDoS case studies as well as synthetic attacks injected on real data. The studied attacks were recorded at Merit's NetFlow collector. The first event, labeled as 'Library' case study, involved heavy UDP-based DNS and NTP traffic to an IP registered to a public library in Michigan, and is considered a volumetric attack (see Figure 6). The second event, named 'SSDP', is a low-volume attack directed to another host within the network (see Figure 8, left).

We implemented the *Defeat* [26] subspace method and juxtapose its performance against our algorithms on the two attacks. *Defeat* checks for anomalies using principal component analysis (PCA). A dictionary of entropies is built, over moving windows, of distributions of certain signature signals involving source and destination ports and IP addresses. In *Defeat*,

abnormalities are viewed as unusual distributions of these features. The *Defeat* framework is well-suited for multi-dimensional data due to its sketch-based design, and can be utilized for detection, identification and classification of attacks. However, its requirement for construction of empirical histograms makes it less appealing (if any feasible at all) for online realization.

Table III tabulates our analyses. We report two metrics, namely *precision* and *recall*. Precision depicts the fraction of alerts raised that are indeed relevant, and recall captures the ratio of actual anomalies that were detected. The ground truth (i.e., instances that the attack was ongoing) was obtained by offline data analysis that revealed the times when the target IPs and the corresponding protocol ports ranked among the top-10. Again, we considered time windows of 100,000 flow records. Due to the fact that additional attacks unknown to us might be present in the data, the precision criterion should be interpreted as a worst-case, lower bound.

The *Defeat* method was calibrated by the significance level α (necessary for its monitoring statistic threshold) and the number of 'votes' raised by *Defeat*'s internal detection processes. A sketch of size 484 was employed. For our system, we ran all three detection methods and reported their results; we also demonstrate the overall AMON accuracy by accounting the union of alerts. As illustrated in Table III, both *Defeat* and AMON perform remarkably well on 'Library'. Recall that this event is a voluminous one, and hence both techniques can easily detect it. On the other hand, the 'SSDP' case is a harder one (see Figure 8). *Defeat* reports true alerts for a higher time fraction, and both methods show consistent and similar false positive rates. The fact that *Defeat* checks for more traffic features than our methods seems to be the explanation of its higher attack discovery rate. However, we emphasize that AMON indeed rapidly uncovered the underlying event during its period of appearance. Further, it is extensible and adding monitoring features like source and destination ports into its design is straightforward.

The *Defeat* method works well but is rather sophisticated and requires a *training period*. This training process is computationally intensive and has to be performed offline. Moreover, the construction of signal distributions and entropy calculations, required to *run* the PCA-based detection requires very large memory structures, which do not scale well in real network conditions. Making this method work in real-time requires a formidable and independent effort. Further, finding a sufficiently long, anomalous-free period that satisfies the stationarity assumption might

TABLE III: Detection accuracy; comparison with *Defeat* [26].

(a) The Library case study

Method	Prec.	Recall
Defeat ($\alpha = 0.01$)	0.95	0.94
Defeat ($\alpha = 0.001$)	0.80	0.95
Fréchet ($p = 0.95, \lambda_\alpha = 0.6$)	0.89	0.22
Rel. Vol. ($\lambda_p = 0.6, L = 1.64$)	0.80	0.48
Connectivity ($p = 0.9999$)	0.74	0.95
AMON (all methods)	**0.94**	**0.93**
Fréchet ($p = 0.85, \lambda_\alpha = 0.6$)	0.65	0.41
Rel. Vol. ($\lambda_p = 0.6, L = 1.64$)	0.80	0.48
Connectivity ($p = 0.9999$)	0.74	0.95
AMON (all methods)	**0.94**	**0.93**
Fréchet ($p = 0.95, \lambda_\alpha = 0.6$)	0.89	0.22
Rel. Vol. ($\lambda_p = 0.6, L = 2$)	0.80	0.29
Connectivity ($p = 0.9999$)	0.74	0.95
AMON (all methods)	**0.95**	**0.93**

