Streaming Solutions for Fine-Grained Network Traffic Measurements and Analysis

Faisal Khan, Nicholas Hosein, Soheil Ghiasi, Chen-Nee Chuah, Puneet Sharma

***Abstract*—Online network traffic measurements and analysis is critical for detecting and preventing any real-time anomalies in the network. We propose, implement, and evaluate an online, adaptive measurement platform, which utilizes real-time traffic analysis results to refine subsequent traffic measurements. Central to our solution is the concept of Multi-Resolution Tiling (MRT), a heuristic approach that performs sequential analysis of traffic data to zoom into traffic sub-regions of interest. However, MRT is sensitive to transient traffic spikes. In this paper, we propose three novel traffic streaming algorithms that overcome the limitations of MRT and can cater to varying degrees of computational and storage budgets, detection latency, and accuracy of query response. We evaluate our streaming algorithms on a highly parallel and programmable hardware as well as a traditional software based platforms. The algorithms demonstrate significant accuracy improvement over MRT in detecting anomalies consisting of synthetic hard-to-track elephant flows and global icebergs. Our proposed algorithms maintain the worst case complexities of the MRT, while incurring only a moderate increase in average resource utilization.**

I. Introduction

Accurate traffic measurement and monitoring is key to a wide range of network applications such as traffic engineering, anomaly detection, and security analysis. A number of critical network management decisions, such as blocking traffic to a victim destination, require extraction and analysis of real-time spatio-temporal patterns in network traffic. The large traffic volumes seen in today's high speed networks pose enormous computational and storage requirements for accurate traffic measurements.

Traditionally, traffic measurements are performed by configuring conservative sampling factors [1] at the routers with very limited local storage. The collected samples are periodically sent to high-end servers where they are post-processed to answer some higher level user queries (e.g., traffic volume from a customer domain) or to perform network troubleshooting and anomaly detection. Figure 1(a) illustrates this traditional paradigm.

Sampling solutions, though straightforward, often introduce inaccuracies in estimating various flow statistics or in preserving traffic features critical for anomaly detection [2]. To bridge the gap between accuracy and detection latency, the concept of programmable measurements was proposed [3] to configure measurement rules that are representative of user requirements. Such measurement specifications may not readily be available, especially in situations where the adaptation is based on network behavior rather than a fixed pattern. For instance, during the search of a volumetric anomaly such as a heavy-hitter, the measurements need to quickly adapt and track the evolving anomaly instead of being updated periodically with static sampling ratios.

Recently, iterative measurements have gained attention as alternate to sampling based solutions [4], [5]. The idea behind iterative measurements is to perform multiple sequential measurements and analysis of progressively finer resolutions. The contention is that repetitive measurements, analysis and automated refinement of measurement goals, smartly prunes away uninteresting data in a manner that is tied with the user requirements.

A high-level description of the iterative measurement paradigm is presented in Figure 1(b). Central to the scheme is a tight integration between the measurement requirements and the actual data collection. This is achieved by breaking a higher level user-query into multiple *measurement rules* and then refining these rules iteratively over time until the user-query gets answered. The underlying rationale is that the interesting traffic patterns could be detected or learned on the fly, via iterative rule based traffic measurements, online analysis of collected information, and iterative evolution of subsequent rules for further and finer traffic inspection. Each round in the iterative process guides the subsequent measurements towards the goal, thereby reducing redundant measurements and leading to a more accurate response to user-query.

The iterative scheme temporally distributes the complexity in answering the user query by breaking it down into multiple rules, or *rule-sets*, that are answered over multiple iterations. The efficiency of the rule evaluation and synthesis is therefore crucial in defining the overall effectiveness of the iterative scheme. Multi-Resolution Tiling (MRT) Algorithm [6] has been previously proposed in configuring the rule-sets by performing iterative analysis over collected data. In an online [1] setting, such an iterative refinement of measurement rules is sensitive to dynamic changes in traffic composition. A naive solution to the problem could be to aggressively configure new measurement rules, which will lead to very large rule-sets. However, the online systems often have limited rule-processing resources. The challenge is therefore in determining the optimal set of rules that can accurately answer the user query in reasonable time, while adhering to computational and storage budgets.

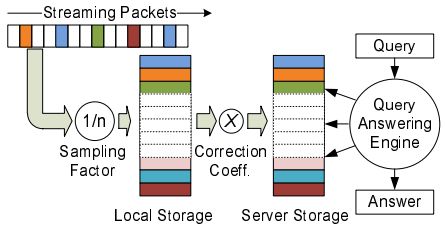
F. Khan and N. Hosein are graduate student at the Department of Electrical and Computer Engineering, University of California, Davis

S. Ghiasi is Associate Professor at the Department of Electrical and Computer Engineering, University of California, Davis

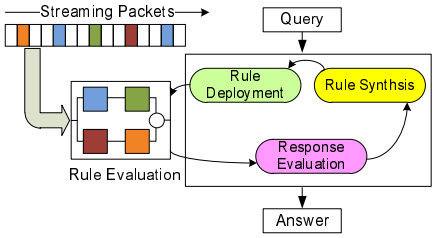
C. Chuah is Professor at the Department of Electrical and Computer Engineering, University of California, Davis

P. Sharma is Principal Research Scientist at HP-Labs, Palo Alto

[1] also referred as 'streaming' in the paper



(a) Traditional Sampling Based Methods



(b) Iterative Measurement Framework

Fig. 1: Network Measurement Paradigms

We previously presented an online iterative measurement framework that provided solution to the above challenges [7]. Fundamental to our streaming solution are three novel algorithms that can cater to different levels of computational and storage budgets, detection latencies, and user level knowledge of the anomaly. Our key design goal is to direct the limited resources to where they are needed the most. Another consideration in the design of our algorithms is to maintain scalability in terms of computational and storage costs, while achieving the desired anomaly detection accuracy and latency. Our results demonstrated the merits of our streaming solutions which could quickly and accurately isolate hard-to-track volumetric anomalies by offering higher stability to traffic fluctuations, as compared to MRT.

The contributions of this paper are summarized as follows:

- We propose three traffic streaming algorithms, *Equilibrium Rollback*, *Flow Momentum*, and *Directed Momentum*, to guide the iterative configuration of measurement rule-sets by taking into account resource constraints, detection latency, and measurement goals. We demonstrate how our proposed algorithms address the shortcomings of MRT.
- With the recent trend towards software defined networking paradigm and the adoption of OpenFlow [8] in various switches, we extend the evaluation of the iterative measurement framework on both hardware and software platforms. We examine the actual rule-processing costs associated with different algorithms, and examine the trade-offs between cost and accuracy of our algorithms when implemented on these two different rule-processing platforms.
- We extend our algorithmic analysis to a distributed framework and show the effectiveness of our algorithms in detecting hard-to-isolate global icebergs. Our results demonstrate 100% detection accuracy of our algorithms across the platform choices, with low to moderate utilization of computation and storage budgets.

The paper is organized as follow. We discuss related work on traffic measurements and describe MRT algorithm in Section-II. This is followed by problem statement and motivation in Section-III. Section-IV presents the three proposed streaming algorithms. We then discuss the state-of-the-art hardware based rule-processing BURAQ platform [9] as well as software-based rule-processing in Section-V, with emphasis on algorithmic implications arising from platform-specific details. Using real and synthetic traces, we evaluate the algorithms for both software and hardware based rule-mapping in Section-VI. We conclude the paper in VII. An analytical evaluation of the algorithms is presented in our technical report [10], providing mathematical upper bounds on the expected false alarms associated with our solutions. The bounds yield interesting insights that can be leveraged to fine-tune the desired trade-offs between the system accuracy and detection latency under specific network conditions and computational budgets.

II. Background and Related Work

Network traffic measurement fundamentally involves collecting information about a subset of traffic that satisfies some criteria. Traffic is generally grouped in terms of *flows*, where a flow refers to a set of packets that have the same n-tuple values in their header fields. Typical definitions of the flow include 6-tuple: $\{prt, tos, sip, spt, dip, dpt\}$ where, prt is the protocol field, tos is type of service, sip and dip are the source and destination IP addresses, and spt and dpt are the source and destination ports, respectively. We define a *flowset* to be an aggregation of flows. For instance, the CIDR prefix is a particular type of a flowset that aggregates all the flows that have matching significant bits corresponding to the size of the prefix.

Traditional measurement schemes typically maintain unique "*per-flow*" based statistics. The collected information is post-processed *offline* for answering higher-level user-queries [11] such as detecting an anomalous behavior. The per-flow schemes, however, require storing information about potentially huge number of flows, straining the limited SRAM budgets of measurement hardware. The scalability issues of the per-flow scheme have traditionally been addressed using packet or flow based sampling approaches [12]. Studies have shown, however, that sampling leads to inaccuracies in answering the user-queries [2]. Recently, smart sampling approaches, such as cSamp [13] and FlexSample [3], are proposed to balance the monitoring goals with resource constraints through smarter provisioning of resources based on application requirements. However, these schemes require measurement goals to be defined a priori, which can be challenging with highly dynamic network or traffic conditions.

