# CLEF: Limiting the Damage Caused by Large Flows in the Internet Core

Hao Wu[1,2][0000-0002-5100-1519], Hsu-Chun Hsiao[3][0000-0001-9592-6911],
Daniele E. Asoni[4][0000-0001-5699-9237],
Simon Scherrer[4][0000-0001-9557-1700], Adrian Perrig[4][0000-0002-5280-5412], and
Yih-Chun Hu[1][0000-0002-7829-3929]

[1] University of Illinois at Urbana Champaign
[2] Rubrik, Inc.
[3] National Taiwan University
[4] ETH Zurich

**Abstract.** The detection of network flows that send excessive amounts of traffic is of increasing importance to enforce QoS and to counter DDoS attacks. Large-flow detection has been previously explored, but the proposed approaches can be used on high-capacity core routers only at the cost of significantly reduced accuracy, due to their otherwise too high memory and processing overhead. We propose CLEF, a new large-flow detection scheme with low memory requirements, which maintains high accuracy under the strict conditions of high-capacity core routers. We compare our scheme with previous proposals through extensive theoretical analysis, and with an evaluation based on worst-case-scenario attack traffic. We show that CLEF outperforms previously proposed systems in settings with limited memory.

**Keywords:** Large-flow detection, damage metric, memory and computation efficiency

## 1 Introduction

Detecting misbehaving large network flows[5] that use more than their allocated resources is not only an important mechanism for Quality of Service (QoS) [29] schemes such as IntServ [5], but also for DDoS defense mechanisms that allocate bandwidth to network flows [4, 19, 23]. With the recent resurgence of volumetric DDoS attacks [3], the topics of DDoS defense mechanisms and QoS are gaining importance; thus, the need for efficient in-network accounting is increasing.

Unfortunately, per-flow resource accounting is too expensive to perform in the core of the network [12], since large-scale Internet core routers have an aggregate capacity of several Terabits per second (Tbps). Instead, to detect misbehaving flows, core routers need to employ highly efficient schemes which do not require

[5] As in prior literature [12, 34], the term *large flow* denotes a flow that sends more than its allocated bandwidth.

them to keep per-flow state. Several approaches for large-flow detection have been proposed in this context; they can be categorized into probabilistic (i.e., relying on random sampling or random binning) and deterministic algorithms. Examples of probabilistic algorithms are Sampled Netflow [8] and Multistage Filters [11,12], while EARDet [34] and Space Saving [25] are examples of deterministic approaches.

However, previously proposed algorithms are able to satisfy the requirements of core router environments only by significantly sacrificing their accuracy. In particular, with the constraints on the amount of high-speed memory on core routers, these algorithms either can only detect flows which exceed their assigned bandwidth by very large amounts, or else they suffer from high false-positive rates. This means that these systems cannot prevent the performance degradation of regular, well-behaved flows, because of large flows that manage to stay "under the radar" of the detection algorithms, or because the detection algorithms themselves erroneously flag and punish the well-behaved flows.

As a numeric example, consider that for EARDet to accurately detect misbehaving flows exceeding a threshold of 1 Mbps on a 100 Gbps link, it would require $10^5$ counters for that link. Maintaining these counters, together with the necessary associated metadata, requires between 1.6 MB and 4MB of state[6], which exceeds typical high-speed memory provisioning for core routers, and would come at a high cost (for comparison, note that only the most high-end commodity CPUs approach the 1–4 MB range with their per-core L1/L2 memory, and the price tag for such processors surpasses USD 4000 [15]).

In this paper we propose a novel randomized algorithm for large flow detection called *Recursive Large-Flow Detection* (RLFD). RLFD works by considering a set of potential large flows, dividing this set into multiple subsets, and then recursively narrowing down the focus to the most promising subset. This algorithm is highly memory efficient, and is designed to have no false positives. To achieve these properties, RLFD sacrifices some detection speed, in particular for the case of multiple concurrent large flows. We improve on these limitations by combining RLFD with the deterministic EARDet, proposing a hybrid scheme called CLEF, short for *in-Core Limiting of Egregious Flows*. We show how this scheme inherits the strengths of both algorithms: the ability to quickly detect very large flows of EARDet (which it can do in a memory efficient way), and the ability to detect low-rate large flows with minimal memory footprint of RLFD.

To have a significant comparison with related work, we define a *damage* metric which estimates the impact of failed, delayed, and incorrect detection on well-behaved flows. We use this metric to compare RLFD and CLEF with previous proposals, which we do both on a theoretical level and by evaluating the amount of damage caused by (worst-case) attacks. Our evaluation shows that CLEF performs better than previous work under realistic memory constraints,

[6] The IP metadata consists of source and destination addresses, protocol number, and ports. Thus, it requires about 16 bytes and 40 bytes per counter for IPv4 and IPv6, respectively.

both in terms of our damage metric and in terms of false negatives and false positives.

To summarize, this paper's main contributions are the following: a novel, randomized algorithm, RLFD, that provides eventual detection of persistently large flows with very little memory cost; a hybrid detection scheme, CLEF, which offers excellent large-flow detection properties with low resource requirements; the analysis of worst-case attacks against the proposed large-flow detectors, using a damage metric that allows a realistic comparison with the related work.

## 2 Problem Definition

This paper aims to design an efficient large-flow detection algorithm that minimizes the *damage* caused by misbehaving flows. This section introduces the challenges of large-flow detection and defines a damage metric to compare different large-flow detectors. We then define an adversary model in which the adversary adapts its behavior to the detection algorithm in use.

### 2.1 Large-Flow Detection

A flow is a collection of related traffic; for example, Internet flows are commonly characterized by a 5-tuple (source / destination IP / port, transport protocol). A *large flow* is one that exceeds a flow specification during a period of length $t$. A flow specification can be defined using a leaky bucket descriptor $\mathsf{TH}(t) = \gamma t + \beta$, where $\gamma > 0$ and $\beta > 0$ are the maximum legitimate rate and burstiness allowance, respectively. Flow specifications can be enforced in two ways: *arbitrary-window*, in which the flow specification is enforced over every possible starting time, or *landmark-window*, in which the flow specification is enforced over a limited set of starting times.

Detecting every large flow exactly when it exceeds the flow specification, and doing so with no false positives requires per-flow state (this can be shown by the pigeonhole principle [32]), which is expensive on core routers. In this paper, we develop and evaluate schemes that trade timely detection for space efficiency.

As in prior work in flow monitoring, we assume each flow has a unique and unforgeable flow ID, e.g., using source authentication techniques such as accountable IPs [1], ICING [27], IPA [20], OPT [17], or with Passport [21]. Such techniques can be deployed in the current Internet or in a future Internet architecture, e.g., Nebula [2], SCION [36], or XIA [14].

**Large-flow detection by core routers.** In this work, we aim to design a large-flow detection algorithm that is viable to run on Internet core routers. The algorithm needs to limit damage caused by large flows even when handling worst-case background traffic. Such an algorithm must satisfy these three requirements:

- **Line rate:** An in-core large-flow detection algorithm must operate at the line rate of core routers, which can process several hundreds of gigabits of traffic per second.