(b) The SSDP case study

Method	Prec.	Recall
Defeat ($\alpha = 0.001$, 9 votes)	0.35	0.80
Defeat ($\alpha = 0.001$, 10 votes)	0.31	0.21
Fréchet ($p = 0.95, \lambda_\alpha = 0.6$)	0.40	0.03
Rel. Vol. ($\lambda_p = 0.6, L = 1.64$)	0.47	0.12
Connectivity ($p = 0.9999$)	0.34	0.45
AMON (all methods)	**0.33**	**0.46**
Fréchet ($p = 0.85, \lambda_\alpha = 0.6$)	0.36	0.13
Rel. Vol. ($\lambda_p = 0.6, L = 1.64$)	0.47	0.12
Connectivity ($p = 0.9999$)	0.34	0.45
AMON (all methods)	**0.33**	**0.47**
Fréchet ($p = 0.95, \lambda_\alpha = 0.6$)	0.40	0.03
Rel. Vol. ($\lambda_p = 0.6, L = 2$)	0.52	0.06
Connectivity ($p = 0.9999$)	0.34	0.45
AMON (all methods)	**0.34**	**0.46**

be challenging. Its adaptability in dynamically changing traffic conditions is another concern. In contrast, our methods are highly adaptive to traffic trends and require no training.

To grasp insights into AMON's sensitivity to various tuning parameters we can study Table IV and Table V. For this evaluation, we utilized data collected during a seemingly ordinary period. We randomly injected attacks of various magnitudes at 5 times; the *injected traffic volume occurs directly on the databrick matrices*. We first considered the scenario of many sources sending traffic to one destination. In this scenario, one databrick row is 'inflated' by the synthetic attack magnitude at 5 random instances. The algorithm input was the destinations hash-binned arrays (see Figure 2), and we allowed a grace period of 3 minutes for detection. Each individual experiment was repeated 50 times and we report the average performance in terms of precision, $P_d^{(1)}$, and recall, $R_d^{(1)}$; sub-script 'd' denotes that the algorithm input was the destinations' signal. We also considered the scenario of one source communicating with multiple destinations (see $P_s^{(2)}$ and $R_s^{(2)}$), and the case of several sources to various destinations (rightmost four columns).

Table IV tabulates results for Algorithm 1, which is tuned by the significance level $p_0 = p$ and the smoothing parameter $\lambda = \lambda_\alpha$. Best performance is achieved with $p = 0.95$ and $\lambda_\alpha = 0.50$. Note that p may be calibrated to ease the false alarm rate. Further, observe the connection

TABLE IV: Fréchet method (Algorithm 1)

p	λ_α	Gbps	$P_d^{(1)}$	$R_d^{(1)}$	$P_s^{(2)}$	$R_s^{(2)}$	$P_s^{(3)}$	$R_s^{(3)}$	$P_d^{(3)}$	$R_d^{(3)}$
0.95	0.50	0.50	0.74	1.00	1.00	0.96	1.00	1.00	0.74	1.00
0.95	0.50	1.50	0.73	1.00	1.00	1.00	1.00	1.00	0.74	1.00
0.95	0.50	2.50	0.73	1.00	1.00	0.98	1.00	1.00	0.74	1.00
0.95	0.60	0.50	0.72	0.93	1.00	0.61	1.00	1.00	0.85	1.00
0.95	0.60	1.50	0.74	0.99	1.00	0.88	1.00	0.99	0.85	0.99
0.95	0.60	2.50	0.73	1.00	1.00	0.92	1.00	1.00	0.85	1.00
0.99	0.50	0.50	1.00	0.73	0.74	0.22	1.00	1.00	1.00	0.99
0.99	0.50	1.50	1.00	0.94	0.98	0.50	1.00	1.00	1.00	1.00
0.99	0.50	2.50	1.00	0.97	1.00	0.71	1.00	0.99	1.00	0.99
0.99	0.60	0.50	0.76	0.24	0.36	0.09	1.00	1.00	1.00	1.00
0.99	0.60	1.50	0.92	0.40	0.08	0.02	1.00	1.00	1.00	1.00
0.99	0.60	2.50	1.00	0.55	0.16	0.04	1.00	1.00	1.00	0.99

between robustness and adaptivity as dictated by λ_α. Recall that this parameter is used to smooth the heavy-tail exponent, α. Big traffic spikes translate to a heavier distribution tail and thus a low α; this could make our scheme too insensitive/conservative if we do not have an adaptive scheme that accounts for 'historical' α's. On the other hand, high α can make our scheme too sensitive (i.e., many false positives). Table V illustrates the detection performance of a modification of Algorithm 2, which utilizes EWMA control charts on z-scores, as explained at the end of Section IV-C. We employ our methodology for the EWMA (λ_p, L) pairs shown and $\lambda = \lambda_\alpha$ was fixed to 0.5. For this evaluation, our synthetic attacks were persistent for 5 consecutive time slots (i.e., 50 seconds). Users can tame the alert rate by increasing the control limits with a higher L and/or decrease further λ_p.

