

# Pegasus: Precision Hunting for Icebergs and Anomalies in Network Flows

Sriharsha Gangam\*, Puneet Sharma<sup>+</sup>, Sonia Fahmy\*

\*Purdue University, <sup>+</sup>HP Labs

E-mail: sgangam@purdue.edu, puneet.sharma@hp.com, fahmy@purdue.edu

**Abstract**—Accurate online network monitoring is crucial for detecting attacks, faults, and anomalies, and determining traffic properties across the network. With high bandwidth links and consequently increasing traffic volumes, it is difficult to collect and analyze detailed flow records in an online manner. Traditional solutions that decouple data collection from analysis resort to sampling and sketching to handle large monitoring traffic volumes. We propose a new system, Pegasus, to leverage commercially available co-located compute and storage devices near routers and switches. Pegasus *adaptively* manages data transfers between monitors and aggregators based on traffic patterns and user queries.

We use Pegasus to detect *global icebergs* or *global heavy-hitters*. Icebergs are flows with a common property that contribute a significant fraction of network traffic. For example, DDoS attack detection is an iceberg detection problem with a common destination IP. Other applications include identification of “top talkers,” top destinations, and detection of worms and port scans. Experiments with Abilene traces, sFlow traces from an enterprise network, and deployment of Pegasus as a live monitoring service on PlanetLab show that our system is accurate and scales well with increasing traffic and number of monitors.

## I. INTRODUCTION

Detecting anomalies, attacks, and faults, and determining network traffic properties require online monitoring support. Accurate monitoring also aids network operators in optimizing network performance in a timely manner. Network monitoring entails (1) data collection at spatially separated *monitors*, and (2) processing aggregated monitored data at *aggregators*. The aggregators maintain monitoring data and extract globally relevant information that may not be observable at individual monitors. High bandwidth links have led to increasing traffic volumes, and have made it challenging to collect and process detailed flow records across different monitors at a centralized aggregator in an online manner. The monitored data that is *of interest* – the anomalies, attacks, top bandwidth consumers and worms – is typically much smaller than the entire monitoring data collected, making it inefficient to transfer the entire flow record data sets to a centralized location.

To deal with the large size of monitoring data and lack of compute and storage resources at the monitoring locations, standards such as NetFlow [1] and sFlow [10] resort to *sampling* data before exporting it to the centralized location. *Sketches* have also been proposed to represent flow information in succinct forms [12], [16], [18], [22], [35]. However, solutions relying on sampling or sketches invariably introduce false positives and false negatives, as (1) they are not designed

for aggregation queries (2) the analysis engine has to operate over a lossy data set. Sampling and sketching parameters also need to be set without a priori knowledge of the traffic and user queries.

We argue that network monitoring is more effective when collection of monitoring data and analysis of the data are coupled via a *feedback mechanism*. In other words, the monitoring requirements of the network operator should dictate the load of the monitoring tasks. Such an adaptive (or iterative) approach to network monitoring reduces the monitoring cost while maintaining high measurement accuracy. The feedback-based mechanism can use monitoring range assignment (what flows should each monitor collect and maintain) and sampling assignment (the sampling rate of the flows at different monitors) as tuning knobs for the monitoring task. With recent advances in hardware technology, co-located storage and compute can be supported in network devices at a low cost. For example, the Cisco SRE modules [2] or HP ONE blades [5] connected to network switches via two 10 Gbps links can make the feedback-based monitoring system feasible. We leverage recent advances in co-located compute and storage at network devices to design a feedback-based network monitoring system which we call *Pegasus*.

As one of the application instances, we use Pegasus to detect *global icebergs*, a set of flows with a common property contributing a significant amount of traffic across the entire network [25], [41], [42]. Example icebergs include bandwidth-consuming flows, distributed denial of service (DDoS) attacks, heavy-hitters in content distribution networks (CDNs) that denote “important” objects, anomalies, and worms. Current commercial network monitoring systems like the InMon *Traffic Sentinel* [6] support iceberg queries such as identifying “top talkers” and top destinations in order for network operators to detect congestion and anomalies. Other example iceberg queries are presented in MIND [32]. The main challenge with global iceberg detection is that icebergs may appear small at individual monitors, but contribute significant traffic when aggregated over all monitors. For instance, a DDoS attack can go undetected at a monitor but when the monitoring data across all the monitors is analyzed collectively, the attack can be detected and appropriate steps can be taken.

Pegasus comprises multiple aggregators and monitors, each deployed on a co-located compute and storage unit to share the responsibility of monitoring the entire network. We propose an aggregation technique for Pegasus that uses a feedback-based

algorithm to detect iceberg flows with low monitoring cost and high accuracy.

The key contributions of this work are: (1) The design of Pegasus, a system using co-located compute and storage for distributed network monitoring, (2) An adaptive aggregation method for Pegasus to answer general queries made possible by our *sketch-set* representation, and (3) With extensive experiments with traces from three different sources and a live PlanetLab deployment, we show that Pegasus can provide high precision monitoring with significant reduction in communication overhead for the iceberg detection problem.

**Roadmap:** Section II defines the iceberg detection application. Section III describes the Pegasus adaptive aggregation system and approach. Section IV explains our algorithms. Section V experimentally compares Pegasus to previous approaches. Section VI summarizes related work, and Section VII concludes the paper.

## II. EXAMPLE: GLOBAL ICEBERG DETECTION

We use global iceberg detection as a case study to illustrate Pegasus operation. The iceberg detection problem entails identifying large flows with a common property denoted by a *flow identifier*. Traditionally, the flow identifier is 5-dimensional (source IP, destination IP, source port, destination port, and protocol), but it may be one-dimensional as well. For example, if the flow identifier is the source IP address, the icebergs correspond to the top talkers or major traffic sources in the network. If the flow identifier is a destination IP, large flows can aid in detecting DDoS attacks. Similarly, destination IPs associated with a larger number of connections (distinct port numbers) may be a sign of suspicious port scanning activity. More generally, the flow records can be mapped to a collection of flow signatures based on monitoring requirements. Aggregation on these signatures can lead to the discovery of network anomalies.

