Efficient Large Flow Detection over Arbitrary Windows: An Algorithm Exact Outside An Ambiguity Region

Hao Wu
University of Illinois at Urbana-Champaign

Hsu-Chun Hsiao
Carnegie Mellon University
National Taiwan University

Yih-Chun Hu
University of Illinois at Urbana-Champaign

ABSTRACT

Many networking and security applications can benefit from *exact* detection of large flows over *arbitrary windows* (i.e. any possible time window). Existing large flow detectors that only check the average throughput over certain time period cannot detect bursty flows and are therefore easily fooled by attackers. However, no scalable approaches provide exact classification in one pass. To address this challenge, we consider a new model of exactness outside an *ambiguity region*, which is defined to be a range of bandwidths below a high-bandwidth threshold and above a low-bandwidth threshold. Given this new model, we propose a deterministic algorithm, EARDet, that detects *all* large flows (including bursty flows) and avoids false accusation against *any* small flows, regardless of the input traffic distribution. EARDet monitors flows over arbitrary time windows and is built on a frequent items finding algorithm based on average frequency. Despite its strong properties, EARDet has low storage overhead regardless of input traffic and is surprisingly scalable because it focuses on accurate classification of large flows and small flows only. Our evaluations confirm that existing approaches suffer from high error rates (e.g., misclassifying 1% of small flows as large flows) in the presence of large flows and bursty flows, whereas EARDet can accurately detect both at gigabit line rate using a small amount of memory that fits into on-chip SRAM.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*Network monitoring*; C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*

Keywords

Large Flow Detection; Flow Classification; Ambiguity Region; Arbitrary Windows

IMC'14, November 5–7, 2014, Vancouver, BC, Canada.

ACM 978-1-4503-3213-2/14/11 ...$15.00.
http://dx.doi.org/10.1145/2663716.2663724.

1. INTRODUCTION

Being able to identify large flows[1] over any possible time window (referred to as the *arbitrary window model*) is of great importance for a wide variety of networking and security applications such as traffic engineering, accounting, anomaly detection, and Denial-of-Service (DoS) defense. The arbitrary window model monitors time windows on all time-scales, starting at each instant in time and ending at the current time, and can thus detect bursty flows and outperform models that are based on fixed-length windows. Given the high traffic volumes in today's Internet, no scalable approaches provide exact classification in one pass.

Thus, we consider a novel model of exactness outside a small *ambiguity region*, which contains flows that use bandwidth between two configurable thresholds. Our model classifies flows as either large, medium, or small. A flow is defined to be a large flow if there exists a time window in which the bandwidth of the flow exceeds the high-bandwidth threshold. A small flow is defined to be a flow whose volume is consistently lower than a low-bandwidth threshold function over all arbitrary window. The rest are defined as medium flows, or the flows in the ambiguity region. Exactness outside an ambiguity region guarantees perfect detection of large flows (including bursty flows) and no false accusation against any small flows. This model is reasonable because it limits the damage caused by large flows and allows existing techniques to handle the medium flows statistically. Prior work [17] [16] adopts a similar concept, yet they can only provide probabilistic bounds outside a region. The ambiguity region between the high-bandwidth and low-bandwidth thresholds allows us to trade the level of exactness for scalability, so that we can maintain state small enough to fit into limited on-chip memory for link-speed update.

The new models of exactness and arbitrary window benefit many applications. For example:

- *Detecting various DoS flows*: Denial-of-Service (DoS) attacks use a combination of large attack flows and bursty attack flows. Flows that are only bursty are hard to catch due to their low average traffic bandwidth. However, using the arbitrary window model, a detector can instantly detect DoS attempts that use bursty flows [25].
- *Bandwidth guarantees*: To enforce bandwidth allocation, schemes such as IntServ make impractical

[1]Large flows are flows that consume more than a threshold amount of bandwidth. They are also called elephant flows in the literature.

assumptions that either every router keeps per-flow state [33] or first-hop routers are trusted to regulate traffic on a per-flow basis [36] on behalf of intermediate routers [34]. Although a scalable and robust approach was proposed [28], it causes collateral damage due to the detection delay. Moreover, it cannot catch bursty flows. Our efficient detector built upon these two new models can help enforce bandwidth limits on flows because it enables fast detection with no false accusation on the legitimate small flows and no missed detection on large flows including bursty flows.

To the best of our knowledge, none of the existing algorithms provide exactness outside an ambiguity region under the arbitrary window model. Prior algorithms monitoring average throughput (e.g., algorithms based on fixed window and sliding window models) cannot detect bursty attacks. For example, in a large-flow detection system that resets state and starts a new measurement interval periodically [17], a large bursty flow can bypass detection by staying lower than the threshold of throughput across this whole interval, or even by deliberately spreading its burst across two consecutive intervals. Although randomization of measurement intervals can mitigate the problem of a burst spread across a measurement interval, randomized algorithms may be unable to provide strong deterministic guarantees.

In addition to lacking exactness and the arbitrary window model, the storage overhead of existing algorithms may grow unboundedly with the size of the input traffic in the presence of malicious inputs. For example, an adversary can perturb his traffic patterns (e.g., by varying the size and timing of packets) so as to cause algorithmic complexity attacks [11] because many algorithms bound their storage and computational overhead by assuming that the flow sizes follow a certain distribution, such as Zipfian.

To identify large flows over arbitrary windows with low storage overhead, we explore deterministic algorithms with a new model of exactness considering a small ambiguity region. We propose EARDet (**E**xact-Outside-**A**mbiguity-**R**egion **Det**ector), a simple, efficient, and no-per-flow-state large-flow detector which is exact outside an ambiguity region regardless of the input traffic distribution. Built on the Misra-Gries algorithm (a two-pass frequent items finding algorithm based on average frequency) [31], EARDet is a one-pass streaming algorithm with simple operations: it only keeps a small array of counters which are increased or decreased as each new packet arrives. A flow is identified as a large flow if its associated counter exceeds a threshold.

Surprisingly, despite EARDet's strong guarantees, we show in our analysis that EARDet requires extremely small amounts of memory that fit into on-chip SRAM for line-speed packet processing. We discuss implementation details to further demonstrate EARDet's efficiency. EARDet is highly scalable because it focuses on the accurate classification of large and small flows; unlike prior approaches, it does not aim to estimate flow volumes or identify the medium flows. In addition to our theoretical analysis, we also evaluate EARDet using extensive simulations based on real traffic traces. We demonstrate that existing approaches suffer from high error rates under DoS attacks, whereas EARDet can effectively detect large flows in the face of both flooding and burst DoS attacks [22, 25].

Our main contributions are as follows.

- We propose a deterministic streaming algorithm that is exact outside an ambiguity region regardless of the input traffic. Two novel settings distinguish EARDet from previous work: it monitors flows over arbitrary windows, and it supports exact detection outside an ambiguity region.
- We rigorously prove the two guarantees—catching all large flows and preventing false accusation on any small flows—without making assumptions about the flow size distribution.
- Our numerical analysis shows that EARDet can operate at 40 Gbps high-speed links using only hundreds of bytes of on-chip SRAM, which is substantially smaller than the memory consumption in many existing systems. We also provide guidelines on how to configure EARDet to satisfy application-specific requirements.
- We compare EARDet with two closely related proposals [16, 17] via comparative analysis and extensive simulations based on real and synthetic traffic traces. The results confirm that these two are vulnerable to attack flows that manipulate the input traffic, while EARDet consistently catches all large flows without misclassifying small flows.

2. PROBLEM DEFINITION

Our goal is to design an efficient arbitrary-window-based algorithm which is exact outside an ambiguity region. In this section, we present the system model, formulate the large flow problem over arbitrary windows, and summarize our design goals.

2.1 System Model

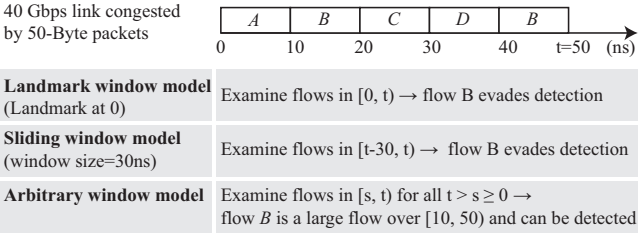
Flow identifiers. Generally, packets are classified into flows based on the *flow identifiers* (or flow IDs) derived from the packet header fields.[2] Because our approach to large-flow detection is generic, we make no assumption on the definition of flow IDs. As in prior traffic monitoring work, we assume flow IDs are unforgeable, which is achievable by ingress filtering [19] and source authentication [4, 24, 27, 32].

Packet streams. Let $\mathcal{X}$ be the packet space. We consider a packet stream $X = \langle x_1, \cdots, x_k \rangle$ coming through a link of capacity ρ, where $x_i \in \mathcal{X}\ \forall i = 1 \cdots k$. Packets in X are processed in sequence by a detection algorithm for identifying large flows. The algorithm can only make one pass over the packet stream due to the high link capacity and limited memory.

For a packet x, we denote by $\mathsf{time}(x)$ the time at which the detection algorithm observes the packet, by $\mathsf{size}(x)$ the size of the packet, and by $\mathsf{fid}(x)$ the flow ID of the packet. The traffic volume of a flow f during a time window $[t_1, t_2)$ is defined as $\mathsf{vol}(f, t_1, t_2) \triangleq \sum_{x \in \mathcal{X}, \mathsf{fid}(x)=f, t_1 \le \mathsf{time}(x) < t_2} \mathsf{size}(x)$.

Synopses. A traffic synopsis is a data structure that summarizes flows and can be used to answer queries regarding certain flow statistics. When a new packet x arrives, the algorithm updates its traffic synopsis based on x's flow

[2]While the flow definitions vary depending on the applications, in most applications a flow consists of packets that share one or more header fields, such as the source IP, destination IP, source port, destination port, and the protocol number.



Landmark window model (Landmark at 0)	Examine flows in [0, t) → flow B evades detection
Sliding window model (window size=30ns)	Examine flows in [t-30, t) → flow B evades detection
Arbitrary window model	Examine flows in [s, t) for all t > s ≥ 0 → flow *B* is a large flow over [10, 50) and can be detected

Figure 1: In this example, a flow is large if it sends more than $40Mbps \cdot w + 500kb$ for any time window of size w. Although flow B violates the limit over the time window $[10, 50)$, it can only be caught in the arbitrary window model.

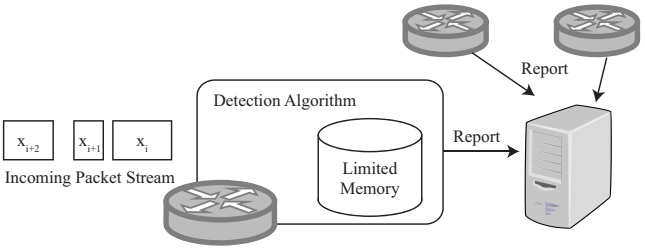


Figure 2: A general framework for a large-flow-detection algorithm. The detection algorithm processes incoming flows and keeps limited state in memory. Results may be reported to a remote server for further analysis.