More recently, iterative measurement schemes were proposed to address the challenges mentioned above [4], [5]. The key idea is to perform top-down and goal-oriented measurements that directly reflect the requirements of high level user-queries by taking in account dynamic traffic variations. In this context, Multi-Tiling Resolution (MRT) algorithm

[6] was proposed to answer the user query in an iterative manner through a progression of finite set of intermediate measurements, also referred to as *rules*. A rule can be viewed as an intermediate question in pursuit of the user-query, that if answered, can help lead the search in a more intelligent manner. We will discuss the rules shortly in the context of MRT (Section II-A).

One of the monitoring applications that needs high-fidelity traffic measurement as input is network anomaly detection. Modern networks are often plagued with a variety of Denial of Service (DoS) attacks, which try to deplete available network bandwidth through insertion of unwanted traffic into the network. The inserted traffic could be in the form of a few heavy flows, referred to as *Elephant* or *Heavy-Hitter (HH)* flows [14], or using a large number of small flows, the *Mice* flows. Yet, another kind of attack involves distributed participation of several hosts (or sources) such that the amount of traffic from an individual host (or destination) may be below the threshold, but the aggregate is above a threshold. We refer to such an anomaly as a *Global Iceberg (GI)*.

There is a rich amount of research work that addresses the above types of attacks. Elephant flow identification approaches have been proposed [14], [15], which are useful for both traffic engineering and anomaly detection. Global iceberg detection has been addressed using sampling [16], sketches [17] as well as hybrid sampling/sketching solutions [18].

A. Multi-Resolution Tiling Algorithm

Multi-Resolution Tiling (MRT)[2] [6] is a recursive top-down heuristic that relies on a simple but powerful observation that if a flowset does not contain an anomaly, then no flow in that flowset can be anomalous. For instance, in the case of elephant flows, if a flowset does not consume θ-fraction of the entire network bandwidth, then no flows within that flowset may be an elephant flow. In terms of CIDR notation, the algorithm states that if a prefix is not an elephant, then all its constituent prefixes of larger number of bits (finer granularities) can be discarded from further consideration.

An MRT iteration for two dimensional tuple space $\{source, destination\}$ involving a *Zoom Ratio* (ZR), or *Expansion Ratio*, of four is illustrated in Figure 2. The ZR defines the rate of exploration within a sub-region. For instance, the ZR of four in the figure dictates that a given tuple-space is partitioned into four sub-regions at a time, implying exploring additional two-bits in the tuple space per iteration. Statistics are collected for individual sub-regions for a given measurement interval. This is achieved using rules that partition the traffic into the four sub-regions. Thus a rule in the context of the MRT can be viewed as a Boolean bit-mask on specific header bits that helps qualify the incoming packets. In the case of HH, statistics corresponding to the traffic mapped to the sub-regions will be collected. The sub-regions that exceed the threshold θ, marked with a cross in the figure, are then selected for further *zooming-in*, or *expansion*, in the next-iteration. A zooming-in corresponding to a selected sub-region, means partitioning it further into sub-regions as per the ZR. Thus each iteration

[2]The algorithm is described in Appendix-A.

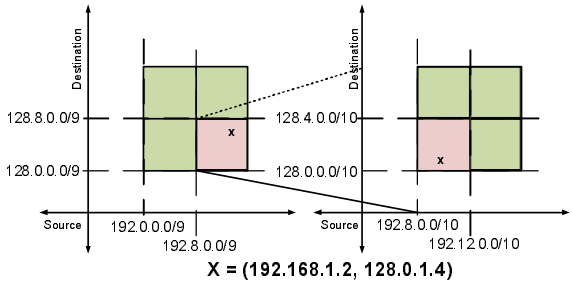


Fig. 2: MRT with zoom ratio of four

in the given example results in resolution of two bits. MRT thereafter continues iterating between partitioning, statistics-collection, and expansion phases until the anomalous flow is isolated, or the all the header bits are resolved. Thus, in the given two tuple-space example, a flow will get isolated in a maximum of 64 iterations.

The MRT's worst case expansion scenario corresponds to a spatio-temporal distribution of flowsets/flows such that every tracked MRT sub-region passes the threshold test. Assuming the tracked flowsets/flows remain consistent, the worst case leads to $log_{\Phi}(n)$ MRT iterations, where n being the number of bits of the search space resolved during the expansion, referred to as the expansion granularity (or just granularity). If MRT is viewed as tree structure with nodes defining the rules and levels describing the MRT iterations, then the algorithmic complexity of MRT's decision phase corresponds to the total number of nodes in the tree structure, given as $\Theta[\Phi^{log_{\Phi}(n)} - 1/(\Phi - 1)]$.

III. Motivation and Problem Statement

The key idea of the MRT is that one can, by observing a flowset, *infer* the characteristics of its subsets or objects (the flows). Therefore, one can selectively zoom into flowsets that might contain anomalies, such as heavy hitters, while ignoring others. As the algorithm explores the traffic landscape, it logs explored regions in a tree structure where nodes represent monitored regions in the IP-space. The parent nodes represent regions in IP-space that are supersets of the region covered by their children nodes, with the root node of the tree covering the entire IP-space. The expansion ratio corresponds to the number of children, or arcs, originating from a parent node.

As shown earlier, the MRT algorithm helps guide traffic measurements in the vast n-tuple search space. However, the limited visibility, with which the iterative guided measurements have to base their decisions on, can lead to false negatives and positives in detecting an anomaly. For instance, a brief spike in activity may lead the MRT to incorrectly declare the presence of a heavy flow, when it may only be a transient Flash crowd [19]. Similarly, a brief absence of an anomaly can lead the MRT to discard a region from future consideration. Thus when the anomalous behavior returns, the MRT will have to restart its tracking process from the top level with coarse granularities, resulting in false negatives and wastage of measurement resources as well as increased

detection latencies. The two scenarios are shown in Figure 3, where the tracked flowset might be above or below the threshold radars depending on the particular iteration. Such an intermittent anomalous behavior is missed out by the MRT (a false-negative), as it keeps toggling between reset and zooming-in phases, which is also highlighted in the figure. The highly sensitive nature of MRT with respect to traffic variations can also be visualized in MRT exploration graph of Figure 4(a), where it could not get past three levels in exploration hierarchy, constantly resetting back to the root node.

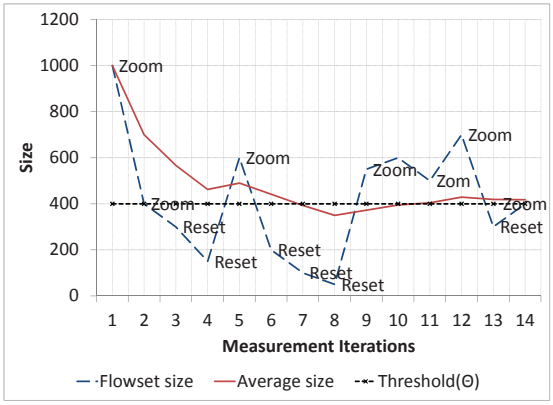


Fig. 3: Sensitivity of MRT algorithm to dynamic traffic fluctuations

The issue of false positives in the MRT can easily be addressed by continuously tracking a declared anomalous flow. However, solving the problem of false negatives due to MRT resets (whenever the current tracked size is below the threshold) is more involved. A naive solution to the problem could be to increase the measurement intervals, δ, at the cost of increased tracking latency. However, as the network traffic is usually bursty and time-varying, it is quite difficult to come up with an interval that can universally address the problem. Furthermore, the attacker can easily outsmart the solution by intentionally reducing the data rate to evade detection.

Another solution to the problem of frequent MRT resets could be to continuously track all the sub-regions. However, such strategy could lead to tracking a huge number of sub-regions, or rules, that may deplete the measurement resources and render the detection infeasible in real-time. A smart measurement solution therefore needs to prune away the irrelevant rules for better utilization of limited resources. Unfortunately, it is quite difficult to predict if a rule is relevant to the search query unless it has been answered. Thus the challenge here is in determining an optimized rule-set that can yield accurate answers with minimal latency while adhering to the resource budgets.

IV. Streaming Algorithms for Smart Guided-Measurements

We now present our streaming solutions that aim to address the challenges associated with application-aware rule-based online traffic measurements, with the goal of providing accurate response despite highly volatile traffic. As discussed earlier, a key design challenge is in maintaining computation and storage scalabilities, while offering high accuracies and minimum latencies in answering the user-query. In this context, we target volumetric anomalies, such as tracking of HHs and GIs, as our main query.

A. Flow Momentum

The Flow Momentum (FM) algorithm addresses the issue of MRT resets by taking into account the average bit-rate that is encountered over a hierarchical path reaching a flowset. This is in contrast to the original MRT, where the expansion/rejection decision is solely based on the flowset's current size. The FM algorithm thus effectively gives the leaf nodes a grace period in the active rule-set, to cope with the temporal variations in the anomaly. The durations of the grace period are proportional to the intensity, or momentum, of the anomalous flows that guided the measurements towards the leaf-node in the first place. Thus in the case of the HH, a leaf node may be more active if the anomalous flow has larger volume.