- **Low memory:** Large-flow detection algorithms will typically access one or more memory locations for each traversing packet; such memory must be high-speed (such as on-chip L1 cache). Additionally, such memory is expensive and usually limited in size, and existing large-flow detectors are inadequate to operate in high-bandwidth, low-memory environments. An in-core large-flow detection algorithm should thus be highly space-efficient. Though perfect detection requires counters equal to the maximum number of simultaneous large flows (by the pigeonhole principle [32]), our goal is to perform effective detection with much fewer counters.
- **Low damage:** With the performance constraints of the previous two points, the large-flow detection algorithm should also minimize the damage to honest flows, which can be caused either by the excessive bandwidth usage by large flows, or by the erroneous classification of legitimate flows as large flows (false positives). Section 2.2 introduces our damage metric, which takes both these aspects into account.

### 2.2 Damage Metric

We consider misbehaving large flows to be a problem mainly in that they have an adverse impact on honest flows. To measure the performance of large flow detection algorithms we therefore adopt a simple and effective *damage* metric which captures the packet loss suffered by honest flows. This metric considers both (1) the direct impact of excessive bandwidth usage by large flows, and (2) the potential adverse effect of the detection algorithm itself, which may be prone to false positives resulting in the blacklisting of honest flows. Specifically, we define our damage metric as $D = D_{over} + D_{fp}$, where $D_{over}$ (*overuse damage*) is the total amount of traffic by which all large flows exceed the flow specification, and $D_{fp}$ (*false positive damage*) is the amount of legitimate traffic incorrectly blocked by the detection algorithm. The definition of the overuse damage assumes a link at full capacity, so when this is not the case the damage metric represents an over-approximation of the actual traffic lost suffered by honest flows. We note that the metrics commonly used by previous work, i.e., false positives, false negatives, and detection delay, are all reflected by our metric.

### 2.3 Attacker Model

In our attacker model, we consider an adversary that aims to maximize damage. Our attacker responds to the detection algorithm and tries to exploit its transient behavior to avoid detection or to cause false detection of legitimate flows.

Like Estan and Varghese's work [12], we assume that attackers know about the large-flow detection algorithm running in the router and its settings, but have no knowledge of secret seeds used to generate random variables, such as the detection intervals for landmark-window-based algorithms [9, 10, 12, 13, 16, 24–26], and random numbers used for packet/flow sampling [12]. This assumption prevents the attacker from performing optimal attacks against randomized algorithms.

We assume the attacker can interleave packets, but is unable to spoof legitimate packets (as discussed in Section 2.1) or create pre-router losses in legitimate flows. Figure 1 shows the network model, where the attacker arbitrarily interleaves attack traffic ($A$) between idle intervals of legitimate traffic ($L$), and the router processes the interleaved traffic to generate output traffic ($O$) and perform large-flow detection. Our model does not limit input traffic, allowing for arbitrary volumes of attack traffic.

In our model, whenever a packet traverses a router, the large-flow detector receives the flow ID (for example, the source and destination IP and port and transport protocol), the packet size, and the timestamp at which the packet arrived.

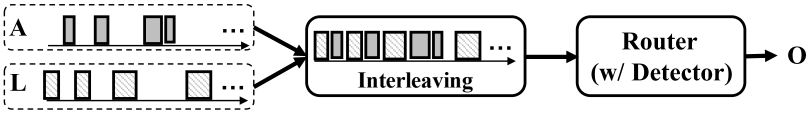


*Fig. 1: Adversary Model.*

## 3 Background and Challenges

In this section we briefly review some existing large flow detection algorithms, and discuss the motivations and challenges of combining multiple algorithms into a hybrid scheme.

### 3.1 Existing Detection Algorithms

We review the three most relevant large-flow detection algorithms, summarized in Table 1. We divide large flows into *low-rate large flows* and *high-rate large flows*, depending on the amount by which they exceed the flow specification.

**EARDet.** EARDet [34] guarantees exact and instant detection of all flows exceeding a high-rate threshold $\gamma_h = \frac{\rho}{m}$, where $\rho$ is the link capacity and $m$ is the number of counters. However, EARDet may fail to identify a large flow whose rate stays below $\gamma_h$.

**Multistage Filters.** Multistage filters [11, 12] consist of multiple parallel stages, each of which is an array of counters. Specifically, *arbitrary-window-based Multistage Filter* (AMF), as classified by Wu et al. [34], uses leaky buckets as counters. AMF guarantees the absence of false negatives (no-FN) and immediate detection for any flow specification; however, AMF has false positives (FPs), which increase as the link becomes congested.

**Flow Memory.** Flow Memory (FM) [12] refers to per-flow monitoring of select flows. FM is often used in conjunction with another system that specifies which flows to monitor; when a new flow is to be monitored but the flow memory is full, FM evicts an old flow. We follow Estan and Varghese [12]'s random eviction. If the flow memory is large enough to avoid eviction, it provides exact detection. In practice, however, Flow Memory is unable to handle a large number of flows, resulting in frequent flow eviction and potentially high FN. FM's real-world performance depends on the amount by which a large flow exceeds the flow specification: high-rate flows are more quickly detected, which improves the chance of detection before eviction.

*Table 1: Comparison of three existing detection algorithms. None of them achieve all desired properties.*

| Algorithm | | EARDet | AMF | FM |
|---|---|---|---|---|
| No-FP | | yes | no* | yes |
| No-FN | low-rate | no** | yes | no* |
| | high-rate | yes | yes | yes*** |
| Instant detection | | yes | yes | yes |

*In our technical report [33] we show that Flow Memory has high FN and AMF has high FP for low-rate large flows when memory is limited.

**EARDet cannot provide no-FN when memory is limited.

***Flow Memory has nearly zero FN when large-flow rate is high.

### 3.2 Advantages of Hybrid Schemes

As Table 1 shows, none of the detectors we examined can efficiently achieve no-FN and no-FP across various types of large flows. However, different detectors exhibit different strengths, so combining them could result in improved performance.

One approach is to run detectors sequentially; in this composition, the first detector monitors all traffic and sends any large flows it detects to a second detector. However, this approach allows an attacker controlling multiple flows to rotate overuse among many flows, overusing a flow only for as long as it takes the first detector to react, then sending at the normal rate so that remaining detectors remove it from their watch list and re-starting with the attack.

Alternatively, we can run detectors in parallel: the hybrid detects a flow whenever it is identified by either detector. (Another configuration is that a flow is only detected if both detectors identify it, but such a configuration would have a high FN rate compared to the detectors used in this paper.) The hybrid inherits the FPs of both schemes, but features the minimum detection delay of the two schemes and has a FN only when both schemes have a FN. The remainder of this paper considers the parallel approach that identifies a flow whenever it is detected by either detector.

The EARDet and Flow Memory schemes have no FPs and are able to quickly detect high-rate flows; because high-rate flows cause damage much more quickly, rapid detection of high-rate flows is important to achieving low damage. Combining EARDet or Flow Memory with a scheme capable of detecting low-rate flows as a hybrid detection scheme can retain rapid detection of high-rate flows while eventually catching (and thus limiting the damage of) low-rate flows. In this paper, we aim to construct such a scheme. Specifically, our scheme will selectively monitor one small set at a time, ensuring that a consistently-overusing flow is eventually detected.