A stream of (one-dimensional) flow records at each monitor is represented by the tuple  $(id, size)$ . The field  $id$  is a flow identifier and  $size$  is the size of the flow. Let  $size(id)$  denote the size of the flow with identifier  $id$ . The iceberg detection problem corresponds to finding the set of flow identifiers  $I$  from the total set of  $N$  distinct identifiers across all the monitors, with total size  $S = \sum_{i \in N} size(i)$ , such that  $\forall i \in I, size(i) \geq S\theta$ , where  $S \gg |N| \gg 1/\theta$  [30]. In other words, iceberg flows have an aggregated size (across the monitors) greater than  $S\theta$ , where  $\theta$  is the global iceberg threshold factor represented as a fraction of the total size. Such iceberg flows are limited in number in comparison to the total number of flows. It is important to note that an iceberg flow may not appear to be a large flow at *any* of the individual monitors. Although the input parameter  $\theta$  can be hard to determine, in some applications like ensuring SLAs, the threshold value can be obtained from SLA agreements.

## III. PEGASUS SYSTEM OVERVIEW

Pegasus comprises two major software components: monitors and aggregators, executing on co-located compute and

storage units. The monitors are responsible for storing and manipulating the local monitoring data received from *agents* (probes). The agents are software components that run within a switch or router to generate flow records. An sFlow agent is one such example that runs on sFlow enabled switches. The monitor typically receives and stores flow records from its nearest agent. For example, a monitor running on Cisco SRE module [2] or an HP ONE blade [5] on a switch chassis receives flow records from agents running on the switch of the same chassis. With dedicated interfaces connected to these co-located compute and storage units, an agent can transfer large volumes of high precision flow data to its local monitor. Currently, a typical minimal blade is configured with Xeon quad-core, 2.2 GHz processors and 128 GB disk memory, enabling the monitors to store detailed flow records and execute expensive monitoring tasks. Monitoring queries like those issued on NetFlow or sFlow data can be distributed and executed over Pegasus monitors. However, some complex aggregation monitoring queries like *top-k* monitoring or global iceberg detection still remain a challenge. In Pegasus, aggregators are responsible for interacting with monitors and providing answers to aggregation queries. Given that monitors can collect large volumes of fine-grained flow records, aggregation incurs a significant load on the network. To reduce to communication overhead between monitors and aggregators, Pegasus uses an adaptive aggregation process.

### A. Overview of Adaptive Aggregation

A naive approach for iceberg detection is to send all flow records to a centralized location for analysis. The volume of flow records, however, can be prohibitively large. Sketching and sampling summarize a large data set using a representative smaller set of monitoring data. Iceberg detection then involves combining flows with common identifiers and estimating their sizes. Sketching and sampling are practical but approximate, usually yielding false positives or negatives. Adaptive aggregation in Pegasus also uses such summarized monitoring data to reduce to monitoring overhead. However, it goes a step further to resolve any uncertainty in the summarized representation, *i.e.*, adaptive aggregation is iterative in nature. Initial iterations summarize a larger volume of data with little memory (low accuracy). After the aggregator provides feedback to the monitors, subsequent iterations accurately summarize more specific (to the monitoring application) data. To support such feedback based resolution, we propose a *sketch-set* flow record representation.

We define a sketch-set as an approximate low memory representation of a collection of flows and flow sizes. Each monitor creates sketch-sets based on the flow records it receives and sends them to the aggregator. The aggregator combines sketch-sets from different monitors, identifies sketch-sets that are likely to contain iceberg flows and eliminates unimportant sketch-sets. For example, suppose that the size of sketch-set is defined as the size of the smallest flow within the sketch-set. If the sum of sizes of sketch-sets that (approximately) represent the same set of flows exceed the iceberg threshold

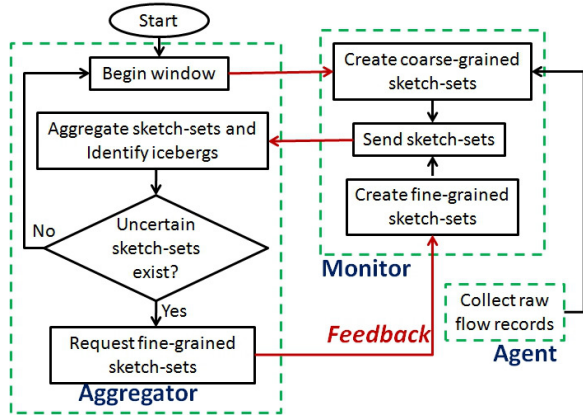


Figure 1: Adaptive aggregation for iceberg detection

( $S\theta$ ), the sketch-sets are likely to contain an iceberg flow. The aggregator can then ask the monitors for a more accurate representation for these specific sketch-sets and begin the next iteration. As another example, assume that the size of a sketch-set is the size of the largest flow within the sketch-set. If the sum of sizes of sketch-sets that represent the same set of flows do not exceed  $S\theta$ , the sketch-sets are guaranteed not to have an iceberg flow. To enable comparisons of common flows among different sketch-sets and their sizes, they need to support basic set operations like union, intersection and subtraction.

Figure 1 pictorially describes the adaptive aggregation process. The aggregator terminates the iterative process when all icebergs have been detected. To ensure convergence, monitors send sketch sets of finer-granularity in each subsequent iteration. The next time window begins when the aggregator detects all the icebergs and there are no uncertain sketch-sets.

In summary, to enable an aggregation mechanism that adaptively and iteratively identifies global icebergs, we propose the following constraints on a sketch-set: **(C1)** A sketch-set must support an arbitrary collection of flows. **(C2)** A sketch-set must represent all flows used in generating the sketch-set. It may additionally represent flows not in the set. This introduces uncertainty (lossy monitoring information) but reduces memory overhead. **(C3)** A sketch-set must support set operations (UNION, INTERSECTION, and SUBTRACTION) that can be manipulated to represent any arbitrary collection of flows. **(C4)** A sketch-set has one or more parameters that represent the *size* (e.g., number of bytes, packets, or connections) of the flows in the sketch-set. **(C5)** A sketch-set has a granularity parameter that characterizes the “coarseness” level of a sketch-set. This is used to quantify the accuracy of representation and ensure that the feedback based mechanism converges. **(C6)** A sketch-set must consume much less memory than all flows represented by the sketch-set<sup>1</sup>.

#### IV. PEGASUS ICEBERG DETECTION

##### A. Sketch-set Representation

We propose a simple range-based sketch-set representation that satisfies conditions **C1–C6** in the last section. A one-

<sup>1</sup>This condition may fail for a sketch-set representing a single flow or a small set of flows

dimensional sketch-set is a 4-tuple containing the following elements: (1) *startId*: All flows that belong to the sketch-set satisfy  $id \geq startId$ . (2) *endId*: All flows that belong to the sketch-set satisfy  $id \leq endId$ . (3) *maxSize*: The upper bound on the size of any flow that belongs to sketch-set. (4) *minSize*: The lower bound on the size of any flow that belongs to sketch-set. A sketch-set query request sent by an aggregator contains only the *startId* and *endId* elements. The *id* and *size* of a flow depends on the specific monitoring application (e.g., a port scanning application can have *id* as the destination IP and *size* as the number of connections).