In addition, we undertook sensitivity analysis with respect to various choices of monitoring intervals used to generate databrick matrices. Table VI illustrates that at the relevant time-scale of interest (e.g., few seconds), detection accuracy remains relatively unchanged. For optimal performance, very brief aggregation intervals for *low-traffic links* are discouraged because the hash-binned arrays will be sparse. Similarly, very large aggregation levels (e.g., minutes) are also

TABLE V: Relative volume method (Algorithm 2)

L	λ_p	Gbps	$P_d^{(1)}$	$R_d^{(1)}$	$P_s^{(2)}$	$R_s^{(2)}$	$P_s^{(3)}$	$R_s^{(3)}$	$P_d^{(3)}$	$R_d^{(3)}$
2.00	0.50	0.50	0.45	0.98	0.61	0.97	0.49	0.99	0.35	0.99
2.00	0.50	1.50	0.45	0.99	0.62	0.96	0.41	0.99	0.33	1.00
2.00	0.50	2.50	0.45	0.98	0.64	0.98	0.41	0.99	0.34	1.00
2.00	0.60	0.50	0.61	0.98	0.80	0.96	0.74	0.99	0.45	1.00
2.00	0.60	1.50	0.62	0.97	0.81	0.99	0.50	0.99	0.41	0.99
2.00	0.60	2.50	0.60	0.98	0.81	0.97	0.48	0.99	0.40	0.99
3.00	0.50	0.50	1.00	0.62	0.76	0.25	0.95	0.99	0.78	0.99
3.00	0.50	1.50	0.98	0.74	0.62	0.16	0.73	0.98	0.60	0.98
3.00	0.50	2.50	1.00	0.81	0.90	0.36	0.58	0.98	0.55	0.99
3.00	0.60	0.50	0.96	0.42	0.50	0.13	0.99	0.98	0.92	0.98
3.00	0.60	1.50	1.00	0.59	0.40	0.10	0.79	1.00	0.65	1.00
3.00	0.60	2.50	1.00	0.72	0.56	0.13	0.71	1.00	0.59	1.00

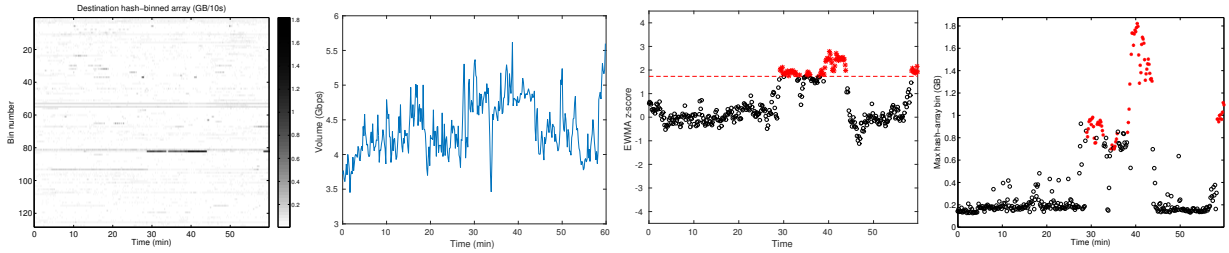


Fig. 6: The 'Library' event. The left panel shows the time-series of the hashed-array for destinations for a period of one hour. Note the dark horizontal stripe (at bin 82) between minutes 30 and 45 and towards the end of the hour. The adjacent panel depicts the aggregate traffic volume over the hour of interest. Observe the elevated traffic volume and note that both the Fréchet and the relative volume methods correctly raised alerts (red) during the malicious activity period (rightmost figures).

unwelcome since short-duration attacks might be masked by collisions with other events. Further, the heavy-tail modeling assumption is not suitable on large time-scales (see Section IV-A).