ID, size, and arrival time. Formally, a large-flow detection algorithm supports three operations over a synopsis S:

- $\mathsf{Init}(params) \to S_0$. The initialization operation takes as inputs the large-flow definitions and desired detection accuracy, etc.
- $\mathsf{Update}(S_{i-1}, x_i) \to S_i$. The update operation outputs an updated synopsis S_i by incorporating the new packet x_i into the previous synopsis S_{i-1}. For convenience, we denote $\mathsf{Update}(S_i, \langle x_{i+1}, \cdots, x_{i+j}\rangle) = \mathsf{Update}(S_{i+1}, \langle x_{i+2}, \cdots, x_{i+j}\rangle) = S_{i+j}$.
- $\mathsf{Detect}(S_i, x_i) \to b \in \{0, 1\}$. The detection operation evaluates S_i to determine if x_i belongs to a large flow.

Time window models. Prior approaches to large-flow detection can be classified into three main categories based on the type of time window they monitor: landmark window [10, 14, 17, 18, 23, 29–31], sliding window [5, 21, 26], and arbitrary window [16].

In the landmark window model, each time window starts at the closest landmark in the past (e.g., a landmark is placed every five seconds) and ends at the current time. In the sliding window model, recent traffic is considered more important than old traffic, so the time window begins at some recent time in the past. The window slides as new packets arrive, such that the measurement incorporates the new packets and excludes the oldest packets. Finally, the arbitrary window model monitors every time window ending at the current time. It is more difficult to evade detection in this model than in others, as illustrated in Figure 1. Note that while the arbitrary window model covers every possible window, flows can still evade detection if the detection algorithm is inaccurate.

2.2 Large-Flow Problem Over Arbitrary Windows

Small, medium, and large flows. A flow f is a *large flow* if there exists a time window $[t_1, t_2)$ over which its volume $\mathsf{vol}(f, t_1, t_2)$ exceeds a high-bandwidth threshold function $\mathsf{TH_h}(t_2 - t_1)$. A flow is a *small flow* if its volume $\mathsf{vol}(f, t_1, t_2)$ is lower than a low-bandwidth threshold function $\mathsf{TH}_\ell(t_2 - t_1)$ over all possible time windows $[t_1, t_2)$. The rest are defined as medium flows, i.e. flows in an ambiguity region.

In this paper we define the two threshold functions in the form of leaky bucket descriptors: $\mathsf{TH_h}(t) = \gamma_h t + \beta_h$ and $\mathsf{TH}_\ell(t) = \gamma_\ell t + \beta_\ell$, where $\gamma_h > \gamma_\ell > 0$ and $\beta_h > \beta_\ell > 0$.[3] Although selecting appropriate parameters largely depends on the targeted application, we provide guidelines for selecting these parameters in Section 4.

Exact-outside-ambiguity-region large-flow problem. As exact solutions are inefficient, we consider a relaxed notion of exactness:

DEFINITION 1. *Given a packet stream, the exact-outside-ambiguity-region large-flow problem returns a set of flows $\mathcal{F}$ such that (1) $\mathcal{F}$ contains every large flow, and (2) $\mathcal{F}$ does not contain any small flow.*

Because the number of large flows can increase indefinitely over time, $\mathcal{F}$ may become too large to fit into the limited on-chip memory. Hence, such an algorithm often reports to a remote server with mass storage that keeps a complete copy of $\mathcal{F}$, as illustrated in Figure 2. The algorithm must therefore operate correctly even without a complete copy of $\mathcal{F}$.

A *positive* is when a flow is added to $\mathcal{F}$, and a *negative* is when a flow is not added to $\mathcal{F}$. Hence, a False Positive of small flow (FP_s) occurs when the detection algorithm wrongly adds a small flow, and a False Negative of large flow (FN_ℓ) occurs when it fails to include a large flow.

The exact-outside-ambiguity-region large-flow problem is reasonable for two reasons: (1) It confines the damage by large flows and allows existing techniques (e.g., Sample and Hold [17]) to handle the medium flows statistically. (2) Prior work [17] [16] also involves a region similar to our ambiguity region, but they only provide probabilistic bounds outside the region.

2.3 Design Goals

Our main goals are as follows:

Exactness outside an ambiguity region. To achieve exactness outside an ambiguity region in traffic monitoring, we desire a deterministic monitor algorithm which identifies every large flow including bursty flow (i.e., no FN_ℓ)and

[3]Instead of using leaky bucket descriptors, prior work in the landmark window model often defines the high-bandwidth threshold to be a fraction of the link bandwidth, e.g., $\gamma_h = 0.01$ and $\beta_h = 0$. However, it is infeasible to adopt this fraction-based definition when it comes to the arbitrary window model because every flow will violate the threshold over a sufficiently small time window (e.g., a window containing only one packet).

protects every small flow (i.e., no FP_s) with no assumption on the input traffic or attack pattern. Hence, the high-bandwidth and low-bandwidth thresholds are also called the no-FN_ℓ and no-FP_s thresholds in this paper, respectively.

Scalability. Although using per-flow leaky buckets enables exact and instantaneous detection of large flows, keeping per-flow state is impractical due to the large number of flows in the Internet. Hence, the algorithm should require few per-packet operations and maintain small router state that fits in fast yet scarce storage devices (e.g., on-chip SRAM or even registers) regardless of input traffic or attack pattern, such that the detection algorithm can operate at line rate.

Fast detection. Fast detection of large flows minimizes collateral damage. Hence, for a large flow violating the high-bandwidth threshold over $[t_1, t_2)$, the algorithm should detect the flow no later than $t_2 + t_{process}$, where $t_{process}$ is the time it takes to process a packet.

3. ALGORITHM

In this section, we first investigate and prove the *no-FP_s* and *no-FN_ℓ relationships* between landmark and arbitrary window models because they are useful for constructing large-flow algorithms over arbitrary windows. Based on these relationships, we present EARDet, a streaming algorithm that efficiently addresses the large-flow problem over arbitrary windows (as defined in Definition 1) with exactness outside an ambiguity region. Finally, we discuss the implementation and optimization techniques in detail, and numerically demonstrate that EARDet can operate at high-speed links while using only hundreds bytes of on-chip SRAM.

It is important to investigate the relationships between landmark and arbitrary window models because they enable us to draw on the rich experience of research on the large-flow problem over landmark windows [10,14,17,18,23,29–31] for designing arbitrary-window algorithms. Particularly, we are interested in knowing whether and to what extent we can leverage existing landmark-window algorithms to build arbitrary-window ones. The technical contributions of this paper include proving two theorems that shed light toward a systematic approach applying existing landmark-window algorithms to arbitrary-window algorithms.

EARDet leverages the Misra-Gries (MG) algorithm [31], which finds all frequent items in a data stream in one pass but may falsely include non-frequent items. The MG algorithm works over landmark windows in the sense that the landmark is at the beginning of the data stream. The research challenges here include (1) how to preserve MG's no-FN_ℓ property (over landmark windows) when porting it to the arbitrary window model, and (2) how to achieve the no-FP_s property when processing packets in one pass. EARDet modifies the MG algorithm in several novel ways to effectively address the above challenges. Interestingly, despite these simple modifications, we prove that EARDet achieves both no-FP_s and no-FN_ℓ properties over arbitrary windows, thereby providing strong guarantees regardless of input traffic.

3.1 Relationships Between Landmark and Arbitrary Windows

Here is a straightforward yet inefficient solution to the exact-outside-ambiguity-region large-flow problem over arbitrary windows: the algorithm divides the problem into multiple sub-problems that each can be handled by a landmark-window algorithm, $\mathcal{L}$. More concretely, let L^i be a copy of $\mathcal{L}$ that monitors a time window starting from packet x_i and ending at the current time. For every newly arrived packet x_i, the algorithm initiates L^i, and adds x_i to the new as well as all previous copies, $L^1, L^2, \cdots, L^i$. Then the algorithm combines the answers returned by $L^1, L^2, \cdots, L^i$. This straightforward solution is correct, but requires space linear in the length of the traffic stream, which is prohibitively expensive.

To make it more efficient, the key idea is to eliminate redundant copies of $\mathcal{L}$. To show why this is possible, we formally state two relationships between landmark windows and arbitrary windows.

No-FP_s relationship. We observe that only one copy of $\mathcal{L}$ is needed to achieve the no-FP_s property over arbitrary window. Specifically, Theorem 2 states that if an algorithm ensures (L1) no FP_s in the landmark window model, then it also ensures (A1) no FP_s in the arbitrary window model.

THEOREM 2. *If an algorithm satisfies*

- L1: *For all t, it never reports a flow whose volume is below $\gamma'_\ell t + \beta'_\ell$ over time interval $[0, t)$.*

then it must also satisfy

- A1: *It never reports a flow whose volume is below $\gamma_\ell(t_2 - t_1) + \beta_\ell$ over time interval $[t_1, t_2)$ for all $t_2 > t_1$.*

when $\gamma'_\ell = \gamma_\ell$ and $\beta'_\ell = \beta_\ell$.

Proof sketch: L1 implies A1 because if a flow sends less than $\gamma_\ell(t_2 - t_1) + \beta_\ell$ for all intervals $[t_1, t_2)$, it must also send less than $\gamma'_\ell t + \beta'_\ell$ for all intervals $[0, t)$ when $\gamma'_\ell = \gamma_\ell$ and $\beta'_\ell = \beta_\ell$. ■

No-FN_ℓ relationship. The no-FN_ℓ relationship is more challenging to prove. We observe that only one copy of $\mathcal{L}$ is needed to achieve the no-FN_ℓ property if *$\mathcal{L}$'s traffic synopsis is "similar" to the initial state* throughout the execution of the algorithm, as checking such a synopsis is roughly equivalent to checking all of $L^1, L^2, \cdots, L^i$. In other words, we can keep only one synopsis which somehow approximates the synopsis in each sub-problem.

Formally, we define a distance metric $\mathsf{dis}(S, S')$ quantifying the similarity between two synopses:

$$\mathsf{dis}(S, S') \triangleq \min_{X, S' = \mathsf{Update}(S, X)} \mathsf{tspan}(X)$$

where the time span of a packet sequence is defined as $\mathsf{tspan}(X) = \max_{x \in X} \mathsf{time}(x) - \min_{x \in X} \mathsf{time}(x)$. S is defined as a snapshot of the algorithm's internal state and thus does not depend on time. The distance between two snapshots is the shortest time it takes to convert one to the other given a link capacity.

Theorem 3 states if an algorithm ensures (L2) no FN_ℓ in the landmark window model and (L3) its synopsis is bounded, it also ensures (A2) no FN_ℓ in the arbitrary window model.