Alternative summarized set representations for flows such as counting Bloom filters or other sketches [12], [16], [18], [22], [35] cannot satisfy all sketch-set requirements and are hence unsuitable for adaptive aggregation (e.g., a Bloom filter cannot perform SUBTRACTION without violating condition **C2**). Flows that are represented by a multi-dimensional identifier (e.g., source address and destination address) require a *startId* and *endId* for each dimension. When the *startId* and *endId* are equal, the sketch-set has a single item (*singleton sketch-set*).

##### B. Coarse Sketch-set Creation at Monitors

Monitors perform the UNION operation on the sketch-sets on a potentially large set of flows. Flows with similar flow sizes are grouped in the same sketch-set. An example of the adaptive aggregation algorithm is depicted in Figure 2. The example finds destination IP addresses whose aggregated packet count ( $S\theta$ ) exceeds 200. Flows from *Agent 1* with identifiers  $128.41.10.10$  to  $128.41.10.50$  are grouped in the same sketch-set.

Granularity ( $G$ ) of a sketch-set is an indicator of how approximate the representation of flows is (the compression factor) in the sketch-set. For the iceberg detection application, the granularity is defined as the difference between *maxSize* and *minSize*. Intuitively, when the granularity of a sketch-set is bounded, all flows represented by the sketch-set have a similar size. In Figure 2, *Agent 1* has created coarse-grained sketch-sets with granularity 15 (i.e., the sizes of the flows in the sketch-sets do not differ by more than 15 from each other).

To ensure convergence, the monitors send finer granularity sketch-sets over each successive iteration. We achieve this by decreasing the granularity threshold  $G$  by an exponential factor after each iteration. This exponential factor is called the *granularity reduction factor* ( $\alpha$ ) and is an input parameter to the adaptive aggregation algorithm. After the  $i^{th}$  iteration, the value of  $G$  is  $S\alpha^i$  where  $\alpha \in (0, 1)$ .

To speed up the algorithm, some large flows (potential global icebergs) are sent as singleton sets (e.g.,  $128.41.10.210$  in Figure 2). We use a second parameter called the *local iceberg threshold factor* ( $\beta$ ) to classify such large flows. If a flow size is  $LIS$  or more, it is sent as a singleton sketch-set (with maximum resolution), where  $LIS = S\theta/M\beta$ ,  $\beta \in (0, 1)$  and  $M$  is the number of monitors. Section V-A1 gives intuition on how the parameters work together and their impact on performance. In summary, parameters  $\alpha$  and  $\beta$  are used to control the granularity variation across iterations.

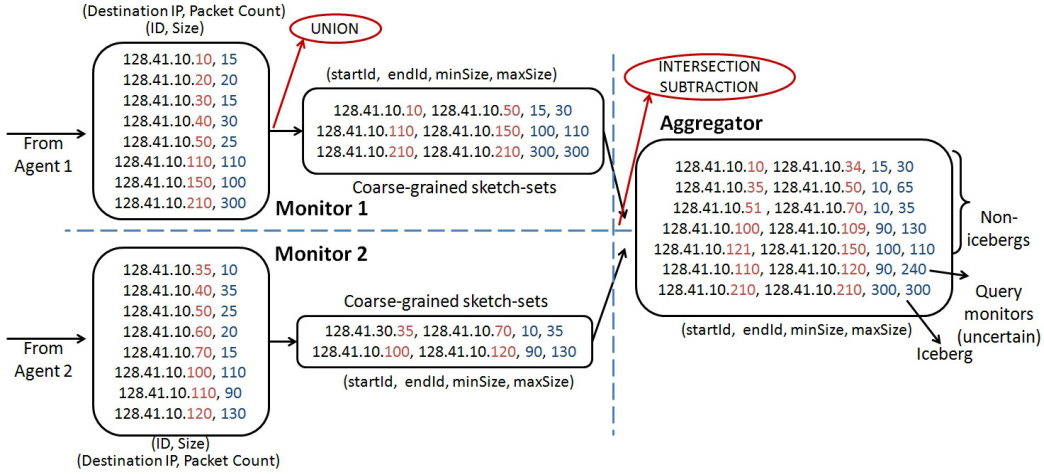


Figure 2: Example showing coarse grained sketch-set generation and the aggregation result (iceberg threshold size is 200).

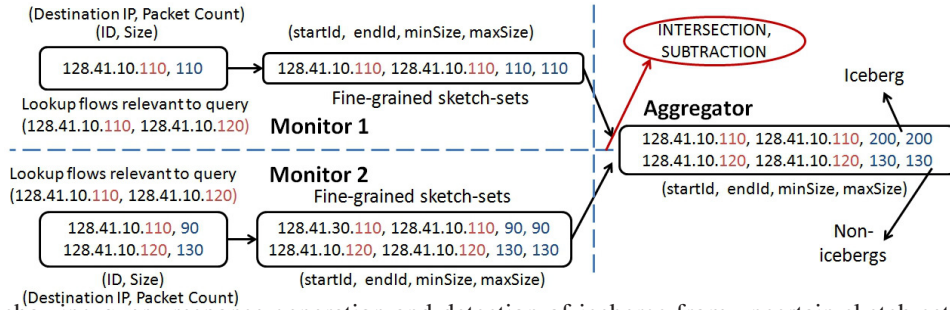


Figure 3: Example showing query response generation and detection of icebergs from uncertain sketch-sets (iceberg threshold size is 200)

Procedure *CreateCoarseSketchSets()* creates coarse-grained sketch-sets with a bounded granularity ( $G$ ) by applying Procedure *DISJOINT-UNION()* on singleton sketch-sets. After this operation, the constraints on the four elements of the sketch-set ( $startId$ ,  $endId$ ,  $maxSize$ , and  $minSize$ ) are not violated. The variable  $SCSS$  denotes the set of all coarse-grained sketch-sets and is the end result of the procedure. The method uses the variable  $LIS$  to classify local icebergs.