We conclude this section be showcasing that AMON's identification may synergistically be coupled with the detection alerts to guide operators' troubleshooting and mitigation efforts. By looking at the database of heavy hitters reported by MJRTY BM, network managers can readily have an IP list of candidate culprits. Upon detection, our algorithm outputs a databrick bin, aimed to identify culprits. One can then examine the flows reported by Boyer-Moore of the sub-streams associated with the relevant hash bin. For example, one first sorts these Boyer-

TABLE VI: Sensitivity on aggregation level ('Library' case)

	Fréchet method		Relative Volume	
	$(p = 0.85, \lambda_\alpha = 0.6)$		$(\lambda_p = 0.6, L = 1.64)$	
Aggregation Level	Precision	Recall	Precision	Recall
200K NetFlow records	0.74	0.52	0.82	0.53
300K NetFlow records	0.76	0.73	0.87	0.70
400K NetFlow records	0.92	0.71	0.81	0.75
500K NetFlow records	0.81	0.73	0.70	0.76

TABLE VII: 'Library' case study: culprit identification using MJRTY Boyer-Moore.

Top-K	1	2	4	8	16	32	64	128
Fréchet method (Alg. 1) - time fraction (%)	39.7	60.3	74.0	90.4	93.2	98.6	100	100
Rel. Vol. (Alg. 2) - time fraction (%)	42.7	61.3	74.7	90.7	93.3	98.7	100	100

Moore-identified flows based on their traffic volume estimate (P_{est} in Algorithm 3), and then proceeds with forensics analysis. In particular, for the 'Library' event, in 39.7% of the flagged times by Algorithm 1, the top ranked flow by Boyer-Moore was indeed one of the (src, dst) pairs of interest (see Table VII, top row). For a 60.3% fraction of times, the same detection method was able to identify flow(s) of interest among the top-two reported BM flows, etc. Similar reports are offered by the second detection algorithm at the bottom row.

D. Diagnosing low-volume attacks

In addition to orchestrated, volumetric attacks that seek to overwhelm the victims with traffic, low-volume DDoS attacks can be pernicious, albeit problematic to detect. Such attacks, like the 'SSDP' event previously analyzed, rely on presumably innocuous message transmissions to thwart standard anomaly detection methods. In this section, we highlight the importance of *visualizations* and of *methods that detect structural patterns* (see Section IV-D) in traffic in revealing these low-profile nefarious actions. As an initial example, consider Figure 8 (bottom left). The bold horizontal line in the depicted databrick is an artifact of the distributed nature of the 'SSDP' attack. Operators can easily, instantaneously and visually observe such patterns by monitoring AMON's live data products. Further, we reiterate the important role that the connectivity algorithm played in automatically uncovering the 'SSDP' instances (see Table IIIb).

Figure 7 depicts another case study of this kind in which sparse traffic patterns (left panel) make Algorithms 1 and 2 to miss these events. As clearly seen by the in-degree counters of

Figure 7 (middle), two possibly suspicious events are occurring. Manual inspection revealed the first event to be UDP misuse affecting a Tor exit router within Merit, and the other (longest running) attempts of SSH-breaking into Michigan-located servers from IPs that belong at an autonomous system registered in the Asia-Pacific region. We refer to this case study as 'Tor'. The right panel illustrates that both events were flagged by our community detection system. This plot shows the number of highly connected destinations (i.e., high in-degree) over the duration of an hour. The correct hash bins were also reported (22 and 53).

Further, Figure 8 (right panels) demonstrates extra visualization aids readily available by our data products; cliques and clique sizes for the sources and destinations *co-connectivity* graphs are depicted. A co-connectivity graph for sources provides insights into the number of *common destinations* between two sources; the destinations co-connectivity graph sheds similar information for destinations. To obtain these (undirected) graphs we utilize the binary matrix $A_t = (a_t(i,j))_{m \times m}$ (see Section IV-D). The co-connectivity graph for destinations D_t is efficiently obtained as $D_t := A_t \cdot A_t^T$; the graph for sources is $S_t := A_t^T \cdot A_t$, where A_t^T is the transpose of A_t. Based on the co-connectivity graphs one can obtain visualizations about the cliques formed, over time. Figure 8 showcases two such snapshots. These graphs are portrayed as their matrix adjacency representations, and we have re-arranged the node labeling based on the node-degree in decreasing order (i.e., the first row represents the adjacency associations of the node with the highest degree). Note the very large clique formed in src-to-src graph. This depicts the situation in the 'Tor' case study discussed above, when a plethora of sources were contacting the same destination (the Tor exit router). One may also extract the maximum clique for each graph; the bottom row demonstrates this characteristic over time. The reader is pointed to our supplement [48] for an animated version of Figure 8, where clearly one can observe the clique sizes evolving and expanding over the duration of the 'Tor' event.