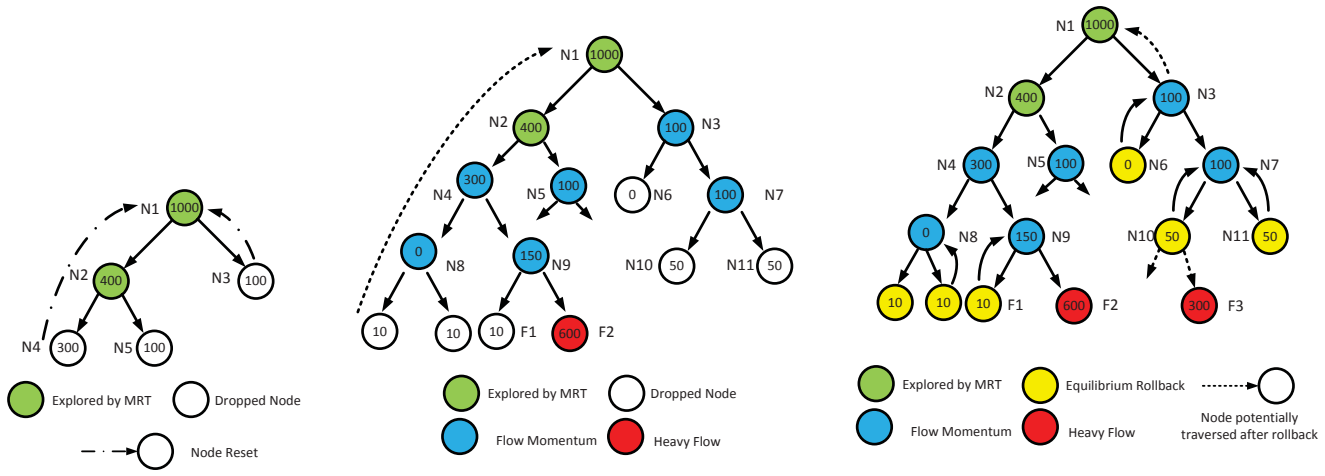
The pseudo code for the FM Algorithm is provided in Appendix-A, while a snapshot of the exploration graph for FM algorithm is shown in Figure 4(b). As expected, the FM is more conservative in making a flowset reset decision than MRT, making it more likely to reach an anomalous flowset, as is marked by explored nodes shown in red. The price paid is a potentially bigger active rule-set size corresponding to the higher number of tracked leaf-nodes. However, the worst case storage and computational complexities for FM are the same as that of MRT.

Proposition 1. *The computational and space complexity for Flow Momentum algorithm is* $\Theta[\Phi^{log_\Phi(n)} - 1/(\Phi - 1)]$ *and* $\theta(2^n)$ *per rule respectively.*

B. Equilibrium Rollback

The FM algorithm increases the duration where a given flowset remains covered by active rule-set, until its effective momentum also falls below the thresholds. In the scenario where the momentum goes below the thresholds, the FM also resets its tracking process from the top level, similar to MRT. The Equilibrium Rollback (ER) Algorithm addresses the problem of such MRT and FM resets through gracefully rolling-back, or zooming-out, of the expanded flowsets, instead of discarding off the flowsets. In doing so, the ER algorithm effectively tries to filter moderate traffic variations in the tracked flowsets, thereby achieving an an *equilibrium point* over the zoomed hierarchy that just passes the θ threshold requirements.

The pseudo code for ER is given in Appendix-A. A snapshot of the exploration graph for ER along with FM is presented in Figure 4(c). The ER algorithm gracefully degrades the zoomed granularities as the anomaly goes below the threshold, helping it to quickly re-expand the flowsets once the anomaly reappears. The cost of the algorithm is an increase in the storage requirements to keep the entire traversed hierarchy. It is to be noted that even though the entire hierarchy is stored, it is only the leaf nodes that are actively being evaluated, or constitute the active rule-set.



(a) MRT (b) Flow Momentum (c) Equilibrium Rollback

Fig. 4: Snapshots of Algorithmic Exploration Graphs with Threshold 400 bytes

Proposition 2. *The computational and storage complexity for Equilibrium Rollback is* $\Theta[\Phi^{log_\Phi(n)} - 1/(\Phi - 1)]$ *per rule.*

The computational complexity of ER (decision phase) is the same as that of MRT, assuming the consistency of the tracked flowsets/flows as in the case for MRT. However, whereas the storage complexity of MRT is $\Theta(2^n)$ per rule in maintaining just the leaf nodes, the ER algorithm's storage complexity corresponds to the storage requirements of the entire hierarchy for the worst case expansion.

C. Directed Momentum

The streaming algorithms discussed so far are quite generic in nature, that is, they do not take into account any opportunity or constraints presented by application or available computational platform. They are thus best suited for scenarios where the knowledge of the anomaly or the environment is limited. However, such a separation between the application/platform and the algorithm may lead to sub-optimal use of the computational resources. A very large rule-set can throttle the system by consuming scarce resources to process redundant or unnecessary rules. An intelligent hacker could actually use this deficiency to outsmart the detection process in real-time by injecting a huge number of false flowsets (or leads) to be tracked. A smart algorithm therefore needs a mechanism to filter out irrelevant leads from the active rule-set.

As discussed earlier, it is quite difficult to predict how relevant a given rule is to the user-query unless it has been evaluated. However, the knowledge of an anomaly can help to intelligently quantify the rules using their past behavior. We make use of the anomaly information in designing a smart Directed Momentum (DM) algorithm that *directs* the search process by associating the limited resources where they are deemed the most useful. We develop the algorithm in the context of elephant flows. However, the ideas behind the algorithm are applicable for other types of volumetric anomalies, such as global iceberg.

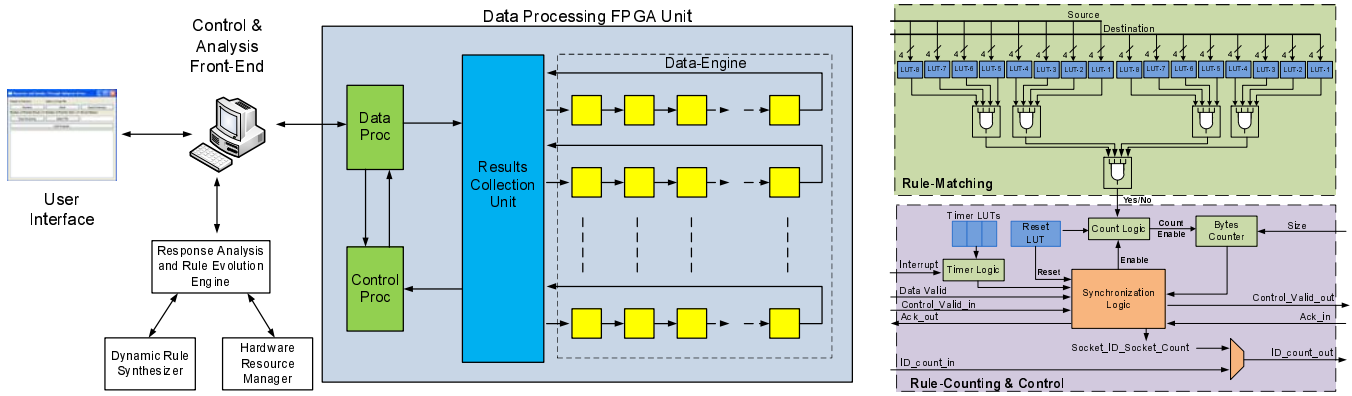
A characteristic feature of elephant flows are their higher longevities. In the context of an iterative search process such as the Flow Momentum, the long lasting property of the elephant flows translates into higher expansion of the corresponding flowsets. We utilize this property in harnessing the *Momentum* to be *directed* towards the anomalous elephant flows by giving preference to the rules that have higher granularities. The Directed Momentum algorithm thus formed is tabulated in Algorithm-1.

Directed Momentum works by scaling the individual flowset longevities using a measure referred to as *Stretch*. The Stretch takes into account the availability of computational resources and could either be positive or negative. A positive Stretch describes a scenario where the active rule-set is smaller than the available rule processing resources. In contrast, a negative Stretch implies exceeding the computational budget by the active rule-set. The DM algorithm uses the negative-Stretch to compute a measure called *pull*, or algorithmic effort; such that the higher the pull, the higher the algorithmic effort is in reducing the active rule-set. The algorithmic *pull* is combined with a rule's granularity to form a measure referred to as *pForce*, as shown in the pseudo code.

Proposition 3. *The computational and space complexity for Directed Momentum algorithm is* $\Theta[\Phi^{log_\Phi(n)} - 1/(\Phi - 1)]$ *and* $\theta(2^n)$ *per rule respectively.*

V. Measurement Platforms

The streaming algorithms discussed in previous sections are composed of two parts: (a) a data-plane to match incoming packets with the rule-set along with (b) a control-plane for algorithmic decisions to process rules that lead to measurements of finer granularities. These two planes have been previously mapped on software [6], hardware [5] and a software-hardware co-designed solutions [20]. In this paper, we implement and evaluate our three streaming algorithms on both hardware and software platforms, as described in the following sub-sections.