## 4 RLFD and CLEF Hybrid Schemes

In this section, we present our new large-flow detectors. First, we describe the *Recursive Large-Flow Detection* (RLFD) algorithm, a novel approach which is designed to use very little memory but provide eventual detection for large flows. We then present the data structures, runtime analysis, and advantages and disadvantages of RLFD. Next, we develop a hybrid detector, CLEF, that addresses the disadvantages of RLFD by combining it with the previously proposed EARDet [34]. CLEF uses EARDet to rapidly detect high-rate flows and RLFD to detect low-rate flows, thus limiting the damage caused by large flows, even with a very limited amount of memory.

### 4.1 RLFD Algorithm

RLFD is a randomized algorithm designed to perform memory-efficient detection of low-rate large flows; it is designed to scale to a large number of flows, as encountered by an Internet core router. RLFD is designed to limit the damage inflicted by low-rate large flows while using very limited memory. The intuition behind RLFD is to monitor subsets of flows, recursively subdividing the subset deemed most likely to contain a large flow. By dividing subsets in this way, RLFD exponentially reduces memory requirements (it can monitor $m^d$ flows with $O(m + d)$ memory).

The main challenges addressed by RLFD include efficiently mapping flows into recursively divided groups, choosing the correct subdivision to reduce detection delay and FNs, and configuring RLFD to guarantee the absence of FPs.

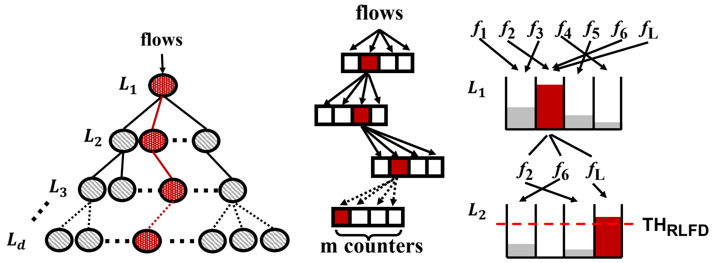
**Recursive subdivision.** To operate with limited memory, RLFD recursively subdivides monitored flows into $m$ groups, and subdivides only the one group most likely to contain a large flow.

We can depict an RLFD as a *virtual counter tree*[7] (Figure 2(a)) of depth $d$. Every non-leaf node in this tree has $m$ children, each of which corresponds to a *virtual counter*. The tree is a full $m$-ary tree of depth $d$, though at any moment, only one node ($m$ counters) is kept in memory; the rest of the tree exists only virtually.

Each flow $f$ is randomly assigned to a path $PATH(f)$ of counters on the virtual tree, as illustrated by the highlighted counters in Figure 2(b). This mapping

[7] The terms “counter tree” and “virtual counter” are also used by Chen et al. [7], but our technique differs in both approach and goal. Chen et al. efficiently manage a sufficient number of counters for per-flow accounting, while RLFD manages an insufficient number of counters to detect consistent overuse.

is determined by hashing a flow ID with a keyed hash function, where the key is randomly generated by each router. Our technical report [33] explains how RLFD efficiently implements this random mapping.



(a) Virtual Counter Tree (Full $m$-branch Tree)

(b) A Tree Branch.

(c) Example with 7 flows, $m = 4$, and $d = 2$.

*Fig. 2: RLFD Structure and Example.*

Since there are $d$ levels, each leaf node at level $L_d$ will contain an average of $n/m^{d-1}$ flows, where $n$ is the total number of flows on the link. A flow $f$ is identified as a large flow if it is the only flow associated with its counter at level $L_d$ and the counter value exceeds a threshold $TH_{RLFD}$. To reflect the flow specification $\mathsf{TH}(t) = \gamma t + \beta$ from Section 2.1, we set $TH_{RLFD} = \gamma T_\ell + \beta$, where $T_\ell$ is the duration of the period during which detection is performed at the bottom level $L_d$. Any flow sending more traffic than $TH_{RLFD}$ during any duration of time $T_\ell$ must violate the large-flow threshold $\mathsf{TH}(t)$, so RLFD has no FPs. We provide more details about how we balance detection rate and the no-FP guarantee in our technical report [33].

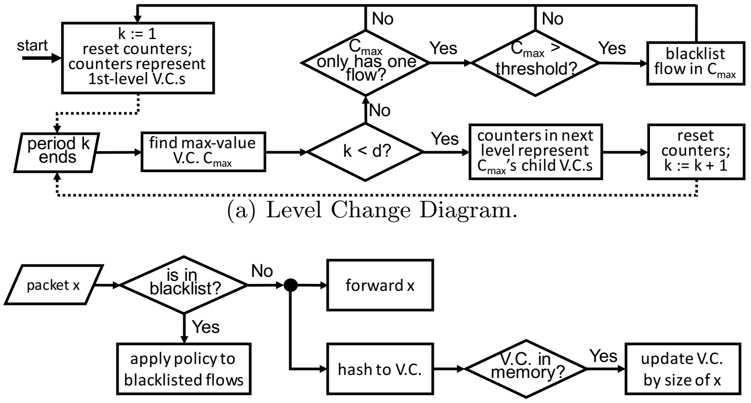
RLFD considers only one node in the virtual counter tree at a time, so it requires only $m$ counters. To enable exploration of the entire tree, RLFD divides the process into $d$ periods; in period $k$, it loads one tree node from level $L_k$. Though these periods need not be of equal length, in this paper we consider periods of equal length $T_\ell$, which results in a RLFD detection cycle $T_c = d \cdot T_\ell$.

RLFD always chooses the root node to monitor at level $L_1$; after monitoring at level $L_k$, RLFD identifies the largest counter $C_{max}$ among the $m$ counters at level $L_k$, and uses the node corresponding to that counter for level $L_{k+1}$. Our technical report [33] shows that choosing the largest counter detects large flows with high probability.

Figure 2(c) shows an example with $m = 4$ counters, $n = 7$ flows, and $d = 2$ levels. $f_L$ is a low-rate large flow. In level $L_1$, the largest counter is the one associated with large flow $f_L$ and legitimate flows $f_2$ and $f_6$. At level $L_2$, the flow set $\{f_L, f_2, f_6\}$ is selected and sub-divided. After the second round, $f_L$ is detected because it violates the counter value threshold $TH_{RLFD}$.

**Algorithm description.** As shown in Figure 3(a), the algorithm starts at the top level $L_1$ so each counter represents a child of the root node. At the beginning of each period, all counters are reset to zero. At the end of each period, the algorithm finds the counter holding the maximum value and moves to the corresponding node, so each counter in the next period is a child of that node. Once the algorithm has processed level $d$, it repeats from the first level.