VI. Conclusions

The paper presents a novel open source monitoring architecture suitable for multi-gigabit (i.e., 10Gbps+) network traffic streams. It is based on PF_RING Zero-Copy and tailored for deployment on commodity hardware for troubleshooting high-impact events that may arise from malicious actions such as DDoS attacks. It is worth noting that the NIC needed for processing

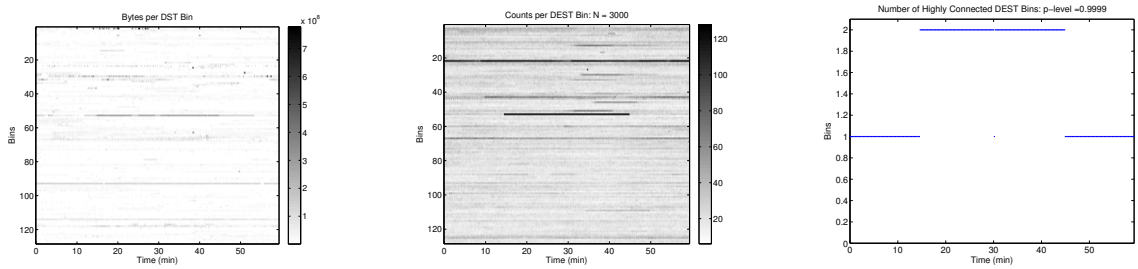


Fig. 7: Low-volume attacks ('Tor' and 'SSH-scanning' events): community detection.

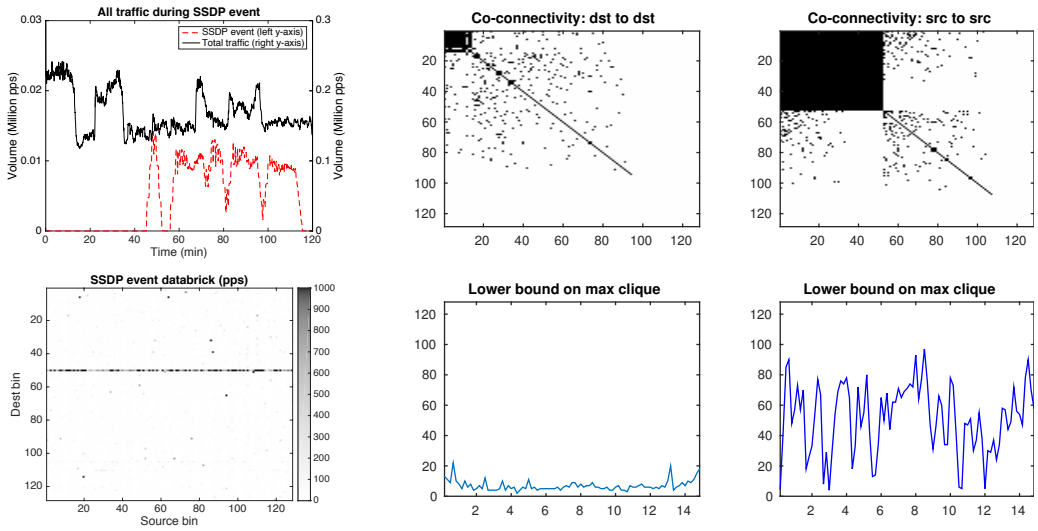


Fig. 8: Visualizations readily available by our data products. *Left:* Merit Network 10:00-12:00 EST, Dec 9, 2015 – the faint 'SSDP' event (in volume) is clearly observable in the time snapshot of the databrick matrix (horizontal line). *Top Right:* Adjacency matrices of co-connectivity graphs (node indices sorted by degree—black corresponds to locations of 1's). *Bottom Right:* Size of max cliques over time during the 'Tor' case study (Section V-D). By observing clique size changes in Dashboards like this, coupled with the detection method of Section IV-D, such seemingly innocuous low-volume events are captured.

traffic data at speeds around 25Gbps (Figure 1b) costs roughly **800 USD**, while specialized FPGA accelerated cards or monitoring appliances cost an order of magnitude higher (above 10,000 USD). We demonstrated the performance of our system architecture and the underlying statistical methods on selected real-world case studies and measurements from the Merit Network.