(a) Platform Overview (b) Dynamically Reconfigurable Socket

Fig. 5: BURAQ Measurement Platform [9]

Algorithm 1: Directed Momentum

input : R: Active rule-set
input : $Q\{\}$: Set of Rule Processors
input : Φ: The expansion/zoom ratio
input : δ: Measurement Interval
output: $E_f\{\}$: set of elephant-flows
$Stretch \leftarrow |R| - |Q|$
$Pull \leftarrow |\min(Stretch, 0)|$

/* Measurement Phase */
while $t \leq \delta$ **do**
  for $R_i \in R$ **do**
    if $R_i = P_t$ **then**
      $R_i.Size \leftarrow R_i.Size + P_t.Size$
      $R_i.M \leftarrow R_i.M + P_t.Size$

/* Decision Phase */
for $R_i \in R$ **do**
  if $(R_i.Size > Size_{Th})$ **then**
    if $\texttt{Granularity}(R_i) = MAX$ **then**
      $E_f \leftarrow E_f + R_i$
    else
      R.replace $\{R_i$, Expand $(R_i, \Phi)\}$
      $R_{expanded}.M \leftarrow R_{parent}.M$
      $R_{expanded}.Static \leftarrow 0$
  else if $\texttt{pForce}(R_i, Pull)/\lambda \geq \theta$ **then**
    $R_i.Static \leftarrow R_i.Static + 1$
    R_i.Hold
  else R_i.Drop

/* Calculates Directed Momentum */
procedure pFORCE(R_i, $Pull$)
  $\lambda_i \leftarrow (R_i.M/k * \delta)$
  $d_i^n \leftarrow \exp^{(Pull * R_i.Static/\texttt{Granularity}(R_i))}$ /* pForce */
  return (λ_i/d_i^n)
end procedure

A. BURAQ Hardware Mapped Rule-Processing System

For hardware platform, we adopt the closed-loop BURAQ measurement and analysis framework [9], which combines the speed of a customized FPGA based rule-processing engine with the flexibility of a software based controller, as shown in Figure 5(a). By combining the speed with flexibility, the BURAQ represents an online measurement framework that can measure and analyze streaming traffic in real-time. We present an overview of the system and discuss the various constraints and challenges pertaining to online traffic measurement and analysis.

1) Rule Processing Data-Plane: BURAQ's data-plane is a custom architecture on an FPGA unit that combines a rule processing unit, the *Data-Engine*, with associated synchronization and data transfer logic. The Data-Engine itself is composed of a number of parallel rule processing units, dubbed as *sockets*, that are arranged similar to a systolic array as shown in Figure 5(a). The sockets are programmable rule matching units that are optimized to match the programmed rules and count the size of streaming network packets, independently and concurrently, in real-time. However, unlike the systolic arrays, the sockets only pass their results on to the next socket in a chain. These results from parallel chains in the Data-Engine are collected by the associated logic and communicated to higher control and analysis layers for further processing.

A high level description of a socket that can process two tuple $\{sip, dip\}$ rule is shown in Figure 5(b). The socket is composed of an array of Look-up-Tables (LUTs) that map the rules in the form of Boolean bit-vectors. As new rules get generated on-the-fly during algorithmic iterations, the LUTs are reprogrammed with updated Boolean bit-vectors corresponding to the new rules. One of the core features of the BURAQ framework is its novel use of fine grained Partial Dynamic Reconfiguration (PDR) of FPGA fabric in (re)programming the LUTs. The PDR programming paradigm does away with traditional just-in-time compilation of FPGA configuration data, a highly latency intensive operation. Instead, the socket (re)programming is based on minute logic changes involving specific LUTs whose entries are dynamically and directly updated in the FPGA's configuration memory. The dynamic nature of the PDR implies that only the operation of the LUT being (re)programmed is effected while the rest of the design operates as usual. We refer the interested reader to [9] for the details of the socket and its (re)programming paradigm.

2) Control & Analysis Front-End: The BURAQ's Control and Analysis Front-End is where user programs the high-level

formulation of the measurement requirements. It incorporates a *Dynamic Rule Synthesizer* that translates the user requirements into socket deployable Boolean bit-vectors. The synthesizer also works in closed-loop with the measurements reported by the data-plane and a *Response-Evaluation Engine* in automating the exploration of the vast search space. It is the response-evaluation engine where the discussed streaming algorithms provide the streaming automation. The response evaluation engine analyzes the intermediate results from the back-end in assisting the dynamic synthesis of intermediate rules on-the-fly. The controller also maps the synthesized rules at the sockets using *Hardware Resource Manager* that performs a resource aware rule deployment at the data-plane.

B. Software Mapped Rule-Processing

A software based solution has more flexibility in maintaining and processing complex data-structures, trading computation with increased memory use. Traditionally, the software mapped rule-processing is done by making efficient use of a tree based data-structure, the radix trie [21]. The radix-trie works by mapping a given rule-set on a tree structure, whose maximum depth corresponds to the worst case number of required match operations. For instance, the 64-bit two tuple $\{source, destination\}$ pair could be mapped onto a tree with maximum depth of 64 nodes, requiring at most 64 comparisons with an incoming packet to declare a match (or a mis-match). This worst case represents significant improvements from naive sequential rule-set matching where an incoming packet may need to be tried with the complete rule-set, leading to the worst case number of match operations equal to the size of the rule-set. The price paid is a slight increase in rule-set mapping latency, as the graph structure corresponding to a given rule-set has to be populated before being available for matching operations.

C. Practical Constraints

Any rule-processing solution is faced with a number of computational and communication constraints. A core constraint is the availability of computational resources for rule-processing. As rule-set grows, digging deeper in the vast n-tuple search space, it puts additional strain on the limited computational resources. The standard practice for a hardware is to *pipeline*, or *roll-over*, the additional rules, over the limited resources in multiple steps. Such a rule-pipelining increases the overall latency in yielding the final answer. For instance, in a platform employing N parallel rule-processing units, it takes $ceil(N/|R|)$ measurement cycles (each having measurement interval δ) for processing a rule-set of size $|R|$ in any given algorithmic iteration.

The active rule-sets may also not be an integer multiple of N, and therefore there may exist unused rule-processing resources in certain algorithmic iterations. In practice, the controller can maximize the hardware resource utilization by simultaneously mapping new rules for successive measurement along with portions of previous rule-sets. However, it makes the design and analysis of an algorithm quite cumbersome. For simplicity, we therefore detach the rule-sets in discrete measurement cycles by assuming the controller only adapts new rules once a given rule-set is completely processed. We refer to such a controller implementation as *blocking*.

The synthesis of rules, deployment and collection of their results from the computing platform involves finite latencies, during which streaming packets may miss observation. We aggregate the above latencies together as reprogramming latency, denoted by ϵ. The BURAQ platform uses PDR to maximize the FPGA utilization in increasing the number of rule-processing sockets. A higher rule-processing opportunity reduces the pipelining effects, however the downside of PDR is a slight increase in reprogramming latencies as compared to a static solution [20]. Similarly, depending on the rule-set size, the reprogramming latency of software mapped solution could be significant as the trie structure needs to be populated. Both rule pipelining and reprogramming latencies effect accuracy of reported results. We discuss accuracy measures for the Flow and Directed Momentum Algorithms with a more detailed analytical analysis in our technical report [10].

Theorem 1. (Measurement Accuracy) *If d_i^n represents pForce in an algorithmic iteration j, then the Measurement Accuracy is given by $\delta/(\sum_{j=1}^{n} \left\lceil \frac{N}{|R^j|} \right\rceil (\delta + \epsilon).d_i^n)$, where $d_i^j = 1$ for Flow Momentum and $|R^j|$ represents the size of rule-set in algorithmic iteration j. ($N = R^j$ for software-mapped rule-processing)*

Proof:

For the measurement to be accurate, the observed measurements should match with ideal measurements. If $E[\alpha_i^j]$ and $E[\beta_i^j]$ represent expected values for observed and ideal measurements in an algorithmic iteration j for Flow or Directed Momentum algorithms, then for the system to be accurate

$$Measurement\ Accuracy = E[\alpha_i^j]/E[\beta_i^j] \leq 1$$

Equating the expressions for the expected values as described in our technical report [10] leads to the desired result. ■

Corollary 1. *The Measurement Accuracy is less than 1 in a blocking measurement system.*

In a blocking implementation, the reprogramming (ϵ), and observation (δ) latencies do not overlap in time. This is to say that the controller only adapts new rules at fixed time durations, rather than reading and reprogramming the sockets at different times to spread the load. In other words, the measurement and reprogramming phases are sequentially chained in blocking implementation, thereby leading to the above observation.

The system's accuracy in detecting streaming anomalies is not only a function of platform's measurement inaccuracies, but also to the algorithmic aspects. The algorithmic inaccuracies occur due to various abstractions that limit the complexity of the problem. For instance, the finite measurement intervals of the iterative paradigm, though helpful in curtailing the complexity of the measurement framework, also leads to limitations in traffic observation. Such limitations raise

possibilities of false positives and false negatives during the anomaly detection process. We discuss these possibilities in more detail and provide mathematical bounds to the presence of false alarms for heavy flow identification in our technical report [10]. The bounds provide tuning knobs for a user to fine-tune the system's accuracy and latency for any given platform and network conditions.

D. Cross-cutting Issues

Theorem-1 may appear to suggest that the $pForce$ has a strictly inverse relationship with the system accuracy, leading to a conclusion that FM algorithm must always have superior accuracy than the DM algorithm. However, $pForce$ also influences the rule-set size, that has its own influence on the system accuracy. As stated earlier, a rule-set that exceeds the available rule-processing resources on hardware will have to be rolled over the platform's limited resources in successive measurement intervals. The rolling process leads to *measurement disentanglement* between the parent and the children rule-sets. Such a disentanglement reduces system accuracy and is represented by the factor $\left\lceil \frac{N}{|R|} \right\rceil$ in the measurement accuracy equation. Increasing $pForce$ reduces the effects of the measurement disentanglement by reducing the rule-set sizes, and as such can positively influence the system accuracy. However, a very high $pForce$ can also be counter-productive, as the factor $\left\lceil \frac{N}{|R|} \right\rceil$ is bounded by the minimum value of 1. A balanced $pForce$ is therefore essential in maximally utilizing system resources. We will discuss the issue more when we present the results in the next section.