**Input:** Sketch-set:  $ss1$   
**Input:** Sketch-set:  $ss2$   
**Output:** The union of two disjoint sketch-sets:  $ss$   
 $ss \leftarrow \emptyset$ ;  
 $ss.startId \leftarrow MIN(ss1.startId, ss2.startId)$ ;  
 $ss.endId \leftarrow MAX(ss1.endId, ss2.endId)$ ;  
 $ss.minSize \leftarrow MIN(ss1.minSize, ss2.minSize)$ ;  
 $ss.maxSize \leftarrow MAX(ss1.maxSize, ss2.maxSize)$ ;  
**return**  $ss$  ;

#### Procedure DISJOINT-UNION

### C. Sketch-set Aggregation across Monitors

When an aggregator receives the set of coarse grained sketch-sets from the monitors, it dismantles them into disjoint sketch-sets. Figure 2 shows an example of the disjoint set generation. The sketch-sets from different monitors are aggregated together and their  $maxSize$  and  $minSize$  estimates are updated (e.g., the second sketch-set at the

**Input:** The set of flow records (singleton sketch-sets) observed by the monitor:  $FR$   
**Input:** Granularity threshold:  $G$   
**Input:** Size of local iceberg:  $LIS$   
**Output:** A set of coarse grained sketch-sets:  $SCSS$   
 $ss \leftarrow \emptyset$  ;  
 $SCSS \leftarrow \emptyset$  ;  
**foreach**  $f \in FR$  **do**  
  **if**  $f.minSize \geq LIS$  **then**  
     $SCSS \leftarrow SCSS \cup f$  /\*  $f$  is large enough to be sent as a singleton sketch-set \*/;  
  **else**  
     $union \leftarrow DISJOINT-UNION(ss, f)$  /\* union of  $ss$  and  $f$  \*/;  
    **if**  $union.Granularity < G$  **then**  
       $ss \leftarrow union$  ;  
    **else**  
       $SCSS \leftarrow SCSS \cup ss$  ;  
       $ss \leftarrow f$  ;  
    **end**  
  **end**  
**end**  
 $SCSS \leftarrow SCSS \cup ss$  ;  
**return**  $SCSS$  ;

#### Procedure CreateCoarseSketchSets

```

Input: The set of coarse grained sketch-sets from all monitors:
SCSS
Output: Aggregated sketch-sets across all monitors with
estimated lower and upper bounds of item sizes: DS
DS ← ∅ ;
foreach ss ∈ SCSS do
  foreach ds ∈ DS do
    if INTERSECTION(ss, ds) ≠ ∅ then
      DS ← DS ∪ INTERSECTION(ds, ss) ;
      ss ← SUBTRACT(ss, ds) ;
      ds ← SUBTRACT(ds, ss) ;
    end
  end
  DS ← DS ∪ ss ;
end
return DS ;

```

#### Procedure Aggregation

Aggregator  $\{128.41.10.35, 128.41.10.50, 10, 65\}$  is the INTERSECTION of the sketch-sets  $\{128.41.10.10, 128.41.10.50, 15, 30\}$  from *Monitor 1* and  $\{128.41.10.35, 128.41.10.70, 10, 35\}$  from *Monitor 2*. The non-intersecting sketch-set range (e.g.,  $\{128.41.10.10, 128.41.10.34\}$  and  $\{128.41.10.51, 128.41.10.70\}$ ) of the sketch-sets do not have their *maxSize* and *minSize* estimates updated.

The *Aggregation()* method provides the pseudo code for the aggregation of sketch-sets. The definitions of the SUBTRACT and INTERSECTION functions specific to our sketch-set are omitted for brevity, but can be found in our technical report [24].

After aggregating sketch-sets and producing a collection of disjoint sketch-sets with updated min and max sizes, sketch-sets with a small *maxSize* (non-iceberg flows) are eliminated. If the *minSize* of a sketch-set exceeds the iceberg threshold size ( $S\theta$ ), it is guaranteed to have an iceberg. In Figure 2, all sketch-set which are grouped as *Non-Icebergs* do not have a flow whose *maxSize* more than 200. If the *maxSize* is larger than the iceberg threshold, the sketch-set is marked as *uncertain*. The sketch-set  $\{128.41.10.110, 128.41.10.120, 90, 240\}$  (at the aggregator) in Figure 2 has a *maxSize* of 240 which implies that there could be a flow in the sketch-set with an aggregated size more than 200. To resolve uncertain sketch-sets, the aggregator requests finer-grained sketch-sets from the appropriate monitors. With the example in Figure 2, the aggregator requests both the monitors for finer-grained sketch-sets whose flow identifiers (destination IP) lie within  $\{128.41.10.110, 128.41.10.120\}$ . A query sketch-set generated by the aggregator contains two fields (*startId* and *endId*) to indicate the range of flow identifiers that can have global icebergs. The size parameters (*minSize* and *maxSize*) are not included in the query.

#### D. Monitor Response to Query

After the monitors receive requests for finer-grained sketch-set information, all flows corresponding to a given sketch-set query request are looked up. The flows are combined again using the *CreateCoarseSketchSets()* procedure with a smaller granularity threshold ( $\alpha G$ ). Since  $\alpha < 1$ , all the sketch-sets in

the iteration will have a finer resolution than in the previous one. Figure 3 continues the example of showing how monitors respond to a query for the second iteration. After a local lookup, both the monitors decide to send singleton sketch-sets. The aggregator processes these singleton sketch-sets and identifies global icebergs in sketch-sets that were uncertain in the previous iteration.

## V. EVALUATION

The Pegasus monitor and aggregator were implemented in  $\approx 3000$  lines of C++ code using the GNU Linux socket API. This includes wrapper scripts that convert flow records specified in multiple formats (sFlow, NetFlow, and flow-tools) to a common raw flow format (5-tuples) used by our implementation. The messages sent by the monitors to the aggregators include the number of sketch-sets followed by a binary representation of the list of sketch-sets.

We evaluate Pegasus using three setups: (1) A five-minute Abilene trace collected from eleven sites, with Pegasus (all monitors and an aggregator) deployed on a single host. (2) sFlow traces from an enterprise network, with Pegasus deployed on a blade server. (3) Pegasus deployment and monitoring of live PlanetLab traffic saved as NetFlow traces. Due to space limitations, performance benefits for the sFlow traces and scalability benefits of our adaptive aggregation method are presented in the appendix of our technical report [24].

We again use the iceberg detection application as a case study, evaluating its performance and communication overhead. The communication overhead is measured in terms of the percentage of communication cost savings over a naive approach. The cost of the naive approach is the total size of all records transferred from the agents to a centralized aggregator.