Our framework is extensible, and allows for further statistical, filtering and visualization modules. Currently, we are in the process of deploying an interactive filtering mode of operation that would enable network operators to zoom-in and examine IP ranges of interest in real-time. As an example, consider the 'Tor' and 'SSH-scanning' events of Figure 7 for which our methods automatically flagged bins 22 and 53. With the filtering option, operators can rapidly zoom exclusively into the sub-stream of traffic that gets mapped into the flagged bins.

Note that due to randomization, these hash-bins are not associated with traditional IP-ranges (e.g., subnets or specific IP addresses). Thus, such filters cannot be implemented using existing filtering infrastructures such as BPF or hardware-based filters.

Acknowledgments: The authors thank L. Deri and the `ntop.org` team for providing the necessary PF_RING ZC licenses free-of-charge. We also thank M. Morgan and Y. Kebede for their valuable assistance. This work is supported by a Department of Homeland Security Science and Technology Directorate FA8750-12-2-0314 grant (MK), and National Science Foundation grants CNS-1422078 (MK, GM) and DMS-1462368 (SS). We also thank all anonymous reviewers.

Supplementary material: Software prototype, animations and multimedia provided in [48].

APPENDIX

Proof of Proposition 1. This result is a simple consequence of Theorem 3.3.7, p. 131 in [45].

Proof: By the independence of the $X(i)$'s, for all fixed $x > 0$, we have

$$\mathbb{P}(m^{-1/\alpha} D_m(X) \leq x) = \mathbb{P}(X \leq m^{1/\alpha} x)^m = (1 - \mathbb{P}(X > m^{1/\alpha} x))^m.$$

Now, by (1) with x replaced by $m^{1/\alpha} x$, we observe that $\mathbb{P}(X > m^{1/\alpha} x) \sim c/(m x^\alpha)$, as $m \to \infty$. Thus, using the fact that $(1 - c x^{-\alpha}/m)^m \to e^{-c/x^\alpha},\ m \to \infty$, we conclude that

$$\mathbb{P}(m^{-1/\alpha} D_m(X) \leq x) \longrightarrow e^{-c/x^\alpha}, \quad \text{as } m \to \infty.$$

This implies the desired convergence in (3), since $\mathbb{P}(c^{1/\alpha} Z_\alpha \leq x) = e^{-c/x^\alpha},\ x > 0$. ■

Proof of Proposition 6.

Proof: Part **(i)** is a direct consequence of (6). Now, to prove **(ii)**, observe that by (7),

$$\frac{V(k;m)}{V(\ell;m)} \stackrel{d}{=} \frac{\sum_{j=1}^{k} \overline{F}^{-1}(\Gamma_j/\Gamma_{m+1})}{\sum_{j=1}^{\ell} \overline{F}^{-1}(\Gamma_j/\Gamma_{m+1})}. \tag{9}$$

By the Strong Law of Large Numbers, we have that $\Gamma_j/\Gamma_{m+1} \sim \Gamma_j/m$, as $m \to \infty$, almost surely, for all $j = 1, \ldots, \ell$. Recall that ℓ is fixed. Thus, in view of (1), $\overline{F}^{-1}(p) \sim (p/c)^{-1/\alpha}$, as $p \downarrow 0$, and hence for all $j = 1, \ldots, \ell$, with probability one, we have

$$\overline{F}^{-1}\Big(\frac{\Gamma_j}{\Gamma_{m+1}}\Big) \sim \Big(\frac{\Gamma_j}{c\Gamma_{m+1}}\Big)^{-1/\alpha}, \quad \text{as } m \to \infty.$$

This implies that the right-hand side of (9) converges almost surely to

$$\frac{\sum_{j=1}^{k} \Gamma_j^{-1/\alpha} (c\Gamma_{m+1})^{1/\alpha}}{\sum_{j=1}^{\ell} \Gamma_j^{-1/\alpha} (c\Gamma_{m+1})^{1/\alpha}} = W_\alpha(k, \ell),$$

which completes the proof of (8). ■

REFERENCES

[1] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss, "Surfing wavelets on streams: one-pass summaries for approximate aggregate queries," in *Procedings of VLDB, Rome, Italy*, 2001.