THEOREM 3. *If an algorithm satisfies*

- L2: *For all t, it reports all flows whose volume exceed $\gamma'_h t + \beta'_h$ over time interval $[0, t)$.*
- L3: *Throughout the execution of $\mathcal{L}$, $\mathsf{dis}(S_0, S_i) \leq \Delta$, where Δ is a small constant and $S_i = \mathsf{Update}(S_0, \langle x_1, \cdots, x_i \rangle)$.*

then it must also satisfy

- A2: *It always reports a flow whose volume exceeds* $\gamma_h(t_2 - t_1) + \beta_h$ *over time interval* $[t_1, t_2)$ *for some* $t_2 > t_1$.

when $\gamma_h(t_2 - t_1) + \beta_h \geq \gamma'_h(t_2 - t_1 + \Delta) + \beta'_h$.

Proof sketch: Let $\mathcal{L}$ be an algorithm satisfying L2 and L3. Let f be a flow that sends more than $\gamma_h(t_2 - t_1) + \beta_h$ over some time interval $[t_1, t_2)$, and t_2 is the smallest among all possible values if the flow f violates the spec multiple times. To prove this No-FN$_\ell$ relationship, in the following we show that $\mathcal{L}$ can catch any f when $\gamma_h = \gamma'_h$ and $\beta_h \geq \beta'_h + \gamma_h\Delta$, thus satisfying A2 as well.

For convenience, we denote by X_a^b the incoming packet stream between time interval $[a, b)$. Since $\mathcal{L}$ satisfies L3 (i.e., the synopsis state of $\mathcal{L}$ is always bounded), $\mathsf{dis}(S_0, S^{t_1}) \leq \Delta$ where S^{t_1} is its synopsis state at t_1.

Based on the definition of the distance function, there exists a packet sequence X' with a time span less than Δ and $\mathsf{Update}(S_0, X') = S^{t_1}$. In other words, from the algorithm's perspective, $\mathsf{Update}(S_0, X_0^{t_2}) = \mathsf{Update}(S_0, \mathcal{X}' \| X_{t_1}^{t_2})$,[4] i.e., the two packet sequences produce identical synopses. As a result, if the algorithm $\mathcal{L}$ can detect f in $X' \| X_{t_1}^{t_2}$ then it can also detect f in $X_0^{t_2}$ because the output of the detection function, Detect, solely depends on the synopsis.

Moreover, by construction, f sends more than $\gamma_h(t_2 - t_1) + \beta_h$ in the new sequence $X' \| X_{t_1}^{t_2}$, whose time span is $t_2 - t_1 + \Delta$. Therefore, $\mathcal{L}$ can detect f in the new sequence because $\gamma_h(t_2 - t_1) + \beta_h \geq \gamma'_h(t_2 - t_1 + \Delta) + \beta'_h$ holds when $\gamma_h = \gamma'_h$ and $\beta_h \geq \beta'_h + \gamma_h\Delta$. Hence, $\mathcal{L}$ can also detect f in the original stream, $X_0^{t_2}$. ■

We note that L1 and A2 contradict each other for any parameter selection: for any $\gamma'_\ell, \beta'_\ell, \gamma_h$, and β_h, consider an interval $[t_1, t_2)$ satisfying $t_1 = t_2 - \epsilon$ and $t_2 > \frac{\beta_h - \beta'_\ell + \gamma_h\epsilon + 1}{\gamma'_\ell}$. Then a flow sending $\gamma_h(t_2 - t_1) + \beta_h + 1$ over $[t_1, t_2)$ will violate the high bandwidth threshold over $[t_1, t_2)$ but comply with $\gamma'_\ell t_2 + \beta'_\ell$ over $[0, t_2)$. That is, no algorithm can satisfy (A2, L2, L3) and (A1, L1) at the same time.

The above two theorems can be viewed as guidelines for designing new arbitrary-window algorithms based on existing landmark-window algorithms.

3.2 Algorithm Construction

Several existing landmark-window approaches [10, 14, 17, 23, 29–31] satisfy L2 when β'_h is set to zero. Among these approaches, we observe that the MG algorithm [31] can be made to satisfy L2 in a general setting (i.e., β'_h can be non-zero) as well as L3 with slight modifications. As a result, we choose to leverage the MG algorithm for designing EARDet.

We prove in the next section that EARDet's design ensures L2 and L3, and therefore achieves the no-FN$_\ell$ property (i.e., catching every large flow) based on Theorem 3. We also prove that EARDet achieves the no-FP$_s$ property (i.e., protecting every small flow), whereas the MG algorithm requires a second pass to remove false positives in the landmark window model.

Background of the MG algorithm. We briefly review the MG algorithm, which inspires our design. The MG algorithm finds the exact set of frequent items (defined as items that appear in a stream of m items for more than $\frac{m}{n+1}$ times) in two passes with only n counters. This algorithm generalizes the Majority algorithm [6, 20], which focuses on the case when $n = 2$. The same generalization was rediscovered by Demaine et al. [14] and Karp et al. [23].

The MG algorithm assumes an associative array of counters indexed by items. Counters are initialized to zeros. We say that an item is stored if its counter is above zero. For each incoming item e, the MG algorithm works as follows: (1) If e is stored (i.e., $ctr[e] > 0$), increase $ctr[e]$ by 1; (2) Else if the number of non-zero counters is less than n, $ctr[e] = 1$; and (3) Otherwise, decrease all non-zero counters by 1.

Since there are at most n non-zero counters kept at any time, the storage overhead is $O(n)$. This can be easily extended to items with positive weights. After the first pass, the MG algorithm guarantees that every frequent item is associated with a non-zero count, and a second pass is required to remove falsely included infrequent items.

The correctness of this algorithm can be shown intuitively as follows: Suppose an item e appears more than $\frac{m}{n+1}$ times, but is not stored at the end. The total count would have been reduced by more than $\frac{m}{n+1} \cdot (n+1) = m$ counts during the execution, which is impossible since it is more than the total number of items.

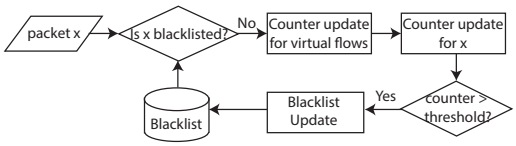


Figure 3: EARDet's decision diagram.

EARDet Overview. Figure 3 illustrates EARDet's decision diagram for each incoming packet. At a high level, EARDet works similarly to the MG algorithm except three crucial distinctions:

- **Blacklist:** EARDet keeps a local blacklist $\mathcal{L}$ that stores recently identified large flows. Counters are updated only if the flow ID of the packet is not blacklisted. The main purpose of keeping the blacklist is to avoid increasing a flow's counter when the counter value has already exceeded a *counter threshold*, β_{TH}. Additionally, we can avoid spending unnecessary resources on accounting blacklisted flows. We bound the blacklist's size in Section 3.3.
- **Counter threshold:** A flow is added to the blacklist if its associated counter value exceeds a threshold β_{TH}. Setting a counter threshold together with blacklisting ensures that counter values are always confined, i.e., $\leq \beta_{TH} + \alpha$, where α is the maximum packet size.
- **Virtual traffic:** In contrast to the frequent-item problem, the large-flow problem has to take the idle time between two consecutive packets into account so as to accurately detect large flows with respect to the link capacity. EARDet handles this by virtually filling the unused bandwidth with virtual traffic. Virtual traffic consists of multiple virtual flows, each of which is crafted purposely to comply with the low-bandwidth threshold, thus avoiding unnecessary alarms.

Algorithm description. Algorithm 1 describes how EARDet works. As the MG algorithm, EARDet keeps

[4] $\|$ denotes concatenating two packet sequences into one sequence.

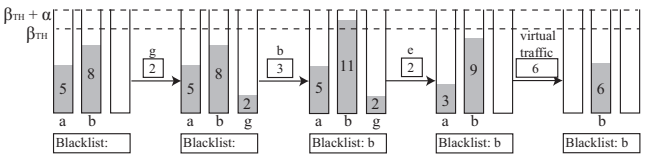


Figure 4: Example of EARDet's counter update.

n counters, each initialized to zero. Counters are stored in an associative array indexed by flow IDs, and the number of non-zero counters never exceeds n. EARDET counters are increased and decreased by the size of the packets, since a packet of size w can be viewed as w uni-sized items. We denote by C the set of non-zero counters. We discuss how to implement EARDET efficiently in Section 3.3.

Algorithm 1 EARDET

1: Initialization ($S \leftarrow \mathsf{Init}(n)$, Line 8-9)
2: **for** each packet x in the stream **do**
3: **if** x's FID f is not blacklisted ($f \notin \mathcal{L}$) **then**
4: Update counters for virtual traffic (Line 18-22)
5: Update counters for x ($S \leftarrow \mathsf{Update}(S, x)$, Line 10-17)
6: **if** detect violation ($\mathsf{Detect}(S, x) == 1$, Line 21-22) **then**
7: Add f to blacklist ($\mathcal{L} \leftarrow \mathcal{L} \cup \{f\}$)

8: **Initialization,** $\mathsf{Init}(n)$
9: initialize all counters to zeros, $\mathcal{L} \leftarrow \emptyset$, $C \leftarrow \emptyset$

10: **Update counters for packet** x**,** $\mathsf{Update}(S, x)$
11: **if** x's FID f is kept ($f \in C$) **then**
12: Update f's counter by the packet size w ($c_f \leftarrow c_f + w$)
13: **else if** less than n counters are kept ($|C| < n$) **then**
14: Set f's counter to w ($c_f \leftarrow w$, $C \leftarrow C \cup \{f\}$)
15: **else**
16: Decrease all counters by $d = \min\{w, \min_{j \in C} c_j\}$
17: Set c_f to $w - d$, and $\forall j$ remove j from C if $c_j = 0$

18: **Update counters for virtual traffic between** x_i **and** x_{i-1}
19: Compute the virtual traffic size, v ($v = \rho t_{idle} - \mathsf{size}(x_{i-1})$, and $t_{idle} = \mathsf{time}(x_i) - \mathsf{time}(x_{i-1})$)
20: For each unit u in the virtual traffic, update counters as if u belongs to a new flow (e.g., unit is 1 byte)

21: **Detect violation,** $\mathsf{Detect}(S, x)$
22: Return whether x's flow counter exceeds threshold ($c_f > \beta_{TH}$)

Figure 4 gives an example showing how to update counters when $n = 3$, $\beta_{TH} = 10$, and $\alpha = 3$, where α is the maximum packet size. First, since there is an empty counter, flow g is added and its counter value becomes 2, the size of the new packet. Then, since flow b is stored already, its counter is increased by 3. The new value of $ctr[b]$ exceeds β_{TH}, and thus flow b is blacklisted. The next flow, e, is not stored yet and there is no empty counter, so all counters are decreased by the packet size. Finally, the virtual traffic is divided into single-unit packets with new flow IDs, resulting in the final state.[5]

Despite EARDET's simple operations, work remains to prove the no-FP$_s$ and no-FN$_\ell$ properties and to devise practical parameters. We answer these in Section 4.

[5]Conceptually, the counter values are updated as follows: $[3, 9, 0] \rightarrow [3, 9, 1] \rightarrow [2, 8, 0] \rightarrow [2, 8, 1] \rightarrow [1, 7, 0] \rightarrow [1, 7, 1] \rightarrow [0, 6, 0]$. Section 3.3 discusses techniques to accelerate this process.