The preceding discussion may lead one to argue that software based iterative solutions may offer superior accuracies to the hardware counterparts as they do not have to face the blocking and pipelining effects. However, software solutions face two additional overheads: (1) increased latencies in forwarding packets from a streaming network to the point of measurement (in cases where measurement framework is remotely located), and (2) latencies involved in moving packets up the software stack where rule-processing takes place. The exact latencies are a function of network layout, CPU speed, operating system, and software stack. Furthermore, mirroring the packets for remote processing could additionally consume network bandwidth, and may well interfere with the actual network traffic, leading to measurement inaccuracies. Such additional overheads may well limit the feasibility of software based iterative framework in coming up with accurate and/or quick answers to user-queries.

VI. Empirical Evaluation

In this section, we present our empirical evaluation of the proposed algorithms utilizing BURAQ hardware and Radix-Trie software based rule-processing platforms. We begin by discussing the rule-set sizes created by the algorithms that have a direct implication on running costs of the algorithms. We then map the algorithms on BURAQ platform and demonstrate their accuracies in its context. This is followed by a discussion on the effects of tuning Zoom-Ratio parameter towards the accuracy and detection latencies of the solutions. We next evaluate the algorithms on software platform and analyze the latencies that contribute in both hardware and software mapped solutions. Finally, we apply our algorithms to a distributed measurement framework and demonstrate its effectiveness in isolating distributed anomalies.

The experiments are performed using a PC based workstation on Intel Core i7 Q740 Quad-core processor running at 1.73-GHz and having 4 GB memory. The BURAQ's rule-processing engine is mapped on a Xilinx Virtex-II Pro FPGA, XCV2VP30, running at 100-MHz, and employing $N = 169$ parallel sockets. The rule composition and results analysis is performed at the PC based controller, that is connected with the processing-engine over Ethernet. The complete system setup is shown in Figure 5(a). Unless stated otherwise, the experiment are based on default values for threshold $\theta = 1\%$, Zoom Ratio $\Phi = 2$, and measurement interval $\delta = 1$s.

We evaluate the proposed algorithms by injecting varying degrees of high volumetric flows in CAIDA Backscatter data traces [22]. The injected 10 heavy flows contribute from 0.5% to 1.4% of the total traffic in the traces, such that their intensity at various snapshots mirrors the random traffic activity of the trace data. The piggybacking of trace data variations imply that the inserted flows inherit the real network traffic variations. We used heavy-hitter threshold value θ to be 1%. As such, the inserted flows split evenly between true heavy-hitters (that algorithms need to isolate/announce) and heavy flows (not to be isolated/announced) around the threshold value, thereby producing stress test cases for evaluating the algorithms.

We define a parameter *Score* to quantify the progress of the algorithms in identifying the inserted flows. Mathematically,

$$Score = \frac{\sum max|R_i|}{\sum |f_i|}$$

where $|R_i|$ represents the size, or expansion granularity, of a rule, R_i, in an algorithmic iteration that matches an inserted flow f_i of size $|f_i|$ bits. In cases where multiple rules match an inserted flow, we use longest prefix matching rule or the rule with highest number of matching bits with the inserted flow in the calculations. The parameter thus represents the degree by which an algorithm has either correctly identified the inserted heavy-hitters (referred to as *true-score*), or incorrectly identified the inserted heavy-flows below threshold as heavy-hitters (referred to as *false-score*), with the maximum value of 1 implying all the flows being completely identified and announced.

A. Rule-Set Size Comparison

We start with exploring the different rule-set sizes generated by the discussed algorithms, that reflects the rule-processing costs associated with them. The variations of rule-set sizes with algorithmic iterations is presented in Figure 6. We also combine presented algorithms in various combinations as shown in the figure. The results show that MRT rule-sets suffer from frequent resets, owing to its sensitivity to traffic variations. In contrast, the rule-set size for ER algorithm is fluctuating around a steady state value, corresponding to the

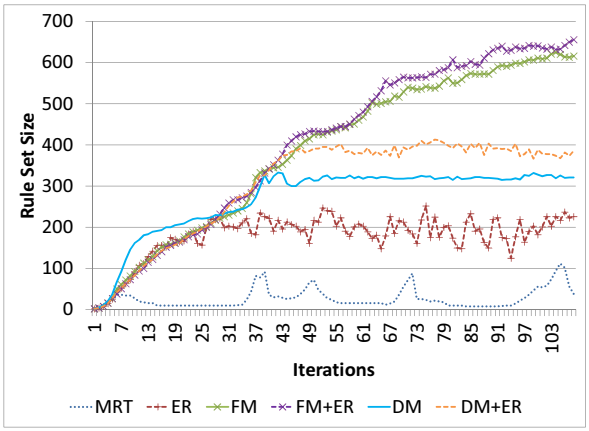


Fig. 6: Rule Set Size with algorithmic iterations

toggling between the expansion and rollback phases. The four variations of Momentum type algorithms, however, vary in the rule-set costs, with FM and FM+ER yielding steady increase in rule-set sizes. As noted earlier, a larger rule-set size may adversely effect an algorithm's convergence time in reaching to a decision. The issue is addressed by the DM algorithm, which maintains a steady rule-set size. A combination of DM with ER (DM+ER) also shows similar steady state rule-size convergence, albeit with a larger bias. This is because the rules that were being discarded by DM are now being caught in the Rollback catch-net.

B. Streaming Algorithms Analysis on BURAQ

The rule-set size variations provide an estimate on the running costs of the algorithms, though their actual effectiveness depends on the latencies and degrees of accuracies they offer. We discuss the algorithms' effectiveness in terms of progression of True/False Scores with respect to clock time. The algorithmic latency is a function of algorithmic iterations, that in turn is derived from rule-set sizes and available rule-processing resources. As each measurement cycle on BURAQ takes constant amount of time ($(\delta + \epsilon) \approx 1.71$ seconds) across the algorithms, a difference in latency implies effects of pipelining of large rule-sets on limited set of resources.

The progression of True-Score (TS) for the presented algorithms is shown in Figure 7(a). It can be seen from the figure that the MRT algorithm performs quite poorly, resetting periodically and unable to isolate/announce any of the inserted flows. In contrast, the ER algorithm avoids such resets by rolling an expanded rule back to its parent granularity if it falls below the threshold, thereby showing TS improvement in the results. However, the bursty nature of the traffic means that the algorithm keeps fluctuating between the rollback and expansion phases, thus settling with a steady state score below 1, and also being unable to announce any of the inserted flows. The difficulties in isolating the anomalous flows are addressed by the Momentum type algorithms that only saturate with the unit-score, that is, after identification and announcement of all the induced heavy-hitter flows. In our earlier work [7], we showed that the Momentum type algorithms have only slight differences with respect to algorithmic iterations in achieving a unity score. However when time is taken in account, the DM algorithm shows its superiority over FM by reaching the unity scores earlier as shown in the figure. This confirms to our contention that DM algorithm works intelligently to prune out the irrelevant rules, thereby focusing or *directing* the search towards the user defined goals.

The False-Score (FS) progression for the various algorithms is shown in Figure 7(b). A FS value shows that the algorithms have narrowed down to certain rules that match the induced false heavy-hitters with varying degrees. However, the FS value does not mean that some of the induced flows have been completely identified/announced. This implies that although we have a False-Score, the algorithm did not yield a false-positive. Indeed, in our experiments, none of the heavy flows below threshold ever got detected. We note that the FS generally follows the variations in rule-set sizes. Such variations further highlight the fact that the FS is a by-product of rules that have randomly matching granularities with the induced heavy flows, rather than an actual algorithmic progression towards them. It is also to be noted that although we do not empirically have a false-positive, there is still a small but finite probability of false-positive.

The 10 inserted flows provide a distribution of true/false positives and negatives across the algorithms in the above experiments. These values are tabulated in Table-I, and combined using the standard F1 measure to produce an aggregate score for the algorithms. As can be noted, the proposed DM and FM algorithms offer marked 100% accuracy improvement over conventional MRT.

Algorithm	t_p	t_n	f_p	f_n	F1 Score
MRT	0	5	0	5	0
ER	0	5	0	5	0
FM	5	5	0	0	1
DM	5	5	0	0	1

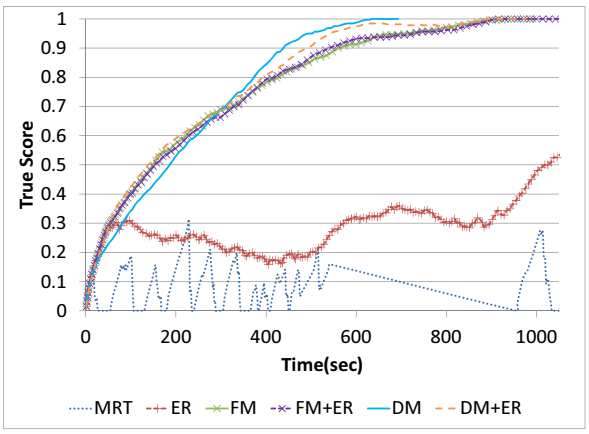
TABLE I: Accuracy Comparison

The results yield an interesting observation that in a practical hardware rule-mapped system, the efficiency of an anomalous flow detection may not necessarily be improved by increasing the rule-set size. In contrast, an intelligent search directed towards the most reliable leads can produce equivalent accuracy while reducing the detection latency. The notion of 'intelligence' is of course application dependent. The results demonstrate the significant gains that can be achieved by incorporating such application knowledge.