We detect global icebergs for a single dimension based on the destination IP address, destination port, or source port. Most of the real-world queries of top talkers, top destinations, and queries in MIND [32] are variants of this one-dimensional iceberg detection. In the Abilene and sFlow traces, one additional iceberg flow is inserted to increase the difficulty of the detection process. The size of the inserted global iceberg is close to the iceberg threshold and is equally distributed across all agents.

#### A. Abilene Trace Experiments

We use 5-minute 11-node Abilene NetFlow traces (also used in [25] to study global iceberg detection using the Sample-Sketch scheme). The flow traces were originally collected with a 1 in 100 sampling rate. The trace is *unsampled* by including additional small flows to perceive the overhead effects in storing and processing a large number of fine-grained flow records. For every sampled NetFlow record, nine small flow records are added such that they contribute about 20% of the total packet count. This way, a heavy tailed distribution of flow sizes is maintained to match typical Internet flow size distributions. After insertion of small flows, the data set contains close to 5 million records across all sites. This trace is consumed by Pegasus in a single time window.

As with most of previous work, Sample-Sketch [25] yields approximate answers due to its lossy nature. Sample-Sketch uses both sampling and sketching to identify global icebergs that are otherwise difficult to detect. We use the multistage sketch [22] as a representative of sketches such as [12], [16], [18], [30], [35] used in Sample-Sketch. The communication cost and accuracy of Sample-Sketch depend on two main parameters: the sampling parameter and the sketch parameter. The sampling parameter is the packet sampling frequency. The sketch parameter is a threshold value to decide if a flow should be categorized as a local iceberg. It is non-trivial to determine the best sample and sketch parameters. Additional sketch parameters (*e.g.*, number of hash functions used or number of counters) are not varied in our experiments. While the results given in [25] studied a limited range of the sampling parameter, we explore a much wider range of the sampling and sketch parameters.

To determine the best parameters for the Sample-Sketch method, icebergs are detected with the help of an *off-line oracle* that can yield complete solutions (*i.e.*, there are no false positives or false negatives). The lowest communication cost and corresponding sketch and sampling parameters for error-free solutions are recorded. Table I lists the set of parameters yielding the best solutions for various iceberg threshold values ( $\theta$ ). Pegasus uses constant parameters ( $\alpha = 0.05, \beta = 0.95$ ) in the Abilene and sFlow experiments for all iceberg threshold sizes. In Section V-A1 we observe that  $\alpha$  and  $\beta$  do not have a significant impact.

Figure 4 depicts the communication cost savings over the naive approach. We compare the best Sample-Sketch parameters with Pegasus for different iceberg sizes. We only record cases with no false positives or negatives. The figure shows that Pegasus detects all icebergs with higher communication cost savings. No Sample-Sketch parameters detect all iceberg flows for all iceberg thresholds (except for inefficient sampling and sketch parameters (50000:0.0001)). For example, in the case of (200000:0.001), the solution has false positives or negatives for  $\theta \in \{0.001, 0.002, 0.004, 0.008\}$  so the data points are missing. Pegasus adapts to the user query (variable  $\theta$ ) and monitoring data to produce a non-linear curve in Figure 4. A small value of  $\theta$  is an “easier” query and thus requires a lower communication cost.

Adaptive aggregation in Pegasus consumes little communication bandwidth. Suppose that the destination IP and the packet count take 4 bytes each. A naive solution would take  $8 \times 10^5 \text{ bytes} \approx 7.63 \text{ MB}$ . For an iceberg threshold  $\theta = 0.08$ , Sample-Sketch takes about 36 KB for parameter pair (175000:0.0001). The parameter set that matches the accuracy of Pegasus takes about 96.65 KB. Pegasus has a communication overhead of 8 KB, a 12 fold reduction in communication cost.

1) *Micro-benchmarks*: As previously discussed, our adaptive aggregation algorithm for detecting global icebergs can be tuned using: (1) a granularity reduction factor  $\alpha$ , and (2) a local iceberg threshold factor  $\beta$ . These parameters determine the granularity of the sketch-sets constructed at the monitors.

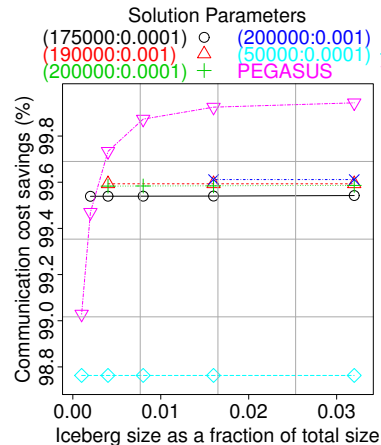


Figure 4: Comparison of Sample-Sketch (with oracle) and adaptive aggregation in Pegasus on the Abilene trace

TABLE I: Best parameters for Sample-Sketch with no false positives or negatives for the Abilene trace

Iceberg size as a fraction of total size ( $\theta$ )	Sampling parameter	Sketch parameter
0.001	50000	0.0001
0.002	175000	0.0001
0.004	190000	0.001
0.008	200000	0.0001
0.016	200000	0.001
0.032	200000	0.001

With a smaller  $\beta$ , our algorithm sends more singleton sketch-sets. A large value of  $\alpha$  leads to the creation of coarse-grained sketch-sets that represent many more flows. Consequently, they provide limited information on the flows over each iteration leading to slow convergence. If  $\alpha$  is small, the sketch-sets have increasingly higher granularity in each iteration and provide more information on the flows. This leads to faster convergence at the cost of higher communication overhead.

Figure 5 and Figure 6 depict the communication cost savings and the number of iterations taken for different values of  $\alpha$  and  $\beta$  in our Abilene trace. We note that  $\beta$  has a negligible influence on the number of iterations and the communication cost. This is possibly due to the heavy tailed distribution of the flow sizes. As expected, the number of iterations increase with larger values of  $\alpha$ . However, it has little impact on the communication overhead. For small  $\theta$ ,  $\alpha$  should be sufficiently large to reduce communication costs. In other words, for small  $\theta$ , a faster converging solution can have a higher communication overhead. In high latency networks, it is important to limit the number of iterations to ensure that the algorithm converges before the next time window begins. This is especially important in our PlanetLab deployment where the hosts experience long and variable delays.