3.3 Data Structure and Optimization

While EARDET requires very little memory state, its processing delay may be high in a naive implementation where EARDET accesses every counter for each decrement operation (i.e., Line 16 in Algorithm 1). We now present several optimization techniques to reduce the number of memory accesses and the processing time.

Reducing number of memory accesses. We minimize the number of memory accesses per packet via the following technique. First, we keep counters in a data structure that allows insertion, deletion, and finding the minimum in logarithmic time. Data structures such as balanced search trees and heaps can satisfy our requirements. Moreover, counter values are not absolute but relative to a *floating ground*, c_{ground}. Hence, the decrement operation, which requires decreasing all counters previously, can now be achieved by elevating the floating ground. The detection function becomes $c_f - c_{ground} > \beta_{BF}$.

The increment operation on Line 12 takes $O(1)$ time using an associative array. Adding a value to an empty counter as described on Line 14 takes $O(\log n)$ time because we have to insert the counter to the data structure. To prevent counter overflow, EARDET periodically resets the floating ground to zero and deducts all counters accordingly.

Efficient counter update for virtual traffic. Virtual traffic ensures accurate accounting of unused bandwidth, but efficient implementation is needed to handle virtual traffic at line speed. We now discuss how to efficiently update EARDET's counters for virtual traffic. Particularly, we aim to divide virtual traffic into multiple virtual flows in a way to minimize the time to process such virtual flows. The only constraint is that each virtual flow should comply with the low-bandwidth threshold to avoid triggering false alarms.

As Line 20 of Algorithm 1 shows, for each unit u of the virtual traffic, EARDET updates its counters (i.e., $\mathsf{Update}(S, u)$) as if the traffic unit belongs to a new flow. We can minimize the number of updates by maximizing the unit size. To avoid false detection, the maximum size per unit is β_{TH} bytes. As β_{TH} must be larger than the minimum packet size (i.e., 40 bytes) for practical use, the overhead of using β_{TH}-byte virtual flows is bounded by the worst-case scenario where the link is congested by minimum-sized packets.

We can further optimize this task based on the following observation: Once all counters become empty, they should stay empty until the next real packet comes. Furthermore, since the maximum counter value is $\beta_{BF} + \alpha$, counters will all be empty if the size of the virtual flow $\geq (\beta_{BF} + \alpha) \cdot n$. (A tighter condition is if the virtual traffic size $\geq (\max_j c_j) * n - \sum_j c_j$, but this requires keeping track of the sum of all counters.) In other words, EARDET can simply reset all counters to zeros and avoid any update if the virtual traffic size exceeds a certain threshold.

Counter implementation. For efficiency, counters are implemented as integers (e.g., in bytes) rather than non-integer numbers. While packet sizes are always multiples of bytes, the size of virtual traffic may be non-integer, which introduces biases on EARDET's guarantees. For example, given a 800Mbps link and a nanosecond time precision at the router, the size of a 1-ns virtual traffic is 0.1 bytes.

We bound such biases with a slightly modified algorithm that adjusts virtual traffic. Let us denote by $\{v_1, v_2, \cdots\}$ the sizes of a sequence of virtual traffic and by $\{v'_1, v'_2, \cdots\}$ the adjusted sizes. We maintain an extra field called "carryover", *co*, which keeps the amount of uncounted virtual traffic. *co* is initialized to zero, and we ensure that $-0.5 \leq co < 0.5$ for all time. Virtual flows are adjusted such that $v'_i \leftarrow [v_i + co_i]$ and $co_{i+1} \leftarrow co_i + v_i - v'_i$ where co_i is the value of *co* before proceeding v_i. By construction, v'_is are all integers, and for any a, b, $|\sum_a^b v_i - \sum_a^b v'_i| = |co_{b+1} - co_a| \leq 1$. In other words, the adjusted virtual traffic differs from the original one by at most 1 unit for any time interval. Consequently, the modified algorithm guarantees to catch flows violating $\mathsf{TH}_h(t) = \gamma_h t + (\beta_h + 1)$ and guarantees not to catch any flow conforming to $\mathsf{TH}_\ell(t) = \gamma_\ell t + (\beta_h - 1)$.

Bounding the blacklist. EARDet keeps in memory not only counters but also a local blacklist storing recently detected large flows.

We propose a simple mechanism to bound $|\mathcal{L}|$ and thus prevent algorithmic complexity attacks to overflow the blacklist as follows. Since the blacklist's main purpose is to avoid increasing a flow's counter when the counter value has already exceeded a counter threshold, once a counter value has dropped below the threshold, the flow can be removed from the blacklist. In this way, the size of $\mathcal{L}$ will always be bounded by n because only flows that are currently stored can be blacklisted.

The key observation here is that flows that are not currently stored can be removed from the blacklist because such removal will not affect EARDet's no-FN$_\ell$ and no-FP$_s$ guarantees, as in EARDet whether a flow will be caught or not does not depend on other flows' behavior. The detector can periodically report the current blacklist to a remote administrator, such that the administrator keeps a complete list of detected large flows, while the detector maintains a small blacklist that helps avoid increasing a flow's counter when the counter value has already exceeded β_{TH}. The only tradeoff of this mechanism is that EARDet may spend unnecessary resources on accounting flows that have been identified as large flows.

Parallelizing EARDet. A common way to reduce processing time is parallelization. EARDet can be parallelized at both the algorithm and instruction levels. At the algorithm level, we can randomly distribute the flows (thus the workload) among multiple copies of EARDet. At the instruction level, we can access and update multiple counters in parallel using multi-port SRAM when the operations are order insensitive. We note that special-purpose SRAMs (i.e., multi-port SRAMs) can support multiple read/write simultaneously [38].

3.4 Storage and Computational Complexity

Given the above optimization techniques, we analyze EARDet's storage and computational overhead.

To operate at line rate on OC-768 (40 Gbps) high-speed links, a typical 3.2 GHz processor has to process 40 million medium-sized (1000 bits) packets per second, which means the per-packet processing time should be at most 32 ns or 76 CPU cycles. In this analysis, we consider the following memory model for commodity routers: CPU has 32 KB L1 cache, 256 KB L2 cache, 20 MB L3 cache, and gigabytes main DRAM memory. Accessing L1, L2, and L3 caches takes 4, 12, and 30 CPU cycles, respectively; accessing the main memory is as slow as 300 cycles.

Storage complexity. EARDet keeps an extremely small traffic synopsis, a blacklist of detected flows, and storage of flow ID keys (i.e. unique keys consist of IP address and port number) for flow identifier. The synopsis consists of n counters and a constant number of additional variables for optimization such as the floating ground. In IPv4, a flow ID key is 48 bits, and for IPv6, it is 144 bits. By implementing the tree map using a red-black tree [8], each counter needs store only one flow ID key. In most applications the synopsis and flow ID keys will be small enough to fit entirely in the router's L1 cache. For instance, in IPv4, using 100 32-bit counters and 100 48-bit key variables requires only 960 bytes; even in IPv6, the same case requires only 2200 bytes, which can fit in the L1 cache. (Guided by our analysis in Appendix A, we use 100 counters as a representative example.) If the number of counters is much larger than 100 in some configuration, we can also use L2 cache with some performance cost. In practice, a special-purpose device with more fast memory can also be installed for large flow detection. Moreover, we can flexibly tune the counter size to further reduce the memory requirement at the cost of a wider ambiguity region between the no-FP$_s$ and no-FN$_\ell$ thresholds.

Computational complexity. For each packet, EARDet looks up and updates one or more counters, and adjusts the internal data structure (e.g., a heap) of counters. In EARDet, locating and updating a counter requires an average of $O(1)$ memory access in a hash-map-based associative memory. Adjusting the data structure of n counters requires $O(\log n)$ memory accesses.

Since EARDet's state is small enough to fit into the L1 cache as we discussed, the per-packet processing time can be as low as tens of nanoseconds, which is suitable for processing packets at 40Gbps high-speed links. Even when all of EARDet's data is in the L2 cache, EARDet can still process packets at line speed on 13Gbps.

4. ANALYSIS

In this section, we prove the no-FP$_s$ and no-FN$_\ell$ properties. Furthermore, we analyze the incubation period of large flows, discuss EARDet's tradeoffs, and present practical guidelines for EARDet configuration. Finally, we compare EARDet with closely related proposals [16, 17] to demonstrate that it outperforms prior work in terms of both efficiency and detection accuracy.

We consider a network link with a capacity of ρ, and a EARDet detector with n counters. The counter threshold is β_{TH}. Once the value of a counter exceeds β_{TH}, the associated flow will be judged as a large flow and cut off immediately. Hence, the maximum value of each counter is $\beta_{TH} + \alpha$, where α is the maximum packet size. Table 1 summarizes the notations used in this section. We will discuss the relationship among parameters and how to set them in the Section 4.6.

4.1 Large-Flow False-Negative Analysis

THEOREM 4. ***No-FN$_\ell$ property.*** *EARDet detects every flow violating the high-bandwidth threshold* $TH_h(t) =$

Table 1: Table of Notations.

Network management parameters:		
ρ	$\triangleq$	Rate of link capacity
α	$\triangleq$	Maximum packet size
t_{upincb}	$\triangleq$	Upper bound of t_{incb} for any large flows
TH_ℓ	$\triangleq$	Low-bandwidth threshold
TH_h	$\triangleq$	High-bandwidth threshold
γ_ℓ, β_ℓ	$\triangleq$	Rate and burst for low-bandwidth threshold
γ_h	$\triangleq$	Rate for high-bandwidth threshold
Tunable parameters:		
n	$\triangleq$	Number of counters in EARDet
β_{TH}	$\triangleq$	Threshold of counters($> \beta_\ell$)
Parameters that depend on tunable parameters:		
β_h	$\triangleq$	Burst for high-bandwidth threshold
β_Δ	$\triangleq$	$\beta_{TH} - \beta_\ell$
Other notation:		
$R(t_1, t_2)$	$\triangleq$	Average flow rate in $[t_1, t_2)$
t_{incb}	$\triangleq$	Incubation period of large flows
R_{NFN}	$\triangleq$	No-FN$_\ell$ rate
R_{NFP}	$\triangleq$	No-FP$_s$ rate

$\gamma_h t + \beta_h$ over a time window of length t, when $\gamma_h \geq R_{NFN} = \frac{\rho}{n+1}$ and $\beta_h \geq \alpha + 2\beta_{TH}$.

Proof sketch: Firstly, we prove that EARDet satisfies L3 in Theorem 3. According to Algorithm 1, the maximum value of each counter c_i is $\beta_{TH} + \alpha$, and there are at most n non-zero counters at any time. Also, given any valid synopsis $S = \{c_i\}$ we can construct a packet stream X consisting of c_i bytes for flow i and no space between packets, and by construction $S = \mathsf{Update}(S_0, X)$. Combining the above two arguments and the definition of the distance function, we conclude that $\mathsf{dis}(S_0, S) \leq \frac{\mathsf{size}(\mathcal{X})}{\rho} \leq \frac{(\beta_{TH}+\alpha)n}{\rho}$. That is, setting $\Delta = \frac{(\beta_{TH}+\alpha)n}{\rho}$ satisfies L3.