Figure 3(b) describes how RLFD processes each incoming packet. When RLFD receives a packet $x$ from flow $f$, $x$ is dropped if $f$ is in the *blacklist* (a table that stores previously-found large flows). If $f$ is not in the blacklist, RLFD hashes $f$ to the corresponding counters in the virtual counter tree (one counter per level of the tree). If one such counter is currently loaded in memory, its value is increased by the size of the packet $x$. At the bottom level $L_d$, a large flow is identified when there is only one flow in the counter and the counter value exceeds the threshold $TH_{RLFD}$. To increase the probability that a large flow is in a counter by itself, we choose $d \geq \lceil \log_m n \rceil$ and use Cuckoo hashing [28] at the bottom level to reduce collisions. Once a large flow is identified, it is blacklisted: in our evaluation we calculate the damage $D$ with the assumption that large flows are blocked immediately after having been added to the blacklist.



(a) Level Change Diagram.

(b) Packet Processing Diagram.

*Fig. 3: RLFD Decision Diagrams. "V.C." stands for virtual counter.*

### 4.2 RLFD Details and Optimization

We describe some of the details of RLFD and propose additional optimizations to the basic RLFD described in Section 4.1.

**Hash function update.** We update the keyed hash function by choosing a new key at the beginning of every initial level to guarantee that the assignment of flows to counters between different top-to-bottom detection cycles is independent and pseudo-random. For simplicity, in this paper we analyze RLFD assuming the random oracle model. Picking a new key is computationally inexpensive and needs to be performed only once per cycle.

**Blacklist.** When RLFD identifies a large flow, the flow's ID should be added to the blacklist as quickly as possible. Thus, we implement the blacklist with a small amount of L1 cache backed by permanent storage, e.g., main memory.

Because the blacklist write only happens at the bottom-level period and the number of large flows detected in one iteration of the algorithm is at most one, we first write these large flows in the L1 cache and move them from L1 cache to permanent storage at a slower rate. By managing the blacklist in this way, we provide high bandwidth for blacklist writing, defending against attacks that overflow the blacklist.

**Using multiple RLFDs.** If a link handles too much traffic to use a single RLFD, we can use multiple RLFDs in parallel. Each flow is hashed to a specific RLFD so that the load on each detector meets performance requirements. The memory requirements scale linearly in the number of RLFDs required to process the traffic.

### 4.3 RLFD's Advantages and Disadvantages

**Advantages.** With recursive subdivision and additional optimization techniques, RLFD is able to (1) identify low-rate large flows with non-zero probability, with probability close to 100% for flows that cause extensive damage (our technical report [33] analyzes RLFD's detection probability); and (2) guarantee no-FP, eliminating damage due to FP.

**Disadvantages.** First, a landmark-window-based algorithm such as RLFD cannot guarantee exact detection over large-flow specification based on arbitrary time windows [34] (landmark window and arbitrary window are introduced in Section 2.1). However, this approximation results in limited damage, as mentioned in Section 3. Second, recursive subdivision based on landmark time windows requires at least one detection cycle to catch a large flow. Thus, RLFD cannot guarantee low damage for flows with very high rates. Third, RLFD works most effectively when the large flow exceeds the flow specification in all $d$ levels, so bursty flows with a burst duration shorter than the RLFD detection cycle $T_c$ are likely to escape detection (where *burst duration* refers to the amount of time during which the bursty flow sends in excess of the flow specification).

### 4.4 CLEF Hybrid Scheme

We propose a hybrid scheme, CLEF, which is a parallel composition with one EARDet and two RLFDs (Twin-RLFD). This hybrid can detect both high-rate and low-rate large flows without producing FPs, requiring only a limited amount of memory. We use EARDet instead of Flow Memory in this hybrid scheme because EARDet's detection is deterministic, thus has shorter detection delay.

**Parallel composition of EARDet and RLFD.** As described in Section 3.2, we combine EARDet and RLFD in parallel so that RLFD can help EARDet detect low-rate flat flows, and EARDet can help RLFD quickly catch high-rate flat and bursty flows.

**Twin-RLFD parallel composition.** RLFD is most effective at catching flows that violate flow specification across an entire detection cycle $T_c$. An attacker can reduce the probability of being caught by RLFD by choosing a burst duration shorter than $T_c$ and an inter-burst duration greater than $T_c/d$ (thus reducing the probability that the attacker will advance to the next round during its inter-burst period). We therefore introduce a second RLFD (RLFD$^{(2)}$) with a longer detection cycle $T_c^{(2)}$, so that a flow must have burst duration shorter than $T_c^{(1)}$ and burst period longer than $T_c^{(2)}/d$ to avoid detection by the Twin-RLFD (where RLFD$^{(1)}$ and $T_c^{(1)}$, are the first RLFD and its detection cycle respectively). For a given average rate, flows that evade Twin-RLFD have a higher burst rate than flows that evade a single RLFD. By properly setting $T_c^{(1)}$ and $T_c^{(2)}$, Twin-RLFD can synergize with EARDet, ensuring that a flow undetectable by Twin-RLFD must use a burst higher than EARDet's rate threshold $\gamma_h$.

**Timing randomization.** An attacker can strategically send traffic with burst durations shorter than $T_c^{(1)}$, but choose low duty cycles to avoid detection by both RLFD$^{(1)}$ and EARDet. Such an attacker can only be detected by RLFD$^{(2)}$, but RLFD$^{(2)}$ has a longer detection delay, allowing the attacker to maximize damage before being blacklisted. To prevent attackers from deterministically maximizing damage, we randomize the length of the detection cycles $T_c^{(1)}$ and $T_c^{(2)}$.

## 5 Evaluation

We experimentally evaluate CLEF, RLFD, EARDet, and AMF-FM with respect to worst-case damage [33, Sec. 5.1]. We consider various large-flow patterns and memory limits and assume background traffic that is challenging for CLEF and RLFD. The experiment results confirm that CLEF outperforms other schemes, especially when memory is extremely limited.

### 5.1 Experiment Settings

**Link settings.** Since the required memory space of a large-flow detector is sublinear to link capacity, we set the link capacity to $\rho = 1$Gbps, which is high enough to incorporate the realistic background traffic dataset while ensuring the simulation can finish in reasonable time. We choose a very low threshold rate $\gamma = 12.5$ KB/s, so that the number of full-use legitimate flows $n_\gamma = \rho/\gamma$ is 10000, ensuring that the link is as challenging as a backbone link (as analyzed in our technical report [33]). The flow specification is set to $\mathsf{TH}(t) = \gamma t + \beta$, where $\beta$ is set to 3028 bytes (which is as small as two maximum-sized packets, making bursty flows easier to catch).

The results on this 1Gbps link allow us to extrapolate detector performance to high-capacity core routers, e.g., in a 100Gbps link with $\gamma = 1.25$ MB/s. Because CLEF's performance with a given number of counters is mainly related to the ratio between link capacity and threshold rate $n_\gamma$ (as discussed in our technical report [33]), CLEF's worst-case performance will scale linearly in link capacity when the number of counters and the ratio between link capacity and threshold rate is held constant. AMF-FM, on the other hand, performs worse as the number of flows increases (according to our technical report [33]). Thus, with increasing link capacity, AMF-FM may face an increased number of actual flows, resulting in worse performance. In other words, AMF-FM's worst-case damage may be superlinear in link capacity. As a result, if CLEF outperforms AMF-FM in small links, CLEF will outperform AMF-FM by at least as large a ratio in larger links.