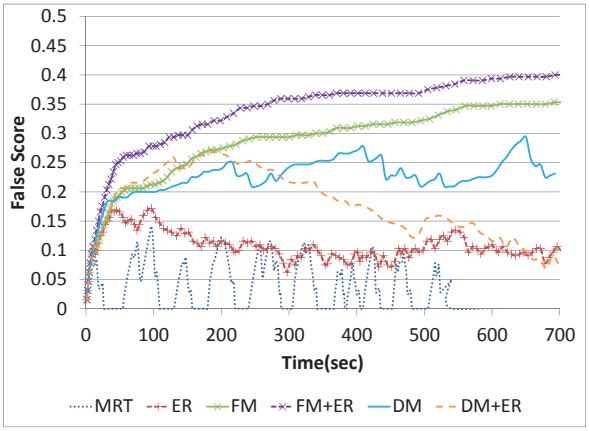
C. Zoom Ratio

The Zoom (or Expansion) Ratio (ZR), has conventionally been associated with the speed of parsing through the search space. As discussed in Section-II-A, a higher ZR conceptually translates into quicker drilling down to finer granularities, and therefore should accelerate the isolation of an anomaly. We hereby investigate such a relation of the ZR using the realistic BURAQ measurement system.

Figure 8 shows the latencies in isolating the five anomalous heavy flows with increasing ZRs using the DM algorithm. The results, to our surprise, indicate that for a practical



(a) True Score



(b) False Score

Fig. 7: Score progression with time on BURAQ

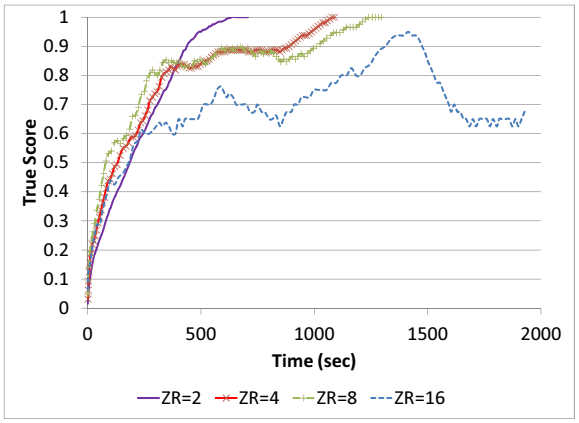


Fig. 8: Variation of Zoom Ratio (ZR)

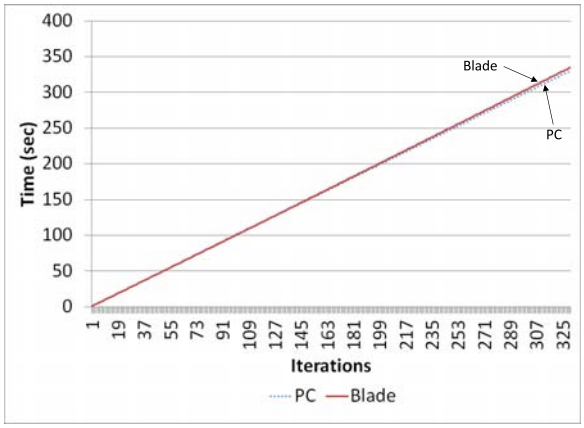


Fig. 9: Software Mapping on PC and Blade Server

measurement system, the ZR actually demonstrates an inverse relationship with the detection latencies[3]. A closer inspection reveals some interesting characteristics of the streaming algorithms. We note that the higher ZRs translate into an increased zoom granularity of the tracked rules (sub-regions). However, generally a network is mostly anomaly free and as such, the increased amounts of inspections with the higher ZRs translate into tracking sub-regions that do not contribute to isolation of the anomalies. In practical terms, the above implies allocation of resources to non-interesting rules/regions, thereby wasting of resources. Thus, although the higher ZRs reduce the number of iterations (not shown), in practice such a reduction in the algorithmic length may not translate into improved latencies in isolating anomalies.

The results in Figure 8 reveal yet another interesting observation: that with higher ZRs, the DM exhibits difficulties preserving continuous true-score progression. This can be observed in the brief plateaus and dips in its TS. This is because of the way DM aggressively tries to offset increase in the rule-set sizes at higher ZRs. As a result, some relevant rules may end up being dropped from active rule-set. Such a scenario leads to a loss in *momentum*, or TS, of the algorithm. The issue can be quickly resolved by adjustments in the algorithmic Stretch parameter, giving increased leniency with increasing ZRs, at the cost of larger rule-set sizes.

[3]We observe similar inverse relationship using other algorithms as well

D. Algorithm Analysis for Software Mapped Rule-Processing

Software defined networking paradigm and the adoption of OpenFlow enable a central controller to reconfigure the packet forwarding/processing tasks in routers/switches. Previous work have demonstrated case studies where OpenFlow can be leveraged to control how packets are forwarded, and well as how traffic measurements [5], [23] are done. With OpenFlow-like capability, our streaming algorithms can easily be implemented in software (as opposed to the hardware counterpart), and as such, we are motivated to characterize its performance.

For evaluation purposes, we employ radix-trie based rule-processing over two platforms: the previously discussed PC platform as well as an HP Blade server Open Networking Environment [24]. The HP Blade comprises of Intel Core 2 Duo $T7500$ processor, running at 1.2 GHz and having 4 GB of memory. To isolate the performance of the algorithms on the software based rule-processing from issues arising from rule-processing at higher application layers (as discussed in

Section-V), we emulate the network at the application layer using the same data as before, and insert the same number of heavy-hitters and heavy flows.

We begin by comparing the relative performance of the two software platforms. The comparison employing FM algorithm is shown in Figure 9. We observed similar identical behaviors for other algorithms, indicating their neutrality over the choice of platform.

We next compare the performance of the two top performing algorithms, the FM and DM along with the ER enhancement, on software framework. The performance of the algorithms in terms of their true and false scores in isolating the injected flows is shown in Figure 10. Compared to 600 seconds detection time of BURAQ platform, the software mapped algorithms were able to completely identify all the heavy-hitters in less than 200 seconds. There are two core reasons behind the difference:

- The software based rule-mapping is independent of blocking and pipelining effects as faced by hardware (since $N = R^j$ for software).
- The software simulations are based on data-traces running at the same algorithmic application layer ($\epsilon \approx 0$).

The latter point emphasizes the nature of our simulations, that do not employ remote data collection or its movement up the software stack. As discussed in Section-V, such latencies are deployment and platform specific. The results thus imply the expected performance of our algorithms in an ideal deployment having zero communication overheads. We shall shortly discuss these latencies to understand their impact on the overall systems' performance.

The question of which platform to use is tied to the needs of the application, and is tangential to the performance of the proposed algorithms. However, the results emphasize that in situations where the rule-set size is a relaxed constraint, such as the software or a hardware platform with vast number of rule-matching resources, the combination of FM+ER is more effective than the DM+ER combination. This is because the significant overheads of rule-set pipelining can be avoided. However in situations where measurements take place in real-time, the use of hardware such as BURAQ becomes a necessity and one may have to constrain the growth of rule-set size with smart algorithms such as DM+ER.

E. Latency Analysis

We next discuss the various latencies involved in the iterative measurement frameworks. The latencies not only play a significant role in deciding the aggregate delay in getting response from the measurement framework, but as shown in Theorem-1, the latencies directly affect the accuracy of the response. A better understanding of the latencies is thus essential in making informed decision about the trade-offs involved in accuracy and delays in response.

The various latencies involved in the two measurement frameworks discussed in this work are tabulated in Table-II. The values present the latencies associated with running the better performing DM + ER for hardware, and FM + ER for software based solutions, over 212 algorithmic iterations, while employing default parametric values.

It can be seen that the major chunk in the latency budget for BURAQ is the communication latency, which is an aggregation of the delays associated with transferring the sockets' statistics and (re)programming bit-vectors across the Ethernet. The statistics collection is the second major component of the latency. The third major component, the rule deployment latency, is associated with delays involved in (re)programming of the sockets for deployment of new rules. Finally, the actual delay involving the rules' evaluation and synthesis at the algorithmic layers is almost negligible.

The above latencies can be sub-divided into algorithmic and platform-specific latencies. In particular, the rule synthesis and statistics collection are functions of the algorithm whereas the other two are by-products of the platform and deployment. Among the algorithmic latencies that are of interest to us, we can isolate the rule synthesis latency as being the base cost of our streaming algorithms. The base cost relates the discussed algorithmic complexities in empirical figures. Note that the base cost of our streaming solutions is quite minimal. In practical terms, this implies that there will be fewer packets that will miss inspection during rule evaluation and synthesis, thereby increasing the confidence in reported results. However, that the base cost for software mapped solution is magnitudes higher than that for hardware. Such a difference is expected, as the software has much higher overheads in populating complex graph based structures for rule-processing.