Next we prove that EARDet satisfies L2 in Theorem 3 as well when setting $\gamma'_h = R_{NFN} = \frac{\rho}{n+1}$ and $\beta'_h = \beta_{TH}$. We prove by contradiction and assume there were a flow f violating $\gamma'_h + \beta'_h$ in the landmark window model at time t but not being detected (i.e., $c_f < \beta_{TH}$). This assumption implies that more than $\gamma'_h t + \beta'_h - \beta_{TH}$ amount of flow f would have been canceled out[6] during the decrement step, or equivalently, more than $(\gamma'_h t + \beta'_h - \beta_{TH}) \cdot (n+1) = \gamma'_h t \cdot (n+1) = \rho t$ amount of traffic would have been canceled out. This statement, however, contradicts the setting where the maximum traffic for t units of time is ρt. Thus, f cannot escape from EARDet, and L2 is satisfied by EARDet.

Based on Theorem 3, we conclude that EARDet satisfies A2 when $\gamma_h = \gamma'_h = \frac{\rho}{n+1}$ and $\beta_h \geq \beta'_h + \gamma_h \Delta = \beta_{TH} + \frac{\rho}{n+1}\frac{(\beta_{TH}+\alpha)n}{\rho} = \beta_{TH} + \frac{n}{n+1}(\beta_{TH} + \alpha)$. In particular, EARDet catches every flow violating the threshold $TH_h(t) = \gamma_h t + \beta_h$ when $\gamma_h \geq R_{NFN}$ and $\beta_h \geq \alpha + 2\beta_{TH}$. That is, EARDet catches all large flows in the arbitrary window model. ■

4.2 Small-Flow False-Positive Analysis

As discussed in Section 3.1, no algorithm can satisfy A2 in Theorem 3 and L1 in Theorem 2 at the same time. Hence, rather than applying Theorem 2, we have to take a different approach in proving the no-FP$_s$ property.

[6]A packet byte is canceled out if it does not contribute to the corresponding counter.

To analyze EARDet's no-FP$_s$ property, we consider how EARDet increases and decreases its counter values. Firstly, let us examine all cases based on the types of incoming flows. We say a flow is old if it is stored in the counters currently; otherwise the flow is new.

1. When the incoming flows are virtual flows and there are l empty counters, in a time window t, the decrement is $\frac{\rho}{l+1}t$ on all counters, and the increment is 0. $(l = 0, 1, 2, 3, ..., n)$
2. When the incoming flows are new real flows and there is no empty counter, in a time window t, the decrement is ρt on all counters and the increment is 0 (which is the same as the first case when $l = 0$).
3. When the incoming flows are old real flows, or new real flows and there are some empty counters, in time interval t, the decrement is 0 and the increment is $\rho\, t$ on one counter.

Thus, in the first and second cases, when there are l empty counters in the detector, the decrement is always $\frac{\rho}{l+1}t$ in the interval of t; and in the third case, the increment is always $\rho\, t$ on one counter in the interval of t. Finally, the increment and decrement cannot happen at the same time.

Lemma 5. *For any small flow f that complies with the low-bandwidth threshold (i.e., $TH_\ell(t) = \gamma_\ell\, t + \beta_\ell$), once the flow f is associated to a counter at t_1, this counter will be always lower than β_{TH} after time $t_1 + t_{\beta_\ell}$, if the counter is occupied by the same flow as the flow f, where $t_{\beta_\ell} = \frac{(n-1)\alpha+(n+1)\beta_\ell}{[1-(n+1)\gamma_\ell/\rho]\rho}$.*

Due to space limitations, the detailed proofs are in our technical report [37].

Theorem 6. ***No-FP$_s$ property.*** EARDet *will not catch any flow complying with the low-bandwidth threshold $TH_\ell(t) = \gamma_t + \beta_\ell$ for all time windows of length t, when $0 < \beta_\ell < \beta_{TH}$, $\gamma_\ell < R_{NFP}$, where $R_{NFP} = \frac{\beta_\Delta}{(n-1)\alpha+(n+1)\beta_\ell+(n+1)\beta_\Delta} \cdot \rho$.*

Proof sketch: According to Lemma 5, to avoid catching a small flow f, the counter should be smaller than β_{TH} before t_{β_ℓ}. Hence, we choose a γ_ℓ to achieve $\gamma_\ell\, t_{\beta_\ell} + \beta_\ell < \beta_{TH}$. Then, $\frac{(n-1)\alpha+(n+1)\beta_\ell}{[1-(n+1)\gamma_\ell/\rho]\rho} < \frac{\beta_{TH}-\beta_\ell}{\gamma_\ell}$,

$$\Leftrightarrow \gamma_\ell < \frac{\beta_\Delta}{(n-1)\alpha + (n+1)\beta_\ell + (n+1)\beta_\Delta} \cdot \rho \qquad (1)$$

The theorem is proved. ■

Interestingly, Theorem 6 shows that γ_ℓ approaches $\frac{\rho}{n+1}$ as β_Δ increases, but cannot go beyond $\frac{\rho}{n+1}$.

4.3 Relationship between Low-Bandwidth and High-Bandwidth Thresholds

Before the discussion, let us define two concepts:
Rate Gap: The ratio between γ_h and γ_ℓ (i.e. γ_h/γ_ℓ);
Burst Gap: The ratio between β_h and β_ℓ (i.e. β_h/β_ℓ).

Based on Theorems 4 and 6, the minimum rate gap is: $(\gamma_h/\gamma_\ell)_{min} = \frac{R_{NFN}}{R_{NFP}} = \frac{(n-1)\alpha+(n+1)(\beta_\ell+\beta_\Delta)}{\beta_\Delta(n+1)}$

Given $\beta_\Delta = \beta_{TH} - \beta_\ell$ and $n + 1 \doteq n \doteq n - 1$, we get

$$(\gamma_h/\gamma_\ell)_{min} \doteq 1 + \frac{2\alpha/\beta + 2}{\beta_h/\beta_\ell - (\alpha/\beta_\ell + 2)} \qquad (2)$$

Thus, the minimum possible rate gap $(\gamma_h/\gamma_\ell)_{min}$ is mainly influenced by the burst gap β_h/β_ℓ. Equation (2) tells us a) β_h/β_ℓ cannot be lower than $\alpha/\beta_\ell + 2$. b) EARDet only needs a low β_h/β_ℓ to achieve small enough $(\gamma_h/\gamma_\ell)_{min}$. For example, to achieve $(\gamma_h/\gamma_\ell)_{min} = 10$, we only need $\beta_h/\beta_\ell = 2.53$. c) $(\gamma_h/\gamma_\ell)_{min}$ cannot be lower than 1. $(\gamma_h/\gamma_\ell)_{min}$ approaches to 1 as β_h/β_ℓ grows.

4.4 Incubation Period of Large Flows

To define the incubation period, we first consider a large flow that violates the high-bandwidth threshold over $[t_1, t_2)$, and the packet at t_a triggers the detection. Because of EARDet's no-FN$_\ell$ property, $t_a \leq t_2$. The incubation period is defined as $t_a - t_1$, representing the time duration for which the large flow remains under the radar. We bound the incubation period as follows.

THEOREM 7. *For the flow f which violates $TH_h(t)$ over some time window $[t_1, t_2)$, if its average rate $R(t_1, t_a)$ is larger than R_{atk} in time interval of $[t_1, t_a)$ (R_{atk} is a constant rate larger than $R_{NFN} = \frac{\rho}{n+1}$), then f's incubation period is bounded by*

$$t_{incb} < \frac{\alpha + 2\beta_{TH}}{R_{atk} - \frac{\rho}{n+1}}. \tag{3}$$

Proof sketch: Because $R(t_1, t_a) > R_{atk}$, intuitively the t_{incb} of flow with average rate of $R(t_1, t_a)$ must be shorter than the t'_{incb} of flow with rate of R_{atk}. That is, $t_{incb} < t'_{incb}$.

Assume a flow f' with rate R_{atk} will violate $TH_h(t)$ over time window $[t'_1, t'_2)$, then

$$\begin{aligned} &R_{atk}(t'_2 - t'_1) = \frac{\rho}{n+1}(t'_2 - t'_1) + \alpha + 2\beta_{TH} \\ &\Rightarrow t_{incb} < t'_{incb} = t'_a - t'_1 \leq t'_2 - t'_1 = \frac{\alpha + 2\beta_{TH}}{R_{atk} - \frac{\rho}{n+1}} \end{aligned} \tag{4}$$

Thus, the theorem is proved. ■

From Theorem 7, the bound of the incubation period decreases as R_{atk} increases. In other words, if R_{atk} is fixed, the bound of the incubation period decreases with increasing n, which implies we can reduce the upper bound by adding extra counters. To guarantee detection of flows whose rate is over R_{atk} ($R_{atk} > \frac{\rho}{n+1}$), the minimum number of counters is $\frac{\rho}{R_{atk}} - 1$, and the upper bound on the incubation period can be lowered significantly by adding a few counters. The details will be discussed in Section 4.6.

4.5 Tradeoff Analysis

We discuss three tradeoffs in EARDet: (1) memory consumption (i.e., the number of counters) vs. the rate gap, (2) the rate gap and burst gap, and (3) the rate gap and the upper bound on the incubation time.

First, since the rate gap can be expressed as $\gamma_h/\gamma_\ell > R_{NFN}/\gamma_\ell = \frac{\rho/\gamma_\ell}{n+1}$, we can see that the rate gap decreases with increasing n. Second, Equation (2) shows that the minimum rate gap γ_h/γ_ℓ is mainly influenced by β_h/β_ℓ, namely the burst gap, and the minimum rate gap decreases as the burst gap increases. Finally, Theorem 7 shows that a large burst gap results in a long incubation period. Hence, a small rate gap results in a big burst gap, and a high incubation period.

4.6 How To Engineer The Detector

To engineer our detector, we first identify parameters that are often known or given in a priori. Usually, users want a detector for a specific link capacity, ρ, to protect small flows which comply with the low-bandwidth threshold: $TH_\ell(t) = \gamma_\ell t + \beta_\ell$, and to detect attack flows that violate the high-bandwidth threshold: $TH_h(t) = \gamma_h t + \beta_h$. However, as discussed in Section 4.5, there is a tradeoff between the rate gap and burst gap, so their requirements cannot be both fulfilled. Thus, we choose to satisfy the rate requirement of γ_h, and then set β_h according to γ_h because it is more important to limit the flow rate than the burst size. Furthermore, since we want to minimize the incubation period of large flows, there is an upper bound of the incubation period, t_{upincb}.

We set $\beta_h = \alpha + 2\beta_{TH}$ and $\gamma_h > \frac{\rho}{n+1}$ to guarantee no FN$_\ell$ according to Theorem 4. Since $\beta_{TH} = \beta_\ell + \beta_\Delta$, we only need to decide the number of counters n and β_Δ. Hence, the problem can be simplified as follows. Given ρ, γ_ℓ, β_ℓ, γ_h, α, and t_{upincb}, we aim to calculate n and β_Δ such that the parameters satisfy the constraints in Theorems 4, 6 and 7. The detailed solution and analysis are in Appendix A. The detailed analysis shows that the tunable parameters depend on only the thresholds and are independent of the input traffic.