**Background traffic.** We consider the worst background traffic for RLFD and CLEF: we determine the worst-case traffic in our technical report [33, Thm. 1]. Aside from attack traffic, the rest of the link capacity is completely filled with full-use legitimate flows running at the threshold rate $\gamma = 12.5$ KB/s. The total number of attack flows and full-use legitimate flows is $n_\gamma = 10000$. Once a flow has been blacklisted by the large-flow detectors, we fill the idle bandwidth with a new full-use legitimate flow, to keep the link always running with the worst-case background traffic.

**Attack traffic.** We evaluate each detector against large flows with various average rates $R_{atk}$ and duty cycle $\theta$. Their bursty period is set to be $T_b = 0.967$s. To evaluate RLFD and CLEF against their worst-case bursty flows ($\theta T_b < 2T_c$), large flows are allotted a relatively small bursty period $T_b = 4T_\ell = 0.967$s, where $T_\ell = \beta/\gamma = 0.242$s is the period of each detection level in the single RLFD. In CLEF, RLFD$^{(1)}$ uses the same detection level period $T_\ell^{(1)} = T_\ell = 0.242$s as well. Since RLFD usually has $d \geq 3$ levels and $T_c \geq 3T_\ell$, it is easy for attack flows to meet $\theta T_b < 2T_c$.

In each experiment, we have 10 artificial large flows whose rates are in the range of 12.5 KB/s to 12.5 MB/s (namely, 1 to 1000 times that of threshold rate $\gamma$). The fewer large flows in the link, the longer delay required for RLFD and CLEF to catch large flows; however, the easier it is for AMF-FM to detect large flows, because there are fewer FPs from AMF and more frequent flow eviction in FM. Thus, we use 10 attack flows to challenge CLEF and the results are generalizable.

**Detector settings** We evaluate detectors with different numbers of counters ($20 \leq m \leq 400$) to understand their performance under different memory limits. Although a few thousands of counters are available in a typical CPU, not all can be used by one detector scheme. CLEF works reasonably well with such a small number of counters and can perform better when more counters are available.

- ***EARDet***. We set the low-bandwidth threshold to be the flow specification $\gamma t + \beta$, and compute the corresponding high-rate threshold, $\gamma_h = \frac{\rho}{m+1}$, for $m$ counters as in [34].
- ***RLFD***. A RLFD has $d$ levels and $m$ counters. We set the period of a detection level as $T_\ell = \beta/\gamma = 0.242$ seconds[8]. $d = \lfloor 1.2 \times \log_m(n) \rfloor + 1$ to have fewer flows than the counters at the bottom level. The counter threshold of the bottom level is $TH_{RLFD} = \gamma T_\ell + \beta = 2\beta = 6056$ Bytes.

[8] If $T_\ell \ll \beta/\gamma$, it is hard for a large flow to reach the burst threshold $\beta$ in such a short time; if $T_\ell \gg \beta/\gamma$, the detection delay is too long, resulting in excessive damage.

- ***CLEF***. We allocate $m/2$ counters to EARDet, and $m/4$ counters to each RLFD. RLFD$^{(1)}$ and EARDet are configured like the single RLFD and the single EARDet above. For the RLFD$^{(2)}$, we properly set its detection level period $T_\ell^{(2)}$ to guarantee detection of most of bursty flows with low damage. The details of the single RLFD and CLEF are in our technical report [33].
- ***AMF-FM***. We allocate half of the $m$ counters to AMF and the rest to FM. AMF has four stages (a typical setting in [12]), each of which contains $m/8$ counters. All $m$ counters are leaky buckets with a drain rate of $\gamma$ and a bucket size $\beta$.

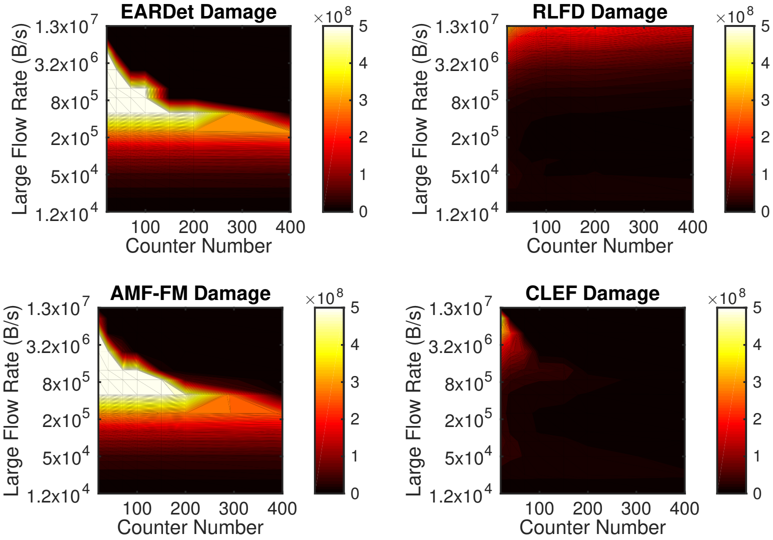
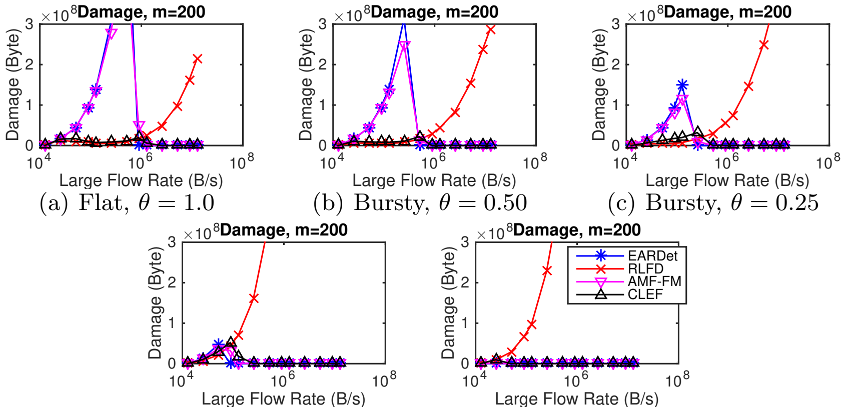


*Fig. 4: Damage (in Bytes) caused by* 200*-second large flows at different average flow rate $R_{atk}$ (in Byte/s) and duty cycle $\theta = 1.0$ (flat large flows) under detection of different schemes with different number of counters $m$. The larger the dark area, the lower the damage guaranteed by a scheme. Areas with white color are damage equals or exceeds $5 \times 10^8$. Figures of bursty flows are shown in our technical report [33]. CLEF outperforms other schemes in detecting flat flows, and has competitive performance to AMF-FM and EARDet over bursty flows.*

### 5.2 Experiment Results

For each experiment setting (i.e., attack flow configurations and detector settings), we did 50 repeated runs and present the averaged results.