2) *Scalability with the Number of Monitors*: Pegasus uses co-located compute and storage at the monitors. However, it may not be possible to have a monitor at every flow-generating source, *i.e.*, every *agent*. In this case, the agents send flow

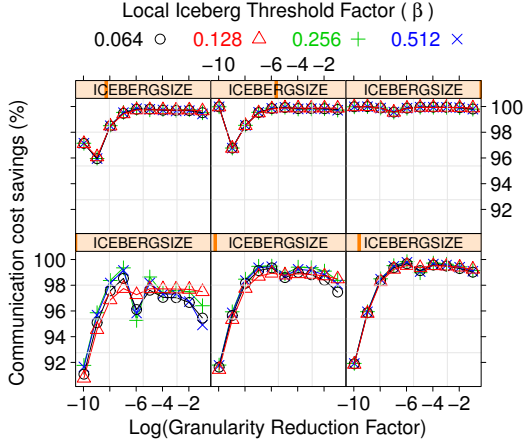


Figure 5: Communication cost for different  $\alpha$ ,  $\beta$  parameters and iceberg sizes ( $\theta=0.001, 0.002, 0.004, 0.008, 0.016, 0.032$ ) on the Abilene trace

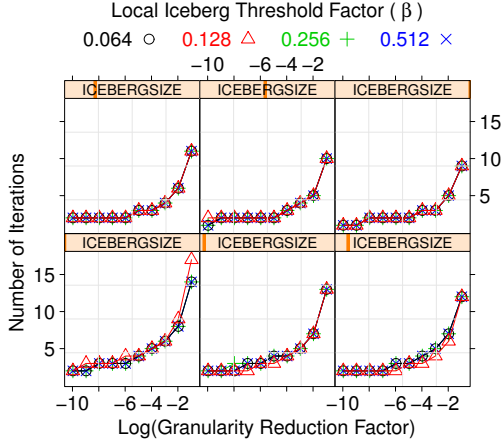


Figure 6: Number of iterations taken to converge for different  $\alpha$ ,  $\beta$  parameters and iceberg sizes ( $\theta=0.001, 0.002, 0.004, 0.008, 0.016, 0.032$ ) on the Abilene trace

records to one of the fewer available monitors and incur additional communication cost. For such agents, the communication cost is the same as that of the naive approach. Agents co-located with monitors do not incur any additional overhead. In the case when there are no monitors, the communication overhead is equivalent to the naive approach where all the flow records are transmitted to a centralized aggregator.

Figure 7 depicts the communication cost savings for a varying number monitors for the Abilene trace. We note that the communication cost savings increase almost linearly with the number of monitors. The iceberg threshold  $\theta$  has negligible impact on the communication cost in comparison to the number of monitors. Similar results hold for our sFlow trace consisting of 249 monitors.

### B. Live PlanetLab Traffic Monitoring

Though Pegasus deployment is primarily aimed at switches and routers with co-located compute and storage, deployment can be naturally extended to end hosts. Deployment of Pegasus on ingress switches or routers can detect attacks and anomalies

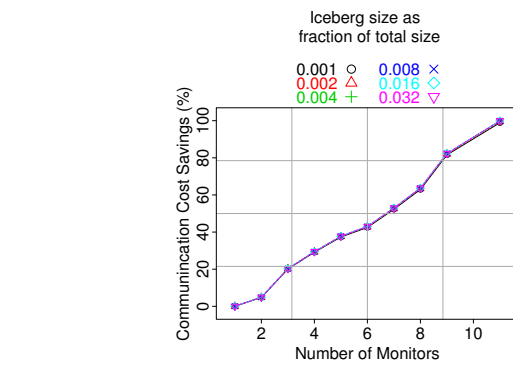


Figure 7: Communication cost savings for different numbers of monitors for the Abilene trace

from the outside network and take necessary preventive action (e.g., detecting DDoS attacks and filtering unusual traffic). Deployment on end hosts can be useful to prevent accidental attacks initiated within the network (e.g., detect if the end hosts are compromised and have formed a botnet).

We deployed Pegasus on PlanetLab to monitor network traffic originating from PlanetLab nodes across all slices. The existing PlanetLab traffic monitoring system, PlanetFlow [27], periodically transfers PlanetLab log traffic to a centralized location to support traffic queries. This is an example of the naive approach. Unlike Pegasus, PlanetFlow decouples the collection of data from its processing and analysis, which entails a centralized collection and analysis of a large amount of monitoring data (about 1 TB of data every month [8]). Monitoring PlanetLab with Pegasus provides a real-time, aggregated query support with low communication overhead.

We again consider the top destination iceberg queries. We determine global iceberg flows ( $\theta = 0.01$ ) based on destination IP, destination port, or source port for outgoing traffic from PlanetLab nodes across all slices in real-time. Such online monitoring can prevent accidental DDoS attacks originating from PlanetLab nodes [11]. Similarly, port scanning activity reported in [11] can be detected by adaptively aggregating the number of unique ports associated with destination IPs. Pegasus can be easily extended to monitor PlanetLab slices for CPU, disk, and traffic utilization.

At each PlanetLab node, the outbound traffic from all slices is collected in real time and stored in one-hour NetFlow version 5 dump files. We use a NetFlow replayer to read the tail of the latest dump file, parse the flow records, and replay them to a Pegasus monitor running on the same PlanetLab host machine. All the NetFlow records recorded across PlanetLab hosts during the same UTC time window are replayed and processed together. Pegasus actively identifies and removes slow and unresponsive PlanetLab nodes from participating so that online monitoring time constraints are met. Pegasus is deployed on 304 PlanetLab hosts out of which  $\approx 250$  hosts participate in the experiments. Due to space limitations, results from PlanetLab replay experiments, are presented in the appendix of our technical report [24].

Pegasus also provides a live iceberg detection service by

consuming flow records as soon as they are produced. Pegasus iceberg destinations can be viewed at a participating PlanetLab host [7]. Over a 14-hour plus 40-minute live iceberg detection service on destination-based flows with  $\theta = 0.01$  over  $\approx 250$  PlanetLab nodes, Pegasus consumes about 403 MB monitoring bandwidth. This amounts to about 1.87 KB of monitoring data on average per PlanetLab node every minute. The naive approach of moving all flow records to a centralized location would take an aggregate total of 2.26 GB, amounting to traffic of  $\approx 10.8$  KB per PlanetLab node every minute. We observed that the compute and storage overhead at each monitor is minimal and does not impact the system. On a typical PlanetLab host, an hour long uncompressed NetFlow trace consumes less than 10 MB. Such storage overheads are easily supported by Pegasus monitors running on end-hosts, Cisco SRE modules or HP ONE blades.