[2] S. Muthukrishnan, "Data streams: Algorithms and applications," *Found. Trends Theor. CS*, vol. 1, no. 2, Aug. 2005.

[3] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based change detection: methods, evaluation, and applications," in *3rd ACM SIGCOMM IMC*, NY, USA, 2003, pp. 234–247.

[4] A. C. Gilbert, M. J. Strauss, J. A. Tropp, and R. Vershynin, "One sketch for all: Fast algorithms for compressed sensing," in *STOC '07*, NY, USA, 2007, pp. 237–246.

[5] S. Stoev, M. Hadjieleftheriou, G. Kollios, and M. Taqqu, "Norm, point, and distance estimation over multiple signals using max-stable distributions," in *IEEE 23rd International Conference on Data Engineering*, April 2007, pp. 1006–1015.

[6] B. Xi, G. Michailidis, and V. N. Nair, "Estimating network loss rates using active tomography," *J. Amer. Statist. Assoc.*, vol. 101, no. 476, pp. 1430–1448, 2006.

[7] E. Lawrence, G. Michailidis, V. N. Nair, and B. Xi, "Network tomography: a review and recent developments," in *Frontiers in statistics*. London: Imp. Coll. Press, 2006, pp. 345–366.

[8] S. Stoev, M. Taqqu, C. Park, G. Michailidis, and J. S. Marron, "LASS: a tool for the local analysis of self-similarity," *Computational Statistics and Data Analysis*, vol. 50, pp. 2447–2471, 2006.

[9] S. Stoev and G. Michailidis, "On the estimation of the heavy–tail exponent in time series using the max–spectrum," *Applied Stochastic Models in Business and Industry*, vol. 26, no. 3, pp. 224–253, 2010.

[10] P. Ferguson and D. Senie, "RFC 2827: Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing," https://tools.ietf.org/html/bcp38.

[11] J. Czyz, M. Kallitsis, M. Gharaibeh, C. Papadopoulos, M. Bailey, and M. Karir, "Taming the 800 pound gorilla: The rise and decline of ntp ddos attacks," in *IMC '14*. NY, USA: ACM, 2014, pp. 435–448.

[12] C. Rossow, "Amplification Hell: Revisiting Network Protocols for DDoS Abuse," in *Proceedings of the 2014 Network and Distributed System Security (NDSS) Symposium*, February 2014.

[13] J. M. Lucas and M. S. Saccucci, "Exponentially weighted moving average control schemes: Properties and enhancements," *Technometrics*, vol. 32, no. 1, pp. 1–29, Jan. 1990.

[14] G. E. P. Box and G. Jenkins, *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990.

[15] J. D. Brutlag, "Aberrant behavior detection in time series for network monitoring," in *Proceedings of the 14th USENIX Conference on System Administration*, ser. LISA '00. Berkeley, CA, USA: USENIX Association, 2000, pp. 139–146.

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

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

[18] ——, "Mining anomalies using traffic feature distributions," in *SIGCOMM '05*. NY, USA: ACM, 2005, pp. 217–228.

[19] G. Cormode and S. Muthukrishnan, "What's hot and what's not: Tracking most frequent items dynamically," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 249–278, Mar. 2005.

[20] R. M. Karp, S. Shenker, and C. H. Papadimitriou, "A simple algorithm for finding frequent elements in streams and bags," *ACM Trans. Database Syst.*, vol. 28, no. 1, pp. 51–55, Mar. 2003.

[21] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, "Finding hierarchical heavy hitters in data streams," in *VLDB 03*, 2003, pp. 464–475.

[22] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 323–336, Aug. 2002. [Online]. Available: http://doi.acm.org/10.1145/964725.633056

[23] R. Schweller, Z. Li, Y. Chen, Y. Gao, A. Gupta, Y. Zhang, P. Dinda, M.-Y. Kao, and G. Memik, "Reverse hashing for high-speed network monitoring: Algorithms, evaluation, and applications," in *INFOCOM 2006*, 2006, pp. 1–12.

[24] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund, "Online identification of hierarchical heavy hitters: Algorithms, evaluation, and applications," in *IMC '04*. NY, USA: ACM, 2004, pp. 101–114.

[25] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, Apr. 2005.

[26] X. Li, F. Bian, M. Crovella, C. Diot, R. Govindan, G. Iannaccone, and A. Lakhina, "Detection and identification of network anomalies using sketch subspaces," in *IMC '06*, NY, USA, 2006, pp. 147–152.