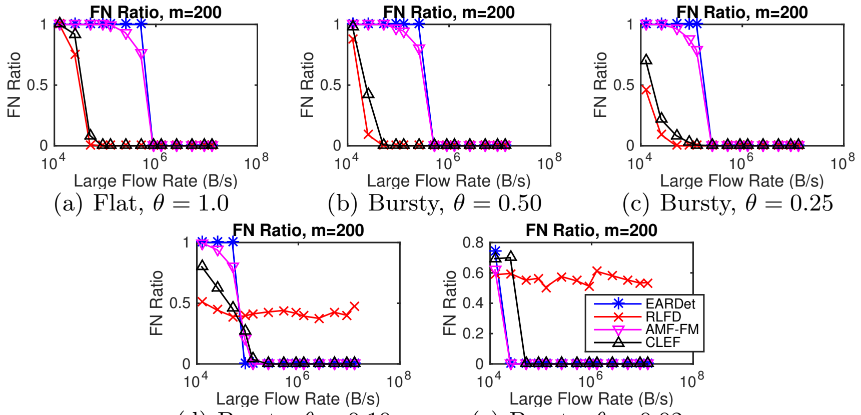
Figure 4 demonstrates the damage caused by large flows at different average rates, duty cycles, and number of detector counters during 200-second experiments; the lighter the color, the higher the damage. The damage $\geq 5 \times 10^8$ Byte is represented by the color white. Figures 5(a) to 5(e) compare damage in cases



(a) Flat, $\theta = 1.0$ (b) Bursty, $\theta = 0.50$ (c) Bursty, $\theta = 0.25$

(d) Bursty, $\theta = 0.10$ (e) Bursty, $\theta = 0.02$

*Fig. 5: Damage (in Bytes) caused by 200-second large flows at different average rate $R_{atk}$ (in Byte/s) and duty cycle $\theta$. Each detection scheme uses 200 counters in total. The clear comparison among schemes suggests CLEF outperforms others with low damage against various large flows.*



(a) Flat, $\theta = 1.0$ (b) Bursty, $\theta = 0.50$ (c) Bursty, $\theta = 0.25$

(d) Bursty, $\theta = 0.10$ (e) Bursty, $\theta = 0.02$

*Fig. 6: FN ratio in a 200-second detection for large flows at different average rate $R_{atk}$ (in Byte/s) and duty cycle $\theta$. Each detection scheme uses 200 counters in total. CLEF is able to detect (FN$<$ 1.0) low-rate flows undetectable (FN$=$ 1.0) by AMF-FM or EARDet.*

of different detectors with 200 counters. Figures 6(a) to 6(e) show the percentage of FNs produced by each detection scheme with 200 counters within 200 seconds. We cannot run infinitely-long experiments to show the $+\infty$ damage produced by detectors like EARDet and AMF-FM over low-rate flows, so we use the FN ratio to suggest it here. An FN of 1.0 means that the detector fails to identify any large flow in 200 seconds and is likely to miss large flows in the future. Thus, an infinite damage is assigned. On the contrary, if a detector has FN rate $< 1.0$, it is able to detect remaining large flows at some point in the future.

**CLEF ensures low damage against flat flows.** Figures 4, 5(a), and 6(a) show that RLFD and CLEF work effectively at detecting low-rate flat large flows and guaranteeing low damage. On the contrary, such flows cause much higher damage against EARDet and AMF-FM. The nearly-black figure (in Figure 4) for CLEF shows that CLEF is effective for both high-rate and low-rate flat flows with different memory limits. Figure 5(a) shows a clear damage comparison among detector schemes. CLEF, EARDet, and AMF-FM all limit the damage to nearly zero for high-rate flat flows. However, the damage limited by CLEF is much lower than that limited by AMF-FM and EARDet for the low-rate flat flows. EARDet and AMF-FM results show a sharp top boundary that reflects the damage dropping to zero at the guaranteed-detection rates.

The damage limited by an individual RLFD is proportional to the large-flow rate when the flow rate is high. Figure 6(a) suggests that AMF-FM and EARDet are unable to catch most low-rate flat flows ($R_{atk} < 10^6$ Byte/sec), which explains the high damage by low-rate flat flows against these two schemes.

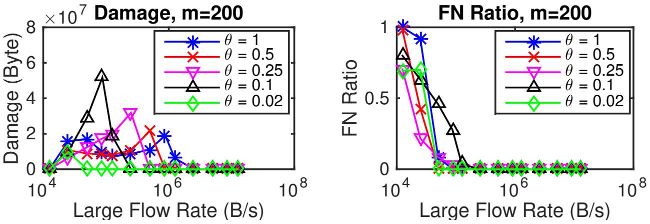
**CLEF ensures low damage against various bursty flows.** Figures 5(b) to 5(e) demonstrate the damage caused by bursty flows with different duty cycle $\theta$. The smaller the $\theta$ is, the burstier the flow. As the large flows become burstier, the EARDet and AMF-FM schemes improve at detecting flows whose average rate is low. Because the rate at the burst is $R_{atk}/\theta$, which increases as $\theta$ decreases, thus EARDet and AMF-FM are able to detect these flows even though their average rates are low. For a single RLFD, the burstier the flows are, the harder it becomes to detect the large flows and limit the damage. As we discussed in Section 4.4, when the burst duration $\theta T_b$ of flows is smaller than the RLFD detection cycle $T_c$, a single RLFD has nearly zero probability of detecting such attack flows. Thus, we need Twin-RLFD in CLEF to detect bursty flows missed by EARDet in CLEF, so that CLEF's damage is still low as the figures show. When the flow is very bursty (e.g., $\theta \leq 0.1$), the damage limitation of the CLEF scheme is dominated by EARDet.

Figures 5(b) to 5(e) present a clear comparison among different schemes against bursty flows. The damage limited by CLEF is lower than that limited by AMF-FM and EARDet, when $\theta$ is not too small (e.g., $\theta \geq 0.25$). Even though AMF-FM and EARDet have lower damage for very bursty flows (e.g., $\theta \leq 0.1$) than the damage limited by CLEF, the results are close because CLEF is assisted by an EARDet with $m/2$ counters. Thus, CLEF guarantees a low damage limit for a wider range of large flows than the other schemes.

**CLEF outperforms others in terms of FN and FP.** To make our comparison more convincing, we examine schemes with classic metrics: FN and FP. Since we know all four schemes have no FP, we simply check the FN ratios in Figures 6(a) to 6(e). Generally, CLEF has a lower FN ratio than AMF-FM and EARDet do. CLEF can detect large flows at a much lower rate with zero FN ratio, and is competitive to AMF-FM and EARDet against very bursty flows (e.g., Figures 6(b) and 6(e)).

**CLEF is memory-efficient.** Figure 4 shows that the damage limited by RLFD is relatively insensitive to the number of counters. This suggests that RLFD can work with limited memory and is scalable to larger links without requiring a large amount of high-speed memory. This can be explained by RLFD's recursive subdivision, by which we simply add one or more levels when the memory limit is low. Thus, we choose RLFD to complement EARDet in CLEF.

In Figure 4, CLEF ensures a low damage (shown in black) with tens of counters, while AMF-FM suffers from a high damage (shown in light colors), even with 400 counters.