As highlighted earlier, the rule deployment and communication overheads are functions of deployment. These overheads along with algorithmic base costs yield the aggregate latency overheads for the measurement framework, ϵ. Thus depending on the deployment and aggregate latency overheads, one needs to select the value for measurement interval to satisfy the accuracy goals, while keeping the total response delay within the budgets.

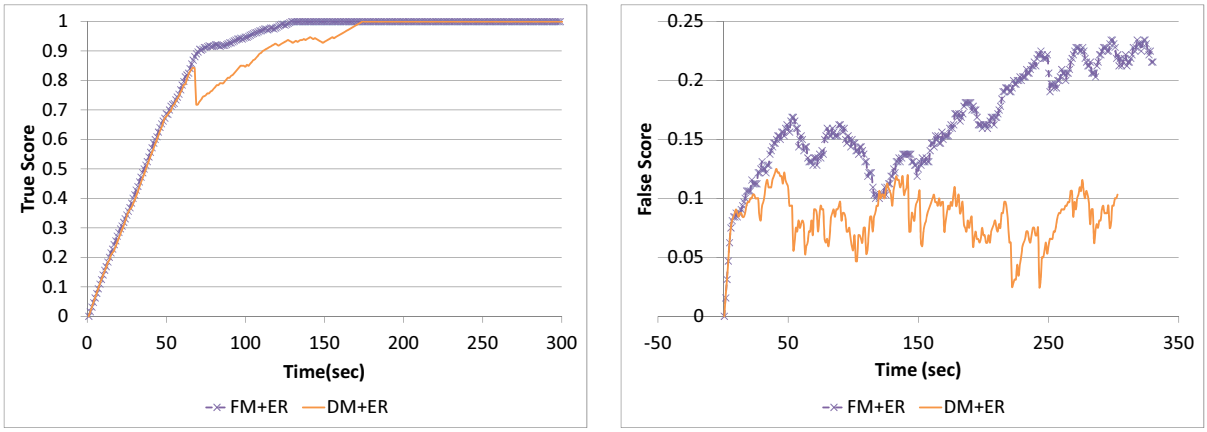
	Rule Synthesis	Rule Deployment	Statistics Collection	Communication
BURAQ	0.0004	101	212	363
Software	0.6	NA	212	NA

TABLE II: Latency Comparison (seconds)

F. Distributed Measurements

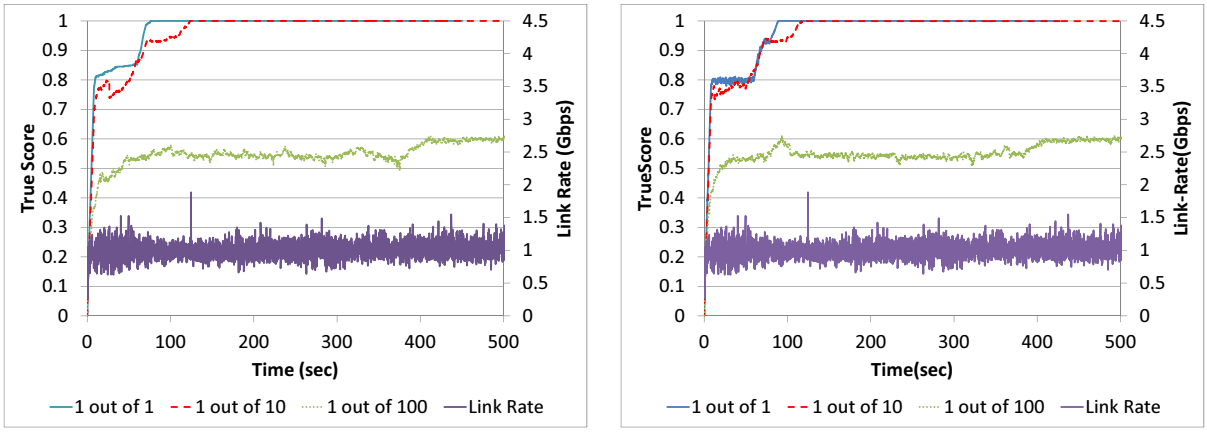
Lastly, we evaluate the effectiveness of our algorithms in a distributed framework to isolate global icebergs and utilize the distributed setup to evaluate the effects of sampling on the performance of the algorithms. For this purpose, we used data from Abilene network [25]. The data was collected over 11 distributed sites in 2007 and contains 8 distributed icebergs that lie above 1% threshold value. However, for the purposes of stress testing, we insert 10 global icebergs, 5 of them just above the threshold (true-icebergs) while the remaining being just below the threshold (false-icebergs).

We emulate the distributed network in software by running parallel threads, corresponding to 11 distributed network sites.



(a) True Score (b) False Score

Fig. 10: Score Progression for software-mapped solution



(a) Using Flow Momentum Algorithm (b) Using Directed Momentum Algorithm

Fig. 11: True score progression for Distributed Measurements

Each of the distributed site contains an independent rule-matching unit that collects the programmed statistics pertaining to its network site. The rule-matching units are managed by a central map/reduce software layer, that handles the mapping of the measurement requirements coming from the higher algorithmic layer. Furthermore, it also aggregates (reduce) the collected statistics from the rule-matching units for further processing by the algorithmic layer. The algorithmic layer is where our proposed algorithms process the results and make their decisions about future refined measurements in the distributed network.

The True-Score results for FM + ER and DM + ER algorithms are shown in Figure 11(a) and Figure 11(b), respectively. As expected, the accuracy of the algorithms decreases with decrease in sampling rate. However, the algorithms demonstrate robust performance despite sampling, achieving more than 50% of the True-Score even with a low sampling rate of 1 out of every 100 packets. The False-Score with the low sampling rate, however, led the DM+ER to announce one false iceberg as shown in Figure 12. Though not presented, the False-score for FM+ER algorithm has only been slightly less than DM+ER.

We finally present a comparison of rule-set sizes for the proposed algorithms in the distributed setting as shown in Figure 13. As expected, the DM + ER algorithm was able to have a more stable rule-set size as compared to FM + ER algorithm. The question on which algorithm to utilize for distributed measurements is again dependent on the choice of platform, with DM + ER a more suitable candidate for fast but limited hardware based rule-matching engines such as BURAQ and FM + ER being a better choice for more latency relaxed software environments.

VII. Conclusion

Iterative measurements offer streaming measurements and analysis, and help overcome the offline nature of traditional measurements. However, the effectiveness of iterative measurements are heavily tied with smart algorithms that can efficiently make use of constraints as imposed by the measurement platforms. The state-of-the-art Multi-Resolution Tiling (MRT) algorithm is too sensitive to specific traffic patterns, and is blind to any constraints or opportunities arising from measurement platforms. We address the issues with three iterative streaming algorithms that cater to varying degrees

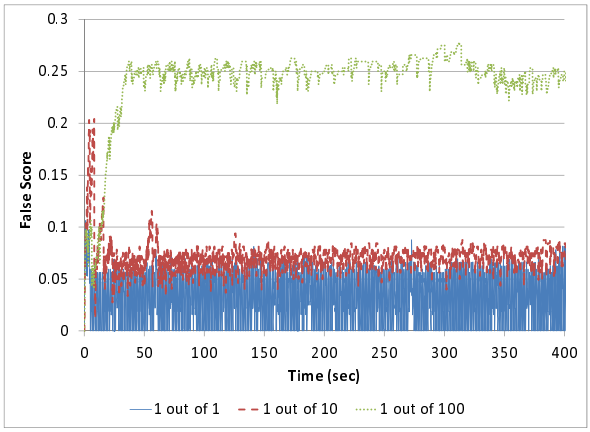


Fig. 12: False Score progression for DM+ER algorithm

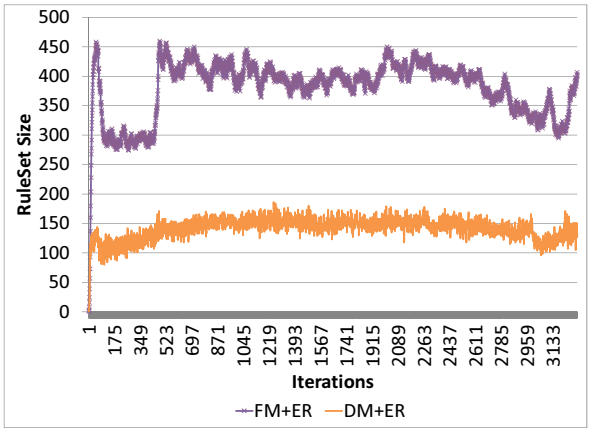


Fig. 13: Rule-set size comparison

of computational and storage budgets, detection latency, and accuracies of query response. We evaluate the streaming algorithms on diverse hardware and software based measurement platforms, for local as well as distributed measurement settings. The results demonstrate a marked 100% improvement in detection accuracy compared to MRT, with a moderate increase in storage and computational complexities.

REFERENCES

[1] "Cisco NetFlow," http://www.cisco.com/warp/public/732/Tech/netflow.