[27] D. van der Steeg, R. Hofstede, A. Sperotto, and A. Pras, "Real-time DDoS attack detection for Cisco IOS using NetFlow," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, May 2015, pp. 972–977.

[28] A. C. Gilbert, M. J. Strauss, J. A. Tropp, and R. Vershynin, "Algorithmic linear dimension reduction in the ℓ-1 norm for sparse vectors," in *Allerton 2006*, 2006.

[29] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan, "Counting distinct elements in a data stream," in *RANDOM '02*. London, UK, UK: Springer-Verlag, 2002, pp. 1–10.

[30] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," in *28th Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: ACM, 1996, pp. 20–29.

[31] J. Tropp and A. Gilbert, "Signal recovery from random measurements via orthogonal matching pursuit," *Information Theory, IEEE Transactions on*, vol. 53, no. 12, pp. 4655–4666, Dec 2007.

[32] D. Donoho, "Compressed sensing," *Information Theory, IEEE Transactions on*, vol. 52, no. 4, pp. 1289–1306, April 2006.

[33] P. Indyk, "Explicit constructions for compressed sensing of sparse signals," in *19-th ACM-SIAM SODA*. PA, USA: Society for Industrial and Applied Mathematics, 2008, pp. 30–33.

[34] G. Cormode and S. Muthukrishnan, "Space efficient mining of multigraph streams," ser. PODS '05. New York, NY, USA: ACM, 2005, pp. 271–282.

[35] S. Ranshous, S. Shen, D. Koutra, S. Harenberg, C. Faloutsos, and N. F. Samatova, "Anomaly detection in dynamic networks: a survey," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 7, no. 3, pp. 223–247, 2015.

[36] L. Deri, "Improving passive packet capture: Beyond device polling," in *In Proceedings of SANE 2004*, 2004.

[37] R. Boyer and J. Moore, "MJRTY - a fast majority vote algorithm," in *Automated Reasoning*, ser. Automated Reasoning Series, R. Boyer, Ed., 1991, vol. 1, pp. 105–117.

[38] J. Carter and M. N. Wegman, "Universal classes of hash functions," *Journal of Computer and System Sciences*, 1979.

[39] M. Kallitsis, S. Stoev, and G. Michailidis, "Hashing Pursuit for Online Identification of Heavy-Hitters in High-Speed Network Streams," July 2014, http://arxiv.org/abs/1412.6148.

[40] F. Fusco and L. Deri, "High speed network traffic analysis with commodity multi-core systems," in *10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 218–224.

[41] S. Gallenmüller, P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle, "Comparison of frameworks for high-performance packet IO," in *ANCS '15*. Washington, DC, USA: IEEE Computer Society, 2015, pp. 29–38.

[42] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the self-similar nature of ethernet traffic (extended version)," *Networking, IEEE/ACM Transactions on*, vol. 2, no. 1, pp. 1–15, Feb 1994.

[43] M. Crovella and A. Bestavros, "Self-similarity in world wide web traffic: evidence and possible causes," *Networking, IEEE/ACM Transactions on*, vol. 5, no. 6, pp. 835–846, Dec 1997.

[44] S. Stoev, G. Michailidis, and M. Taqqu, "Estimating heavy–tail exponent through max self–similarity," *IEEE Transactions on Information Theory*, vol. 57, no. 3, pp. 1615–1636, 2011.

[45] P. Embrechts, C. Klüppelberg, and T. Mikosch, *Modelling Extreme Events*. New York: Springer-Verlag, 1997.

[46] M. Kallitsis, S. Stoev, S. Bhattacharya, and G. Michailidis, "AMON: An Open Source Architecture for Online Monitoring, Statistical Analysis and Forensics of Multi-gigabit Streams," September 2015, http://arxiv.org/abs/1509.00268.

[47] D. Lambert and C. Liu, "Adaptive thresholds: monitoring streams of network counts," *J. Amer. Statist. Assoc.*, vol. 101, no. 473, pp. 78–88, 2006. [Online]. Available: http://dx.doi.org/10.1198/016214505000000943

[48] M. Kallitsis, S. Stoev, and G. Michailidis, "This paper (Supp. Material)," September 2015, http://tinyurl.com/pdccpce.