(a) Damage (b) FN Ratio

*Fig. 7: Damage and FN ratio for large flows at different average rate $R_{atk}$ (in Byte/s) and duty cycle $\theta$ under detection of CLEF with $m = 200$ counters. CLEF is insensitive to bursty flows across duty cycles: 1) the damages are around the same scale (not keep increasing as duty cycle decrease, because of EARDet), 2) the FN ratios are stable and similar.*

**CLEF is effective against various types of bursty flows.** Figures 7(a) and 7(b) demonstrate the changes of damage and FN ratio versus different duty cycles $\theta$ when CLEF is used to detect bursty flows. In the 200-second evaluation, as $\theta$ decreases, the maximum damage across different average flow rates increases first by ($\theta \geq 0.1$) and then decreases by ($\theta < 0.1$). The damage increases when $\theta \geq 0.1$ because Twin-RLFD (in CLEF) gradually loses its capability to detect bursty flows. The damage therefore increases due to the increase in detection delay.

However, the maximum damage does not increase all the way as $\theta$ decreases, because when $\theta$ is getting smaller, EARDet is able to catch bursty flows with a lower average rate. This explains the lower damage from large flows in the 200-second timeframe. Figure 7(b) shows that the FN ratio curve changes within a small range as $\theta$ decreases, which also indicates the stable performance of CLEF against various bursty flows. Moreover, the FN ratios are all below 1.0, which means that CLEF can eventually catch large flows, whereas EARDet and AMF-FM cannot.

**CLEF operates at high speed.** We also evaluated the performance of a Golang-based implementation under real-world traffic trace from the CAIDA [6] dataset. The implementation is able to process 11.8M packets per second, which is sufficient for a 10 Gbps Ethernet link, which has a capacity of 14.4M packets per second.

## 6 Related Work

The most closely related large-flow detection algorithms are described in Section 3.1 and compared in Section 5 and further in our technical report [33]. This section discusses other related schemes.

**Frequent-item finding.** Algorithms that find frequent items in a stream can be applied to large-flow detection. For example, Lossy Counting [24] maintains a lower bound and an upper bound of each item's count. It saves memory by periodically removing items with an upper bound below a threshold, but loses the ability to catch items close to the threshold. However, the theoretical memory lower bound of one-pass exact detection is linear to the number of large flows, which is unaffordable by in-core routers. By combining a frequent-item finding scheme with RLFD, CLEF can rapidly detect high-rate large flows and confine low-rate large flows using limited memory.

**Collision-rich schemes.** To reduce memory requirement in large-flow utilization, a common technique is hashing flows into a small number of bins. However, hash collisions may cause FPs, and FPs increase as the available memory shrinks. For example, both multistage filters [11, 12] and space-code Bloom filters [18] suffer from high FPs when memory is limited.

**Sampling-based schemes.** Sampling-based schemes estimate the size of a flow based on sampled packets. However, with extremely limited memory and thus a low sampling rate, neither packet sampling (e.g., Sampled Netflow [8]) nor flow sampling (e.g., Sample and Hold [12] and Sticky Sampling [24]) can robustly identify large flows due to insufficient information. In contrast, RLFD in CLEF progressively narrows down the candidate set of large flows, thereby effectively confining the damage caused by large flows.

**Top-k detection.** Top-k heavy hitter algorithms can be used to identify flows that use more than $1/k$ of bandwidth. Space Saving [25] finds the top-k frequent items by evicting the item with the lowest counter value. HashPipe [30] improves upon Space Saving so that it can be practically implemented on switching hardware. However, HashPipe still requires keeping 80KB to detect large flows that use more than 0.3% of link capacity, whereas CLEF can enforce flow specifications as low as $10^{-6}$ of the link capacity using only 10KB of memory. Tong et al. [31] propose an efficient heavy hitter detector implemented on FPGA but the enforceable flow specifications are several orders looser than CLEF. Moreover, misbehaving flows close to the flow specification can easily bypass such heavy hitter detectors. The FPs caused by heavy hitters prevent network operators from applying strong punishment to the detected flows.

Chen et al. [7] and Xiao et al. [35] propose memory-efficient algorithms for estimating per-flow cardinality (e.g., the number of packets). These algorithms, however, cannot guarantee large-flow detection in adversarial environments due to under- or over-estimation of the flow size.

Liu et al. [22] propose a generic network monitoring framework called UniMon that allows extraction of various flow statistics. It creates flow statistics for all flows, but has high FP and FN when used to detect large flows.

## 7 Conclusion

In this paper we propose new efficient large-flow detection algorithms. First, we develop a randomized Recursive Large-Flow Detection (RLFD) scheme, which uses very little memory yet provides eventual detection of persistently large flows. Second, we develop CLEF, which scales to Internet core routers and is resilient against worst-case traffic. None of the prior approaches can achieve the same level of resilience with the same memory limitations. To compare attack resilience among various detectors, we define a damage metric that summarizes the impact of attack traffic on legitimate traffic. CLEF can confine damage even when faced with the worst-case background traffic because it combines a deterministic EARDet for the rapid detection of very large flows and two RLFDs to detect near-threshold large flows. We denomstrated that CLEF is able to guarantee low-damage large-flow detection against various attack flows with limited memory, outperforming other schemes even with CLEF's worst-case background traffic. Further experimental evaluation confirms the findings of our theoretical analysis and shows that CLEF has the lowest worst-case damage among all detectors and consistently low damage over a wide range of attack flows.

## 8 Acknowledgments

We thank Pratyaksh Sharma and Prateesh Goyal for early work on this project as part of their summer internship at ETH in Summer 2015. We also thank the anonymous reviewers, whose feedback helped to improve the paper.

The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013), ERC grant agreement 617605, the Ministry of Science and Technology of Taiwan under grant number MOST 107-2636-E-002-005, and the US National Science Foundation under grant numbers CNS-1717313 and CNS-0953600. We also gratefully acknowledge support from ETH Zurich and from the Zurich Information Security and Privacy Center (ZISC).

## References

1. Andersen, D.G., Balakrishnan, H., Feamster, N., Koponen, T., Moon, D., Shenker, S.: Accountable internet protocol (AIP). In: Proceedings of ACM SIGCOMM (2008). https://doi.org/10.1145/1402958.1402997, http://portal.acm.org/citation.cfm?doid=1402958.1402997