1) *PlanetLab Network Traffic*: In our most recent deployment of the Pegasus live monitoring service on PlanetLab for one week, we found that, not surprisingly, most of PlanetLab outbound traffic is destined towards other PlanetLab hosts. There were only a few hosts *without* a university or research lab domain name (*e.g.*, Comcast and Cox). Due to the lack of public information on slices running on each PlanetLab host, it is difficult to verify high activity slices with PlanetFlow [9].

Iceberg detection on source ports and destination ports reveal a few interesting phenomena. Figures 8a and 8b show the sorted iceberg source ports and destination ports respectively. The iceberg flows on source port were collected by the Pegasus live service ( $\theta = 0.01$ ) on PlanetLab during one day from  $\approx 9AM$  EST May 7 to  $\approx 9AM$  EST May 8. The icebergs on destination ports were collected from  $\approx 9AM$  EST May 8 to  $\approx 9AM$  EST May 9 and aggregated across all the time windows. We note that port 8 (unassigned), 3 (CompressNET), 80 (HTTP) and 22 (SSH) make it into the list of iceberg source ports with aggregated packet count more than 4 million. Similarly, ports 0 (reserved), 3 (CompressNET), 53 (DNS), 80 (HTTP), 443 (HTTPS) make it into the list of iceberg destination ports. It is important to note that these observations are only based on the outgoing PlanetLab traffic, since we have no access to incoming traffic.

Based on the presence of port 22 in Figure 8a and its absence in Figure 8b, we hypothesize that significant packets are generated by SSH daemons running on PlanetLab hosts sending traffic to non-PlanetLab hosts. Similarly, the presence of port 443 in Figure 8b and its absence in Figure 8a suggests that PlanetLab hosts are extensively using HTTPS to access non-PlanetLab servers. A similar argument can be applied to port 53 (DNS). Not surprisingly, many PlanetLab experiments issue significant DNS lookups.

In contrast, ports 3 and 80 appear as both source and destination ports for iceberg flows. This implies that there is considerable traffic originating from both clients *and* servers running on PlanetLab nodes. Other examples in this category include ports 7006, 8089, 7107, and 4121.

We specifically looked for the traffic associated with the CoDeeN [3] service which runs on ports 3128, 3127, or 3128.

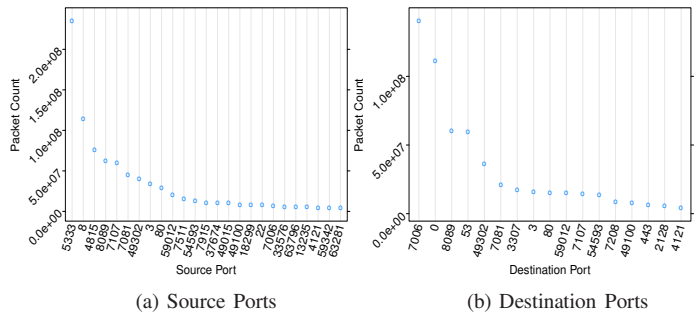


Figure 8: Global icebergs on source and destination ports which contribute more than 4 million packets in PlanetLab’s outgoing traffic in one day ( $S\theta = 4$  million).

Port 3128 was ranked 89<sup>th</sup> among the aggregated icebergs of source ports indicating that it was not a major traffic contributor during this time frame. The CoDNS [4] service running on port 4119 did not rank high in our list of aggregated global iceberg ports. While our observations hold for a one-day aggregated set of global icebergs, many of the ports in Figure 8b and Figure 8a were consistently flagged as iceberg flows in smaller time windows.

## VI. RELATED WORK

Previous sections of this paper discussed the most closely related work to ours: Sample-Sketch [25], ProgME [40], and MIND [32]. In this section, we discuss a few other related approaches. A summable sketch representation based on second frequency moments [41] detects iceberg flows with low error and theoretical performance guarantees. As with Sample-Sketch [25], this solution may still have false positives or negatives. In CSAMP [37], monitor placement and flow range distribution across monitors are varied with the goal of balancing the monitoring load. In contrast, our work focuses on how collection and processing of flow data can impact global aggregation queries. LADS [36] provides scalable multi-stage DDoS detection in provider networks by starting with low cost SNMP data and working towards more expensive high granularity NetFlow data.

Previous literature on continuous global heavy hitter detection and distributed *top-k* monitoring can be directly incorporated into the Pegasus framework. The threshold algorithm (TA) [23] maintains  $k$  entries for each of the *top-k* items. The TA algorithm, however, can take a large number of iterations to converge when the number of monitors is large. The TPUT [14] algorithm limits the number of iterations to three by estimating a local threshold value for monitors. TPUT can be seen as a special case of our adaptive aggregation algorithm when  $\alpha$  and  $\beta$  are both set to 1. Yu *et al.* [39] extend the TPUT algorithm by accounting the data distribution.

Continuous streaming algorithms [13], [31], [33] adopt a *filtering-based* approach. Local monitors process continuous data streams and only sends information that is likely to trigger a global change in heavy-hitter detection or *top-k* identification. Similarly, Huang *et al.* [26] explore a filtering-based approach that leverages in-network processing to detect (PCA-



based) network anomalies. More recent theoretical work using the continuous streaming model [19], [20], [28], [29], [38] proposes communication efficient algorithms and gives bounds to calculate aggregation functions approximately. In contrast to the continuous streaming model, our adaptive aggregation algorithm works in batches of consecutive windows where the goal is to detect global icebergs in the recent past (say the last 1 minute). While Pegasus does not have positives or negatives within a window, it can miss global icebergs that overlap between two consecutive windows. A sliding window model used by [15], [21], [34] provides communication efficient aggregation algorithms within a given approximation error. A recent survey [17] briefly explains some of the developments in the area.

## VII. CONCLUSIONS

We present a system called Pegasus to detect network anomalies by leveraging additional storage and compute from commercial devices such as the HP ONE blades [5] and Cisco SRE modules [2]. To detect global icebergs, we develop an adaptive aggregation mechanism that iteratively extracts selective measurement data from different monitors to reduce the communication overhead and avoid false positives or negatives. To enable this, we propose a generic and flexible sketch-set representation to describe and process monitoring data. Our three data sets/deployments (Abilene’s Netflow, enterprise network sFlow and PlanetLab’s NetFlow) demonstrate key benefits in terms of accuracy, reduced overhead and scalability. We also show that Pegasus is scalable with number of monitors and increasing network traffic [24], and can provide a live iceberg detection service on the PlanetLab testbed [7].