5. EVALUATION

To evaluate EARDet, we compare EARDet with two closely related proposals, which we refer to as FMF [17] and AMF [16]. Both of our theoretical comparison and trace-based simulations demonstrate that EARDet performs better than prior work in terms of both exactness[7] outside an ambiguity region and efficiency. The simulation results using real and synthetic traffic traces are consistent with the analysis in the theoretical evaluation.

5.1 Theoretical Comparison

Multistage filters. *Fixed-window-based Multistage Filters* (FMF) identify large flows in a fixed measurement interval. A FMF consists of parallel stages, each of which is an array of counters initialized to zeros at the beginning of a measurement interval. Each stage is assigned a hash function that maps a packet's flow identifier to a counter in the stage. For each incoming packet, its flow identifier is hashed to one counter in each hash stage, and the counter value increments by the size of the packet. A flow is considered a large flow if all of its corresponding counters exceed a pre-specified threshold.

Arbitrary-window-based Multistage Filters (AMF) identify large flows over arbitrary windows. To work in the arbitrary window model, AMF replaces each counter in FMF with a leaky bucket of a bucket size u and a drain rate r. A flow is considered a large flow if the corresponding leaky buckets are all violated.

Performance Comparison. Table 2 presents a numerical comparison of EARDet, FMF and AMF, where the high-bandwidth rate is 1% of the link capacity, and the low-bandwidth rate is 0.1% of link capacity. The results of FMF and AMF are derived based on the authors' original analysis

[7]The comparison in this section uses our definition of exactness.

Table 2: A numerical example given the requirements in Section 4.6. γ_h is 1% of the link capacity, and γ_ℓ is 0.1% of link capacity.

Scheme	# of counters	FP_s rate	FN_ℓ rate
EARDET	101	0	0
FMF	101/1000	*no guarantee* / $\leq 0.04^*$	0^*
AMF	101/2000	*no guarantee* / ≤ 0.04	0

*FMF's results are derived in the landmark window model, and thus FMF's FP_s and FN_ℓ rates may be higher in the arbitrary window model.

Table 3: Summary of comparison of three schemes.

Scheme	FP_s	FN_ℓ	Memory	Input Traffic
EARDET	no	no	low	independent
FMF	yes	yes	high	dependent
AMF	yes	no	high	dependent

that assumes a specific number of active flows for the input traffic.

EARDET outperforms the other two approaches in several aspects: 1) EARDET guarantees no false detection of small flows, whereas it is unclear how FMF and AMF can achieve this property. Even with ten times or more extra storage, FMF and AMF still have error rates as high as 0.04. 2) Both EARDET and AMF can detect all large flows. However, FMF has FN_ℓ on bursty flows. 3) EARDET requires much less memory compared with multistage filters. 4) EARDET's performance is independent of the input traffic, because the error rate is always zero. On the contrary, to keep the same false positive rate, multistage filters require more stages as the number of active flows increases. Table 3 summarizes the comparison, which suggests that EARDET is exact outside an ambiguity region and more efficient than prior works.

Although EARDET presents several advantages compared with multistage filters, EARDET cannot estimate the size of a detected flow, which multistage filters achieve.

5.2 Experiment Settings

Datasets. Table 4 summarizes the characteristics of the two datasets used in the experiments. The Federico II dataset contains traces collected at the TCP port 80 of a 200 Mbps link [3, 12, 13]. The CAIDA dataset contains anonymized passive traffic traces from CAIDA's equinix-sanjose monitors on 10 Gbps backbone links [2]. For each dataset, we use the first 30 seconds for the experiments. We define flows based on the source and destination IP addresses.

Table 4: Dataset Information.

Dataset	Link capacity	Avg link rate	# of flows	Avg flow size
Federico II	$200Mbps$	$1.85MB/s$	2911	$19.9KB$
CAIDA	$10Gbps$	$279.65MB/s$	2517099	$3.3KB$

Attack scenarios. As stated in the theoretical analysis in Section 5.1, EARDET is exact outside an ambiguity region and efficient comparing to prior algorithms. The theoretical analysis focuses on the worst-case behaviors. To investigate how EARDET performs in reality, we conduct experiments and compare EARDET with FMF and AMF in the face of common attacks.

In particular, we generate attack flows using two simple strategies—*flooding attacks* and *Shrew DoS attacks* [22,25]—and then mix real traces with artificially generated attack flows to simulate an attack environment. In a flooding attack, the adversary sends high-rate flows with a specified rate γ_{large} (e.g. γ_h). Each high-rate flow is generated as follows. We randomly choose a 1-second time slot within the 30-second stream as the first second of the flow. Starting from that second, we randomly generate $\gamma_{large}/packetSize$ packets in each 1-second interval to make the flow size in each interval equal to that specified rate. In this experiment, we set the packet size to 1518 bytes, the maximum packet size. In a Shrew attack, the attacker sends periodic bursts in an attempt to cut off TCP traffic by exploiting TCP's congestion control mechanism. To generate a bursty flow with a period T, burst duration L, and bursty flow rate γ_{burst}, we randomly choose a time point from $[0, 29)$ second as the start time, and then randomly generate $\gamma_{burst} \cdot L$ packets in each L-length burst that occurs every T seconds. We then evaluate (1) how many malicious large flows can evade detection, and (2) how many small benign flows are falsely caught because of these coexisting attack flows.

We configure EARDET based on the guidelines in Section 4.6, such that it can detect large flows violating $TH_h(t) = \gamma_h\, t + \beta_h$, where $\gamma_h = 0.01\rho$, and $\beta_h = 2\beta_{TH} + \alpha$. β_{TH} will be calculated according to our technical report [37]. We also consider small flows that comply with $TH_\ell(t) = \gamma_\ell\, t + \beta_\ell$, where $\beta_\ell = 6072$ bytes and $\gamma_\ell = 0.01\rho$. Also, we require t_{upincb} to be smaller than 1 second. Table 5 summarizes the value of each parameter used in our experiments. In a "non-congested link" setting, a fixed number of attack flows are mixed with the real trace. We also consider a "congested link" setting, where we fill the link with attack flows, for the small dataset (i.e., the Federico II dataset) only. We leave it as future work to scale our attack flow generation tool to work for larger datasets.

To configure the two multistage filters (FMF and AMF), we set FMF's window size to 1 second, number of stages $d = 2$, number of counters in each stage $b = 250$, threshold of FMF $T = \gamma_h \cdot 1$ second, threshold of AMF $u = \beta_h$, and drain rate $r = \gamma_h$. We are also interested in the performance of FMF and AMF when their memory is as small as EARDET's. Hence, we run additional experiments in which the number of counters in each stage is 55. The details of these values are shown in the Table 6.

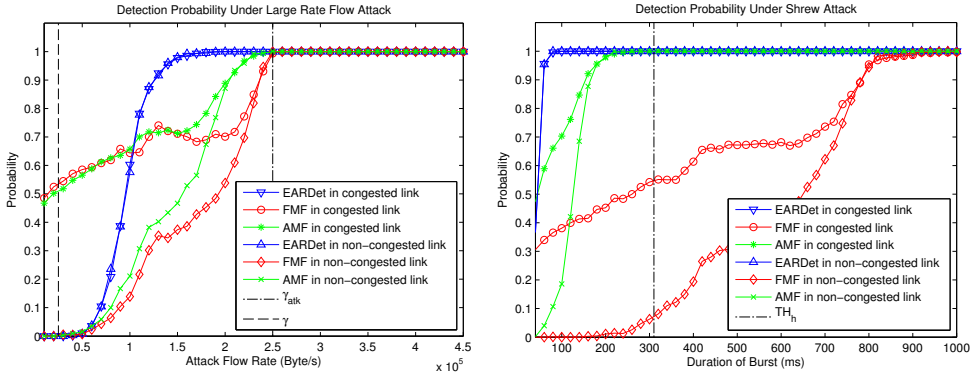
For each experiment environment, we design two sets of experiments to test the performance of these three filters in the presence of flooding attacks and Shrew attacks. We repeat each experiment for 10 times and present the average. In the case of flooding attacks, we randomly generate k_1 attack flows for each attack rate. In the case of Shrew attacks, we randomly generate k_2 bursty flows with $1.2 * \gamma_h$ burst rate and 1-second period for each burst duration L. We set $k_1 = k_2 = 50$ for the non-congested link setting, and set the k_1 and k_2 as large as possible to congest the link in the congested-link setting.

Table 6: Multistage Filter Parameters.

Dataset	$b * d$	T	u	r
Federico II	$55 * 2, 250 * 2$	$250KB$	$15.5KB$	$250KB/s$
CAIDA	$55 * 2, 250 * 2$	$12.5MB$	$15.4KB$	$12.5MB/s$

Table 5: Parameters of Experiment Environment.

Dataset	γ_h	β_h	γ_ℓ	β_ℓ	ρ	α	link status	β_{TH}	n	t_{upincb}
Federico II	$250KB/s$	$15.5KB$	$25KB/s$	$6072B$	$25MB/s$	$1518B$	congested/non-congested	$6991B$	107	$0.8370s$
CAIDA	$12.5MB/s$	$15.4KB$	$1.25MB/s$	$6072B$	$1.25GB/s$	$1518B$	non-congested	$6925B$	100	$0.1242s$



(a) Flooding DoS Attack (b) Shrew DoS Attack

Figure 5: Detection Probability in Experiment with 55*2 counters in Multistage Filter.

Evaluation metrics. We consider three evaluation metrics: detection probability, false positive probability on small flows, and incubation period. Detection probability is the probability to successfully detect a generated attack flow. False positive probability on small flows is the probability to wrongly catch a small flow when the link is attacked by attack flows of a certain rate. Incubation period represents the time needed to catch a generated attack flow since it is generated.

5.3 Experimental Comparison

Since the results for the two datasets are similar, we omit CAIDA's results due to space constraints.

Figure 5(a) and 5(b) shows the detection probability in the face of different types of attack flows. We focus on the scenario of using $55 * 2$ counters in FMF and AMF, as the results of $250 * 2$ counters are similar. In Figure 5(b), the TH_h line indicates whether a bursty flow exceeds the high-bandwidth threshold. The results show that EARDet detects attack flows with a 100% detection probability, which confirms Theorem 4. On the contrary, Figure 5(b) shows that FMF cannot catch most of the Shrew flows. Moreover, EARDet can catch most of the attack flows in the ambiguity region (between $TH_\ell(t)$ and $TH_h(t)$).