2. Anderson, T., Birman, K., Broberg, R., Caesar, M., Comer, D., Cotton, C., Freedman, M.J., Haeberlen, A., Ives, Z.G., Krishnamurthy, A., et al.: The nebula future internet architecture. In: The Future Internet Assembly. pp. 16–26. Springer (2013)
3. Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, A.J., Invernizzi, L., Kallitsis, M., Kumar, D., Lever, C., Ma, Z., Mason, J., Menscher, D., Seaman, C., Sullivan, N., Thomas, K., Zhou, Y.: Understanding the Mirai botnet. In: USENIX Security Symposium (2017)
4. Basescu, C., Reischuk, R.M., Szalachowski, P., Perrig, A., Zhang, Y., Hsiao, H.C., Kubota, A., Urakawa, J.: SIBRA: Scalable internet bandwidth reservation architecture. In: Proceedings of Network and Distributed System Security Symposium (NDSS) (Feb 2016)
5. Braden, R., Clark, D., Shenker, S.: Integrated Services in the Internet Architecture: an Overview. RFC 1633 (Informational) (Jun 1994), http://www.ietf.org/rfc/rfc1633.txt
6. CAIDA: Caida Anonymized Internet Traces 2016. https://data.caida.org/datasets/passive-2016/ (2016)
7. Chen, M., Chen, S., Cai, Z.: Counter tree: a scalable counter architecture for per-flow traffic measurement. IEEE/ACM Transactions on Networking (2016)
8. Claise, B.: Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational) (Oct 2004), http://www.ietf.org/rfc/rfc3954.txt
9. Cormode, G., Muthukrishnan, S.: An Improved Data Stream Summary: The Count-Min Sketch and its Applications. Journal of Algorithms **55**(1), 58–75 (2005). https://doi.org/10.1016/j.jalgor.2003.12.001, http://linkinghub.elsevier.com/retrieve/pii/S0196677403001913
10. Demaine, E.D., López-Ortiz, A., Munro, J.I.: Frequency Estimation of Internet Packet Streams with Limited Space. In: Proceedings of ESA (2002), http://www.springerlink.com/index/0MJ1EXMY9L9MCQAD.pdf
11. Estan, C.: Internet Traffic Measurement: What's Going on in my Network? Ph.D. thesis (2003)
12. Estan, C., Varghese, G.: New Directions in Traffic Measurement and Accounting: Focusing on the Elephants, Ignoring the Mice. ACM Transactions on Computer Systems (TOCS) **21**(3), 270–313 (2003), http://dl.acm.org/citation.cfm?id=859719
13. Fang, M., Shivakumar, N.: Computing Iceberg Queries Efficiently. In: Proceedings of VLDB (1999), http://ilpubs.stanford.edu:8090/423/
14. Han, D., Anand, A., Dogar, F., Li, B., Lim, H., Machado, M., Mukundan, A., Wu, W., Akella, A., Andersen, D.G., Byers, J.W., Seshan, S., Steenkiste, P.: XIA: Efficient support for evolvable internetworking. In: Proc. 9th USENIX NSDI. San Jose, CA (Apr 2012)
15. Intel: Intel Xeon Processor E7 v4 Family. https://ark.intel.com/products/series/93797/Intel-Xeon-Processor-E7-v4-Family (2016)
16. Karp, R.M., Shenker, S., Papadimitriou, C.H.: A Simple Algorithm for Finding Frequent Elements in Streams and Bags. ACM Transactions on Database Systems **28**(1), 51–55 (2003). https://doi.org/10.1145/762471.762473, http://portal.acm.org/citation.cfm?doid=762471.762473
17. Kim, T.H.J., Basescu, C., Jia, L., Lee, S.B., Hu, Y.C., Perrig, A.: Lightweight source authentication and path validation. In: ACM SIGCOMM Computer Communication Review. vol. 44, pp. 271–282. ACM (2014)
18. Kumar, A., Xu, J., Wang, J.: Space-code bloom filter for efficient per-flow traffic measurement. IEEE Journal on Selected Areas in Communications **24**(12), 2327–2339 (2006)

19. Lee, S.B., Kang, M.S., Gligor, V.D.: CoDef: Collaborative defense against large-scale link-flooding attacks. In: Proceedings of CoNext (2013)
20. Li, A., Liu, X., Yang, X.: Bootstrapping accountability in the Internet we have. In: Proceedings of USENIX/ACM NSDI (Mar 2011)
21. Liu, X., Li, A., Yang, X., Wetherall, D.: Passport: Secure and adoptable source authentication. In: Proceedings of USENIX/ACM NSDI (2008), http://www.usenix.org/event/nsdi08/tech/full_papers/liu_xin/liu_xin_html/
22. Liu, Z., Manousis, A., Vorsanger, G., Sekar, V., Braverman, V.: One Sketch to Rule Them All: Rethinking Network Flow Monitoring with UnivMon. In: ACM SIGCOMM (2016). https://doi.org/10.1145/2934872.2934906
23. Liu, Z., Jin, H., Hu, Y.C., Bailey, M.: MiddlePolice: Toward enforcing destination-defined policies in the middle of the internet. In: Proceedings of ACM CCS (Oct 2016)
24. Manku, G., Motwani, R.: Approximate Frequency Counts over Data Streams. In: Proceedings of VLDB (2002), http://dl.acm.org/citation.cfm?id=1287400
25. Metwally, A., Agrawal, D., El Abbadi, A.: Efficient computation of frequent and top-k elements in data streams. In: International Conference on Database Theory. pp. 398–412. Springer (2005)
26. Misra, J., Gries, D.: Finding Repeated Elements. Science of Computer Programming **2**(2), 143–152 (1982)
27. Naous, J., Walfish, M., Nicolosi, A., Mazires, D., Miller, M., Seehra, A.: Verifying and enforcing network paths with ICING. In: Proceedings of ACM CoNEXT (2011). https://doi.org/10.1145/2079296.2079326
28. Pagh, R., Rodler, F.F.: Cuckoo hashing. In: European Symposium on Algorithms. pp. 121–133. Springer (2001)
29. Shenker, S., Partridge, C., Guerin, R.: Specification of Guaranteed Quality of Service. RFC 2212 (Proposed Standard) (Sep 1997), http://www.ietf.org/rfc/rfc2212.txt
30. Sivaraman, V., Narayana, S., Rottenstreich, O., Muthukrishnan, S., Rexford, J.: Heavy-hitter detection entirely in the data plane. In: Proceedings of the Symposium on SDN Research. pp. 164–176. ACM (2017)
31. Tong, D., Prasanna, V.: High throughput sketch based online heavy hitter detection on fpga. ACM SIGARCH Computer Architecture News **43**(4), 70–75 (2016)
32. Trybulec, W.A.: Pigeon hole principle. Journal of Formalized Mathematics **2**(199), 0 (1990)
33. Wu, H., Hsiao, H.C., Asoni, D.E., Scherrer, S., Perrig, A., Hu, Y.C.: CLEF: Limiting the Damage Caused by Large Flows in the Internet Core (Technical Report). Tech. Rep. arXiv:1807.05652 [cs.NI], ArXiv (2018), https://arxiv.org/abs/1807.05652
34. Wu, H., Hsiao, H.C., Hu, Y.C.: Efficient large flow detection over arbitrary windows: An algorithm exact outside an ambiguity region. In: Proceedings of the 2014 Conference on Internet Measurement Conference. pp. 209–222. ACM (2014)
35. Xiao, Q., Chen, S., Chen, M., Ling, Y.: Hyper-compact virtual estimators for big network data based on register sharing. In: ACM SIGMETRICS Performance Evaluation Review. vol. 43, pp. 417–428. ACM (2015)
36. Zhang, X., Hsiao, H.C., Hasker, G., Chan, H., Perrig, A., Andersen, D.G.: SCION: Scalability, control, and isolation on next-generation networks. In: IEEE Symposium on Security and Privacy. pp. 212–227 (2011)