**Acknowledgments:** The authors would like to thank Guanyao Huang (UC Davis) for providing the Abilene traces with Sample-Sketch [25] source code and Sapan Bhatia (Princeton Univ.) for access to live PlanetLab NetFlow traces.

## REFERENCES

- [1] Cisco NetFlow. <http://www.cisco.com>.
- [2] Cisco Services Ready Engine. [http://www.cisco.com/en/US/products/ps10598/prod\\_module\\_series\\_home.html](http://www.cisco.com/en/US/products/ps10598/prod_module_series_home.html).
- [3] CoDeeN. <http://codeen.cs.princeton.edu/>.
- [4] CoDNS. <http://codeen.cs.princeton.edu/codns/>.
- [5] HP proCurve ONE. <http://h17007.www1.hp.com/us/en/products/application-delivery/index.aspx>.
- [6] Inmon Traffic Sentinel. <http://www.inmon.com/products/trafficsentinel.php>.
- [7] Pegasus service on PlanetLab. <http://ricepl-5.cs.rice.edu:8000/icebergs.html>.
- [8] PlanetFlow. <http://www.cs.princeton.edu/~sapanb/planetflow2/>.
- [9] PlanetFlow Slice Statistics. <http://planetflow.planet-lab.org/#slices>.
- [10] sFlow. <http://www.sflow.org>.
- [11] R. Adams. Distributed System Management: PlanetLab Incidents and Management Tools. Technical Report PDN-03-015, PlanetLab Consortium, November 2003.
- [12] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *Proc. of PODS*, pages 286–296, 2004.
- [13] B. Babcock and C. Olston. Distributed top-k monitoring. In *Proc. of ACM SIGMOD*, pages 28–39, 2003.
- [14] P. Cao and Z. Wang. Efficient top-k query calculation in distributed networks. In *Proc. of ACM PODC*, pages 206–215, 2004.
- [15] H.-L. Chan, T. W. Lam, L.-K. Lee, and H.-F. Ting. Continuous monitoring of distributed data streams over a time-based sliding window. *CoRR*, abs/0912.4569, 2009.
- [16] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. *Theor. Comput. Sci.*, pages 3–15, 2004.
- [17] G. Cormode. Continuous distributed monitoring: a short survey. In *Proc. of AlMoDEP*, pages 1–10, 2011.
- [18] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *J. Algorithms*, 55:29–38, 2004.
- [19] G. Cormode, S. Muthukrishnan, and K. Yi. Algorithms for distributed functional monitoring. *ACM Trans. Algorithms*, 7(2):21:1–21:20, Mar. 2011.
- [20] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang. Optimal sampling from distributed streams. In *Proc. of PODS*, pages 77–86, 2010.
- [21] G. Cormode and K. Yi. Tracking distributed aggregates over time-based sliding windows. In *Proc. of SSDBM*, pages 416–430, 2012.
- [22] C. Eстан and G. Varghese. New directions in traffic measurement and accounting. In *Proc. of ACM SIGCOMM*, pages 323–336, 2002.
- [23] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4):614–656, June 2003.
- [24] S. Gangam, P. Sharma, and S. Fahmy. Pegasus: Precision Hunting for Icebergs and Anomalies in Network Flows. <http://www.cs.purdue.edu/homes/sgangam/files/Pegasus.pdf>.
- [25] G. Huang, A. Lall, C. Chuah, and J. Xu. Uncovering global icebergs in distributed streams: Results and implications. *J. Netw. Syst. Manage.*, 19:84–110, March 2011.
- [26] L. Huang, X. Nguyen, M. Garofalakis, and J. M. Hellerstein. Communication-efficient online detection of network-wide anomalies. In *Proc. of INFOCOM*, pages 134–142, 2007.
- [27] M. Huang, A. Bavier, and L. Peterson. Planetflow: maintaining accountability for network services. *SIGOPS Oper. Syst. Rev.*, 40:89–94, January 2006.
- [28] Z. Huang, K. Yi, Y. Liu, and G. Chen. Optimal sampling algorithms for frequency estimation in distributed data. In *Proc. of INFOCOM*, pages 1997–2005, april 2011.
- [29] Z. Huang, K. Yi, and Q. Zhang. Randomized algorithms for tracking distributed count, frequencies, and ranks. In *Proc. of PODS*.
- [30] R. M. Karp, S. Shenker, and C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.*, 28:51–55, March 2003.
- [31] R. Keralapura, G. Cormode, and J. Ramamirtham. Communication-efficient distributed monitoring of thresholded counts. In *Proc. of ACM SIGMOD*, pages 289–300, 2006.
- [32] X. Li, F. Bian, H. Zhang, C. Diot, R. Govindan, W. Hong, and G. Iannaccone. MIND: A distributed multi-dimensional indexing system for network diagnosis. In *Proc. of IEEE INFOCOM*, 2006.
- [33] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *Proc. of ACM SIGMOD*, pages 563–574, 2003.
- [34] O. Papapetrou, M. Garofalakis, and A. Deligiannakis. Sketch-based querying of distributed sliding-window data streams. *Proc. VLDB Endow.*, 5(10):992–1003, June 2012.
- [35] F. Raspall, S. Sallent, and J. Yufera. Shared-state sampling. In *Proc. of IMC*, pages 1–14, 2006.
- [36] V. Sekar, N. Duffield, O. Spatscheck, J. van der Merwe, and H. Zhang. Lads: large-scale automated ddos detection system. In *Proc. of USENIX Annual Technical Conference*, pages 16–16, 2006.
- [37] 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. of NSDI*, pages 233–246, 2008.
- [38] K. Yi and Q. Zhang. Optimal tracking of distributed heavy hitters and quantiles. In *Proc. of PODS*, pages 167–174, 2009.
- [39] H. Yu, H.-G. Li, P. Wu, D. Agrawal, and A. El Abbadi. Efficient processing of distributed top-k queries. In *Proc. of DEXA*, pages 65–74, 2005.
- [40] L. Yuan, C. Chuah, and P. Mohapatra. ProgME: towards programmable network measurement. In *Proc. of ACM SIGCOMM*, pages 97–108, 2007.
- [41] H. Zhao, A. Lall, M. Ogihara, and J. Xu. Global iceberg detection over distributed data streams. In *Proc. of ICDE*, 2010.
- [42] Q. G. Zhao, M. Ogihara, H. Wang, and J. J. Xu. Finding global icebergs over distributed data sets. In *Proc. of PODS*, pages 298–307, 2006.