Figure 6(a) to Figure 6(h) show the results of FP_s rates. While EARDet has zero FP_s in any case as expected, Figure 6(a) to Figure 6(d) shows that both FMF and AMF have high FP_s rates in both attack scenarios when using very limited memory as EARDet. That is, the attacker can successfully incriminate benign small flows. Worst yet, when the link is congested by attack flows, the FP_s rate can be as high as 4% for FMF and 1% for AMF under flooding attacks, and for FMF, the FP_s rate is also extremely high under Shrew attacks. An interesting observation is that, in Figure 6(a), both FMF and AMF have a higher FP_s rate when the link is congested by malicious small flows. As Figure 6(e) to 6(h) show, using more counters in FMF and AMF can reduce, but not eliminate, the FP_s rates. The results of the CAIDA dataset exhibit similar trends.

Also, in our experiments, EARDet always produces similar results no matter whether the link is congested. In contrast, the results of AMF and FMF are much different between the congested-link version and non-congested-link version. This supports the conclusion in Section 5.1 that AMF and FMF rely on the number of active flows but EARDet does not. This advantage makes EARDet stable in any networking environment.

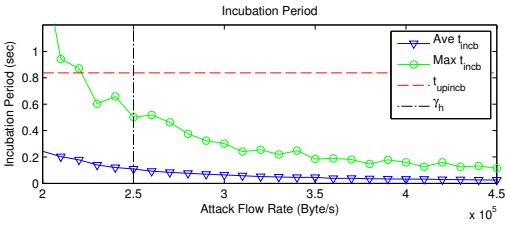
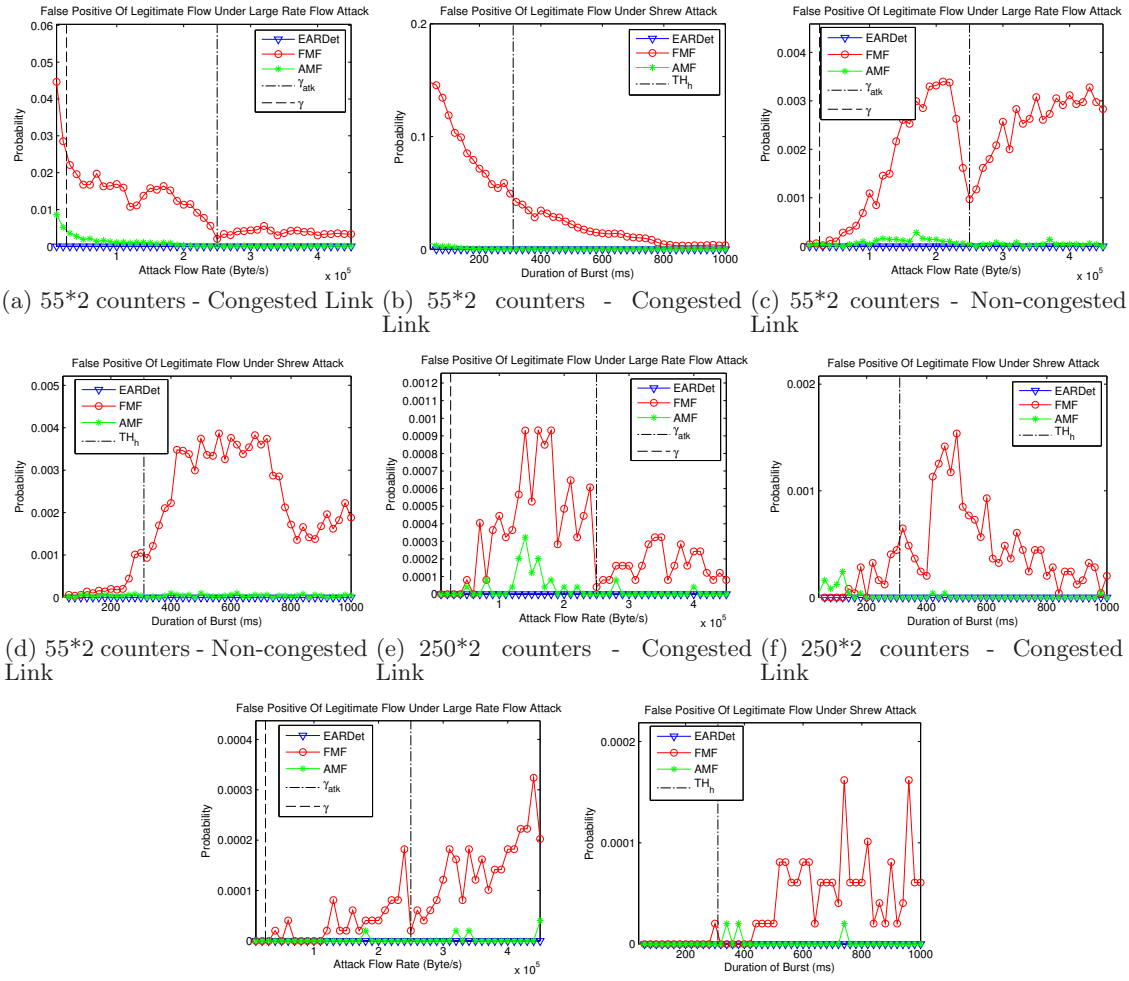


Figure 7: Incubation Period.

Figure 7 describes the maximum and average incubation period of high-rate flows with different rates in flooding attacks. We can find that the maximum incubation period for flows whose rate is over γ_h is always below the theoretical upper bound for the incubation period, t_{upincb}, which supports Theorem 7. Moreover, the average incubation period is much lower than the theoretical upper bound, which shows EARDet's incubation period is much shorter in practice.

6. RELATED WORK

Section 2.1 classifies prior work on detecting large flows and its closely related problem of finding frequent items based on the types of monitoring windows. Cormode and Hadjieleftheriou [9] present a thorough survey and comparison of algorithms for finding frequent items. This section reviews prior approaches based on the techniques used. Most



(a) 55*2 counters - Congested Link (b) 55*2 counters - Congested Link (c) 55*2 counters - Non-congested Link

(d) 55*2 counters - Non-congested Link (e) 250*2 counters - Congested Link (f) 250*2 counters - Congested Link

(g) 250*2 counters - Non-congested Link (h) 250*2 counters - Non-congested Link

Figure 6: False Positive on Small Flows.

of the prior work does not consider the arbitrary window model.

Counter-based techniques. Counter-based techniques maintain a small number of counters, each of which associated with a flow or an item. Manku and Motwani [29] present another well-known counter-based technique called Lossy Counting. For each stored item, Lossy Counting maintains and updates the upper bound and lower bound on the count of the item. The algorithm stores every new item and periodically removes items whose upper bound is less than the threshold. Similar to the MG algorithm discussed in Section 3.2, the Space Saving algorithm [30] proposed by Metwally et al. maintains k (item, counter) pairs and increases the corresponding counter of each incoming item. If the new item e is not stored currently, the stored item with the lowest count is replaced by the new item, and the counter increases accordingly.

Sketch-based techniques. Multistage filters identify large flows over fixed time windows [17] and over arbitrary windows [16]. Fang et al. [18] propose a similar multistage algorithm but their algorithm requires more than one pass over the input stream. Cormode and Muthukrishnan [10] present a novel data structure called count-min sketch, which summarizes an input stream and can answer several queries including finding frequent items. As pointed out in their paper, despite the fact that the construction is similar to that of multistage filters, count-min sketches can flexibly support negative weights and require only pairwise independence hash functions rather than fully independent ones. In general, sketches can support a richer set of queries with a higher memory overhead compared with counter-based techniques.

Sampling-based techniques. Sampled NetFlow [1] maintains a generic traffic summary of sampled packets. With a sampling rate $1/r$, the frequency estimate is derived by multiplying the count by r. To improve the accuracy of the estimates, both Sticky sampling [29] and Sample and Hold [17] examine every incoming item and increase the corresponding count if the item is being monitored. If the new item is not being monitored, it is sampled and added to the monitoring list with a certain probability. Sampling-based techniques in general cannot achieve high accuracy due to

the lack of per-packet information. Duffield [15] studies how to perform fair sampling in traffic flow measurements.

Calders et al. [7] define a new frequency measure as the maximum frequency over all possible windows ending at the current time. Although their core idea is the same as the arbitrary window model, their algorithm focuses on accurately estimating the frequency based on the new frequency measure, whereas we seek to accurately identify large flows.

7. CONCLUSION

EARDet is a deterministic streaming algorithm that robustly catches all large flows and protects all small flows regardless of the traffic distribution. The core ideas differentiating EARDet from prior work are that it monitors flows over arbitrary windows and provides exactness outside an ambiguity region. We believe that EARDet can aid emerging applications such as detecting DoS attacks by bursty flows [25] and enforcing QoS-based SLA compliance [35], which require robust monitoring for high assurance. One future direction is to explore the design space of large-flow algorithms in the arbitrary window model by applying the no-FP and no-FN theorems to existing landmark-window-based algorithms. Another interesting future work is to formally examine the robustness of EARDet and prior algorithms against malicious inputs.

8. ACKNOWLEDGMENTS

We gratefully thank our shepherd Darryl Veitch and the anonymous reviewers for their valuable feedback. This research was supported in part by NSF under grants CNS-0953600, CNS-1248080, and CNS-1040801.

9. REFERENCES

[1] Random Sampled NetFlow. `http://www.cisco.com/en/US/docs/ios/12_0s/feature/guide/nfstatsa.html`.

[2] The CAIDA UCSD Anonymized Internet Traces 2012 - 1220. `http://www.caida.org/data/passive/passive_2012_dataset.xml`.

[3] Traces 1 of TCP port 80 traffic traces from Federico II. `http://traffic.comics.unina.it/Traces/ttraces.php`.

[4] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker. Accountable Internet Protocol (AIP). In *Proceedings of ACM SIGCOMM*, 2008.

[5] A. Arasu and G. S. Manku. Approximate Counts and Quantiles over Sliding Windows. In *Proceedings of ACM PODS*, 2004.

[6] B. Boyer and J. Moore. A Fast Majority Vote Algorithm. Technical report, ICSCA-CMP-32, Institute for Computer Science, University of Texas, 1981.

[7] T. Calders, N. Dexters, and B. Goethals. Mining Frequent Items in a Stream Using Flexible Windows. *Intelligent Data Analysis*, 12(3):293–304, 2008.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*, chapter 13: Red-Black Trees. MIT Press and McGraw-Hill, 2001.

[9] G. Cormode and M. Hadjieleftheriou. Finding Frequent Items in Data Streams. *Proc. VLDB Endow.*, 1(2):1530–1541, 2008.

[10] G. Cormode and S. Muthukrishnan. An Improved Data Stream Summary: The Count-Min Sketch and its Applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[11] S. A. Crosby and D. S. Wallach. Denial of Service via Algorithmic Complexity Attacks. In *Proceedings of USENIX Security*, 2003.

[12] A. Dainotti, A. Pescapè, P. Salvo Rossi, F. Palmieri, and G. Ventre. Internet Traffic Modeling by means of Hidden Markov Models. *Computer Networks (Elsevier)*, 52:2645–2662, 2008.

[13] A. Dainotti, A. Pescapè, and G. Ventre. A Cascade Architecture for DoS attacks Detection based on the Wavelet Transform. *Journal of Computer Security*, 17(6/2009):945–968, 2009.