[2] J. Mai, C.-N. Chuah, A. Sridharan, T. Ye, and H. Zang, "Is sampled data sufficient for anomaly detection?" in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, ser. IMC '06. New York, NY, USA: ACM, 2006, pp. 165–176. [Online]. Available: http://doi.acm.org/10.1145/1177080.1177102

[3] A. Ramachandran, S. Seetharaman, and N. Feamster, "Fast monitoring for traffic subpopulations," in *IMC '08: Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, 2008, pp. 257–270.

[4] C. Estan, S. Savage, and G. Varghese, "Automatically inferring patterns of resource consumption in network traffic," in *SIGCOMM*, 2003.

[5] L. Jose, M. Yu, and J. Rexford, "Online measurement of large traffic aggregates on commodity switches," in *Hot-ICE'11*, 2011, pp. 13–13.

[6] L. Yuan, C.-N. Chuah, and P. Mohapatra, "ProgME: towards programmable network measurement," in *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, 2007, pp. 97–108.

[7] F. Khan, N. Hosein, C.-N. Chuah, and S. Ghiasi, "Streaming solutions for fine-grained network traffic measurements and analysis," in *Proceedings of the Seventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ser. ANCS '11, 2011.

[8] 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, no. 2, pp. 69–74, Mar. 2008.

[9] F. Khan, S. Ghiasi, and C.-N. Chuah, "A dynamically reconfigurable system for closed-loop measurements of network traffic," *IEEE Transactions on Computers*, vol. 99, p. 1, 2012.

[10] F. Khan, N. Hosein, S. Ghiasi, C.-N. Chuah, and P. Sharma, "Streaming solutions for fine-grained network traffic measurements and analysis," UC-Davis, Tech. Rep. ECE-CE-2013-2, UC-Davis, 2013. [Online]. Available: http://www.ece.ucdavis.edu/cerl/techreports/2013-2/

[11] N. Brownlee, C. Mills, and G. Ruth, "Traffic Flow Measurement: Architecture," RFC 2722, 1999, http://www.ietf.org/rfc/rfc2722.txt.

[12] N. G. Duffield, "Sampling for passive internet measurement: A review," *Statistical Science*, vol. 19, no. 3, 2004.

[13] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, and D. G. Andersen, "cSamp: A system for network-wide flow monitoring," in *Proc. 5th USENIX NSDI*, San Francisco, CA, Apr. 2008.

[14] C. Estan and G. Varghese, "New Directions in Traffic Measurement and Accounting: Focusing on the elephants, ignoring the mice," *ACM Transactions on Computer Systems*, vol. 21, no. 3, pp. 270–313, 2003.

[15] G. Cormode and S. Muthukrishnan, "An improved data stream summary: the count-min sketch and its applications," *J. Algorithms*, vol. 55, pp. 58–75, April 2005. [Online]. Available: http://portal.acm.org/citation.cfm?id=1073713.1073718

[16] Q. G. Zhao, M. Ogihara, H. Wang, and J. J. Xu, "Finding global icebergs over distributed data sets," in *Proceedings of the 25th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 2006.

[17] H. Zhao, A. Lall, M. Ogihara, and J. Xu, "Global iceberg detection over distributed data streams," in *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, march 2010.

[18] G. Huang, A. Lall, C.-N. Chuah, and J. Xu, "Uncovering global icebergs in distributed streams: Results and implications," *Journal of Network & Systems Management*, vol. 19, 2011. [Online]. Available: http://dx.doi.org/10.1007/s10922-010-9186-5

[19] J. Jung, B. Krishnamurthy, and M. Rabinovich, "Flash crowds and denial of service attacks: characterization and implications for CDNs and web sites," in *11th International Conference on World Wide Web*, 2002.

[20] F. Khan, L. Yuan, C.-N. Chuah, and S. Ghiasi, "A Programmable Architecture for Scalable and Real-time Network Traffic Measurements," in *ANCS '08: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, 2008.

[21] D. R. Morrison, "Patricia - practical algorithm to retrieve information coded in alphanumeric," *Journal of the ACM*, vol. 15(4), 1968.

[22] "CAIDA: Cooperative Association for Internet Data Analysis," http://www.caida.org/data/passive/backscatter_tocs_dataset.xml.

[23] G. Huang, S. Raza, S. Seetharamanh, and C.-N. Chuah, "Dynamic measurement-aware routing in practice," in *IEEE Networks Special Issue on Network Traffic Monitoring and Analysis*, 2011, pp. 29–34.

[24] "HP Blade Open Network Environment," http://h18004.www1.hp.com/products/blades/components/c-class-interconnects.html.

[25] "Internet2 abilene network," http://abilene.internet2.edu.

Faisal Khan is a Ph.D. from University of California, Davis, where his research focus has been in addressing real-time and computationally intensive problems in network security using reconfigurable computing. During the course of Ph.D., he also explored his research interests as a Computational Intern at Lawrence Livermore National Laboratory. Prior to joining UC-Davis, Faisal received his M.S. from KFUPM, Dhahran, Saudi Arabia and B.E. from NED University, Karachi, Pakistan in 2005 and 2001 respectively. His research interests span in providing embedded system solutions for mission critical and real-time problems. Faisal is currently working at Altera, San Jose, where he is involved in developing IPs for high-speed serial communications.

Nicholas Hosein is a PhD candidate in electrical and computer engineering at the University of California, Davis. His research interests include computer architecture, cyber-physical systems, and body sensor networks. He received his B.S. degree from the University of California, Berkeley in 2008.

Chen-Nee Chuah is a Professor in Electrical and Computer Engineering at the University of California, Davis. She received her Ph.D. in Electrical Engineering and Computer Sciences from the University of California, Berkeley. Her research interests include Internet measurements, network management, anomaly detection, online social networks, and vehicular ad hoc networks. Chuah is an ACM Distinguished Scientist. She received the NSF CAREER Award in 2003, and the Outstanding Junior Faculty Award from the UC Davis College of Engineering in 2004. In 2008, she was named a Chancellor's Fellow of UC Davis. She has served on the executive/technical program committee of several ACM and IEEE conferences. She was an Associate Editor for IEEE/ACM Transactions on Networking from 2008 to 2013.

Soheil Ghiasi is an associate professor of electrical and computer engineering at the University of California, Davis. His research interests include architecture, design methodologies, and design automation techniques for embedded systems. He received his B.S. degree from Sharif University of Technology, Tehran, Iran in 1998, and his M.S. and Ph.D. in Computer Science from University of California, Los Angeles in 2002 and 2004, respectively. He has served on the organizing and technical program committees of numerous conferences, and currently serves as an associate editor of the Journal of Reconfigurable Computing.

Puneet Sharma is a Principal Research Scientist at HP Labs where he conducts research on Data Center Networks, Cloud Architectures, Network Monitoring and Mobility. Prior to joining the HP Labs, he received his PhD in Computer Science from the University of Southern California. He also holds a B.Tech. in Computer Science and Engineering from IIT Delhi. His work on Mobile Collaborative Communities was featured in the New Scientist Magazine. He has participated in various standardization efforts. He contributed to the UPnP's QoS Working Group efforts as co-author for QoSv3 standards. Earlier, he had co-authored the IETF standards' RFCs on the multicast routing protocol PIM. Puneet is a Distinguished Scientist of the ACM and a Senior Member of the IEEE.

APPENDIX A
ALGORITHMS

Algorithm 2: Equilibrium Rollback

```
   /* Decision Phase                                   */
1  for R_i ∈ R do
2      if (R_i.Size > Size_Th) then
3          if Granularity(R_i) = MAX then
4              E_f ← E_f + R_i
5          else R.replace {R_i, Expand (R_i,Φ)}
6      else if Granularity(R_i) > 1 then
7          R.replace {R_i, Collapse (R_i)}
8      else R_i.Drop
```

Algorithm 3: MRT Algorithm

input : P_t: Packet enumerator at time t
input : Φ: The expansion/zoom ratio
input : R: Active rule-set
input : $R_i.Size$: Aggregate size for rule R_i
input : δ: Measurement Interval
input : θ: Threshold bandwidth consumption ratio
input : λ: Link rate (bits/second)
output: $E_f\{\}$: set of elephant-flows

```
   Size_Th ← λ * θ * δ
   /* Measurement Phase                                */
1  while t ≤ δ do
2      for R_i ∈ R do
3          if R_i = P_t then
4              R_i.Size ← R_i.Size + P_t.Size
   /* Decision Phase                                   */
5  for R_i ∈ R do
6      if (R_i.Size > Size_Th) then
7          if Granularity(R_i) = MAX then
8              E_f ← E_f + R_i
9          else R.replace {R_i, Expand (R_i,Φ)}
10     else R_i.drop
```

Algorithm 4: Flow Momentum

λ_i : Flow rate for flowset/flow i
k : Algorithmic iteration

```
   /* Measurement Phase                                */
1  while t ≤ δ do
2      for R_i ∈ R do
3          if R_i = P_t then
4              R_i.Size ← R_i.Size + P_t.Size
5              R_i.M ← R_i.M + P_t.Size
   /* Decision Phase                                   */
6  for R_i ∈ R do
7      λ_i ← R_i.M/k * δ
8      if (R_i.Size > Size_Th) then
9          if Granularity(R_i) = MAX then
10             E_f ← E_f + R_i
11         else R.replace {R_i, Expand (R_i,Φ)}
12         R_expanded.M ← R_parent.M
13     else if (λ_i/λ ≥ θ) then
14         R_i.Hold
15     else Ri.Drop
```