[14] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Frequency Estimation of Internet Packet Streams with Limited Space. In *Proceedings of ESA*, 2002.

[15] N. Duffield. Fair Sampling Across Network Flow Measurements. In *Proceedings of ACM SIGMETRICS*, 2012.

[16] C. Estan. *Internet Traffic Measurement: What's Going on in my Network?* PhD thesis, 2003.

[17] C. Estan and G. Varghese. 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.

[18] M. Fang and N. Shivakumar. Computing Iceberg Queries Efficiently. In *Proceedings of VLDB*, 1999.

[19] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827 (Best Current Practice), May 2000. Updated by RFC 3704.

[20] M. Fischer and S. Salzberg. Finding a Majority Among N Votes: Solution to Problem 81-5. *Journal of Algorithms - JAL*, 3(4):362–380, 1982.

[21] L. Golab, D. DeHaan, E. D. Demaine, A. López-Ortiz, and J. I. Munro. Identifying Frequent Items in Sliding Windows over On-Line Packet Streams. In *Proceedings of ACM IMC*, 2003.

[22] M. Guirguis, A. Bestavros, and I. Matta. Exploiting the Transients of Adaptation for RoQ Attacks on Internet Resources. In *Proceedings of IEEE ICNP*, 2004.

[23] R. M. Karp, S. Shenker, and C. H. Papadimitriou. A Simple Algorithm for Finding Frequent Elements in Streams and Bags. *ACM Transactions on Database Systems*, 28(1):51–55, 2003.

[24] T. Kim, C. Basescu, L. Jia, S. B. Lee, Y.-C. Hu, and A. Perrig. Lightweight Source Authentication and Path Validation. In *Proceedings of ACM SIGCOMM*, 2014.

[25] A. Kuzmanovic and E. Knightly. Low-Rate TCP-Targeted Denial of Service Attacks and Counter Strategies. *IEEE/ACM Transactions on Networking*, 14(4):683–696, 2006.

[26] L. Lee and H. Ting. A Simpler and More Efficient Deterministic Scheme for Finding Frequent Items over Sliding Windows. In *Proceedings of ACM PODS*, 2006.

[27] X. Liu, A. Li, X. Yang, and D. Wetherall. Passport: Secure and Adoptable Source Authentication. In *Proceedings of USENIX/ACM NSDI*, 2008.

[28] S. Machiraju, M. Seshadri, and I. Stoica. A Scalable and Robust Solution for Bandwidth Allocation. In *Proceedings of IEEE International Workshop on Quality of Service*, volume 00, 2002.

[29] G. Manku and R. Motwani. Approximate Frequency Counts over Data Streams. In *Proceedings of VLDB*, 2002.

[30] A. Metwally, D. Agrawal, and A. El Abbadi. Efficient Computation of Frequent and Top-k Elements in Data Streams. In *Proceedings of ICDT*, 2005.

[31] J. Misra and D. Gries. Finding Repeated Elements. *Science of Computer Programming*, 2(2):143–152, 1982.

[32] J. Naous, M. Walfish, A. Nicolosi, D. Mazieres, M. Miller, and A. Seehra. Verifying and Enforcing Network Paths with ICING. In *Proceedings of ACM CoNext*, 2011.

[33] S. Shenker, C. Partridge, and R. Guerin. Specification of Guaranteed Quality of Service. RFC 2212 (Proposed Standard), Sept. 1997.

[34] I. Stoica, S. Shenker, and H. Zhang. Core-Stateless Fair Queueing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High-Speed Networks. *IEEE/ACM Transactions on Networking*, 11(1):33–46, Feb. 2003.

[35] D. M. Turner, V. Prevelakis, and A. D. Keromytis. A Market-Based Bandwidth Charging Framework. *ACM Transactions on Internet Technology*, 10(1):1–30, 2010.

[36] J. Turner. New Directions in Communications (Or Which Way to the Information Age?). *IEEE Communications Magazine*, 24:8–15, 1986.

[37] H. Wu, H.-C. Hsiao, and Y.-C. Yu. Efficient Large Flow Detection over Arbitrary Windows: An Algorithm Exact Outside An Ambiguity Region. Technical report, CMU-CyLab-14-006, CyLab, Carnegie Mellon University, 2014.

[38] M. Yu, L. Jose, and R. Miao. Software Defined Traffic Measurement with OpenSketch. In *Proceedings of USENIX NSDI*, 2013.

APPENDIX

A. ENGINEERING THE DETECTOR: SOLUTION AND ANALYSIS

We give a detailed solution and analysis to the problem defined in Section 4.6. The problem can be expressed by the inequality set (5):

$$\begin{cases} \dfrac{\alpha+2\beta_{TH}}{\gamma_h-\frac{\rho}{n+1}} < t_{upincb} \\ \dfrac{\beta_\Delta}{\alpha(n-1)+(n+1)\beta_\ell+(n+1)\beta_\Delta}\cdot\rho > \gamma_\ell \\ \dfrac{\rho}{n+1} < \gamma_h \end{cases} \tag{5}$$

$$\Leftarrow \begin{cases} \dfrac{2(\alpha+\beta_{TH})}{\gamma_h-\frac{\rho}{n+1}} \le t_{upincb} \\ \dfrac{\beta_\Delta}{\alpha+\beta_\ell+\beta_\Delta}\cdot\dfrac{\rho}{n+1} \ge \gamma_\ell \\ \dfrac{\rho}{n+1} < \gamma_h \end{cases} \tag{6}$$

$$\Leftrightarrow \begin{cases} \beta_\Delta \le \dfrac{t_{upincb}(\gamma_h-\frac{\rho}{n+1})-2(\alpha+\beta_\ell)}{2} \\ \beta_\Delta \ge \dfrac{\gamma_\ell(\alpha+\beta_\ell)}{\frac{\rho}{n+1}-\gamma_\ell} \\ n > \dfrac{\rho}{\gamma_h}-1 \end{cases} \tag{7}$$

$$\Rightarrow \begin{cases} \dfrac{\gamma_\ell(\alpha+\beta_\ell)}{\frac{\rho}{n+1}-\gamma_\ell} \le \dfrac{t_{upincb}(\gamma_h-\frac{\rho}{n+1})-2(\alpha+\beta_\ell)}{2} \\ n > \dfrac{\rho}{\gamma_h}-1 \end{cases} \tag{8}$$

We can assert that there must exist a solution pair of (n, β_Δ) which fulfills the inequality set (7), if and only if there is a n satisfying $n_{min} \le n \le n_{max}$, where

$$\begin{cases} n_{min} = \lceil \rho/\dfrac{M+\sqrt{M^2-4\gamma_h\gamma_\ell}}{2} \rceil - 1 \\ n_{max} = \lfloor \rho/\dfrac{M-\sqrt{M^2-4\gamma_h\gamma_\ell}}{2} \rfloor - 1 \\ M = \gamma_h+\gamma_\ell-\dfrac{2(\alpha+\beta_\ell)}{t_{upincb}} \ge 0 \end{cases} \tag{9}$$

Figure 8 illustrates the solution space. In this figure, the (n, β_Δ) solution pairs are in the space between the two lines of the lower bound curve and the upper bound curve. Note that the inequality sets (5) and (7) are not totally equal, so there may be additional solutions outside this space.

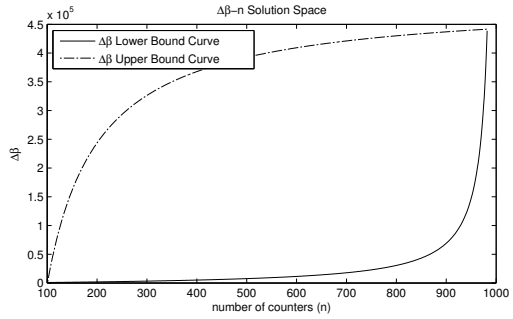


Figure 8: Curve of the lower bound of β_ℓ. (γ_ℓ = 100,000 byte/s γ_h = 1,000,000 byte/s, ρ = 100,000,000 byte/s, α = 1518 bytes, β_ℓ = 6072 bytes, t_{upincb} = 1 second.)

According to the inequality set (7), the lower bound of β_Δ is $\beta_{\Delta\,min} = \frac{\gamma_\ell(\alpha+\beta_\ell)}{\frac{\rho}{n+1}-\gamma_\ell}$. We can see $\beta_{\Delta\,min}$ increases with n, as Figure 8 shows. Since we can reduce memory consumption and the burst gap by using a smaller n and β_Δ, we choose

$$\begin{cases} n = n_{min} = \lceil \rho/\dfrac{M+\sqrt{M^2-4\gamma_h\gamma_\ell}}{2} \rceil - 1 \\ \beta_{TH} = \beta_\ell + \beta_{\Delta\,min} = \beta_\ell + \dfrac{\gamma_\ell(\alpha+\beta_\ell)}{\frac{\rho}{n_{min}+1}-\gamma_\ell} \end{cases} \tag{10}$$

as our final answer to the design problem.

We give a numerical example showing how to configure EARDet based on the result above. Suppose the administrator of the detector chooses $\gamma_\ell = 100KB/s$, $\gamma_h = 1MB/s$, $\rho = 100MB/s$, α = 1518 bytes, β_ℓ = 6072 bytes, t_{upincb} = 1 second. Then using Equation (10), we choose $n = 101$ and β_Δ = 863 bytes. With these two parameters, the incubation period is 0.7848 seconds which is smaller than $t_{upincb} = 1$ second, and the no false positive rate is 100450 byte/s which is larger than γ_ℓ = 100000. The ratio between no false negative rate $\frac{\rho}{n+1}$ and low-bandwidth threshold rate γ_ℓ is $\frac{\rho}{n+1}/\gamma_\ell = 9.80$. The results show that to achieve quick reaction to large flows and a small rate gap, the detector only needs a small number of extra counters compared with the minimum number of required counters (i.e. $\frac{\rho}{\gamma_h}-1 = 99$) and a low burst gap.

We obtain this particular solution by choosing the minimum n and minimum β_Δ. We can also solve the inequality set (5) for different requirements, such as minimizing the rate gap between $\frac{\rho}{n+1}$ and $\frac{\beta_\Delta}{\alpha(n-1)+(n+1)\beta+(n+1)\beta_\Delta}\cdot\rho$.

However, the inequality set (7) may be unsolvable for some $\rho, \gamma_\ell, \beta_\ell, \gamma_h, \alpha, t_{upincb}$. To make it solvable, we need to make sure $M^2 - 8\gamma_h\gamma_\ell$ and M are not negative in the inequality set (9). Namely,

$$\gamma_h+\gamma_\ell-\frac{2(\alpha+\beta_\ell)}{t_{upincb}} \ge \sqrt{4\gamma_h\gamma_\ell} \tag{11}$$

$$\Leftarrow t_{upincb} \ge \frac{2(\alpha+\beta_\ell)}{\gamma_h+\gamma_\ell-2\sqrt{\gamma_h\gamma_\ell}} \tag{12}$$

Moreover, according to Section 4.3, $\gamma_h > \gamma_\ell$ is necessary to make the inequality set (7) solvable.