View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Illinois Digital Environment for Access to Learning and Scholarship Repository

© 2017 Hao Wu

SITE RELIABILITY AGAINST ANOMALOUS BEHAVIORS

BY

HAO WU

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Doctoral Committee:

Associate Professor Yih-Chun Hu, Chair
Associate Professor Nikita Borisov
Associate Professor Michael Bailey
Assistant Professor Hsu-Chun Hsiao, National Taiwan University

ABSTRACT

Many attacks that threaten service providers and legitimate users are anomalous behaviors out of specification, and this dissertation mainly focuses on detecting "large" Internet flows consuming more resources than those allocated to them. Being able to identify large flows accurately can greatly benefit Quality of Service (QoS) schemes and Distributed Denial of Service (DDoS) defenses. Although large-flow detection has been previously explored, proposed approaches have not been practical for high-capacity core routers due to high memory and processing overhead. Additionally, more efficient schemes are vulnerable against specially tailored attacks in which attackers time their packets based on the knowledge of legitimate cross-traffic.

In this dissertation, we aim to design computation- and memory-efficient large-flow detection algorithms to effectively mitigate the large-flow damage in adversarial environments. We propose three large-flow detection schemes: Exact-Outside-Ambiguity-Region Detector (EARDet), Recursive Large-Flow Detection (RLFD), and the scheme of in-Core Limiting of Egregious Flows (CLEF), which is a hybrid scheme with one EARDet and two RLFDs. EARDet is a deterministic algorithm that guarantees exact large-flow detection outside an ambiguity region: there is no false accusation for legitimate flows complying with a low-bandwidth threshold, and no false negative for large flows above a high-bandwidth threshold, with no assumption on the input traffic or attack patterns. Because of the strong enforcement with the arbitrary window model, EARDet is able to immediately detect both flat and bursty flows. RLFD is designed to complement EARDet in detecting large flows in EARDet's ambiguity region. RLFD is a probabilistic detection scheme that gives higher probability for detecting large flows with higher volume, thus guarantee limited damage (to legitimate flows) across a wide range of flow overuse amounts. Finally CLEF combines EARDet and RLFD to achieve both rapid detection for very large flows and eventually detection for small, persistent large flows. Theoretical analysis and experimental evaluation both suggest the CLEF's efficiency and effectiveness outperform existing algorithms.

To my family, for their unconditional love and support.

ACKNOWLEDGMENTS

I would like to express sincere appreciation to my advisor Professor Yih-Chun Hu for his help and advice in developing the work and writing this dissertation. I feel so lucky to meet such a smart and nice advisor who guides me and leads me to act as an engineer and a researcher. All skills learnt from him will definitely benefit me in my future career.

Then, I want to give special thanks to Professor Hsu-Chun Hsiao and Professor Adrian Perrig, who have worked together with me on large-flow detection during my Ph.D. study. Their suggestions and help provided great venues to overcome those hard problems in developing EARDet, RLFD, and CLEF.

I am also grateful for my doctoral committee members Professor Nikita Borisov and Professor Michael Bailey, who gave me unvaluable feedback and guidance on my dissertation work.

Finally, many thanks to my family who supported and helped me finish my Ph.D. degree.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

AMF	Arbitrary-window-based Multistage Filter
AMF-FM	Hybrid scheme of AMF and FM
CLEF	in-Core Limiting of Egregious Flows
(D)DoS	(Distributed) Denial of Service
EARDet	Exact-Outside-Ambiguity-Region Detector
FM	Flow Memory
FMF	Fixed-window-based Multistage Filter
FN	False Negative for large misbehaving flows
FP	False Positive for legitimate flows
MG	Misra-Gries
No-FN	No False Negative for large misbehaving flows
No-FP	No False Positive for legitimate flows
No-FN$_\ell$	No False Negative for large flows in EARDet
No-FP$_s$	No False Positive for small flows in EARDet
QoS	Quality of Service
RLFD	Recursive Large-Flow Detection
Twin-RLFD	Hybrid scheme of two RLFD instances
V.C.	Virtual Counter in RLFD

CHAPTER 1

INTRODUCTION

Many attacks that threaten the service providers and the legitimate users in the Internet are anomalous behaviors that are different from the legitimate behaviors in the Internet. At the network level, the attack traffic has patterns and features that differ from legitimate traffic. For example, the Denial of Service (DoS) attacks flood the targeted machine or resource to cause service with a huge amount of traffic to overload systems and prevent legitimate traffic from being processed.

In this dissertation, we focus on detecting misbehaving "large" network flows[1] that use more than their allocated resources. Large-flow detection is not only an important mechanism for Quality of Service (QoS) [3] schemes such as IntServ [4], but also for DDoS defense mechanisms that allocate bandwidth to network flows [5–7]. With the recent emergence of volumetric DDoS attacks, 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 [2]. Large-scale Internet core routers have an aggregate capacity of several Terabits per second (Tbps), which demands highly efficient schemes to detect flows that violate their flow specifications. The general approach to catch misbehaving flows without per-flow counters is to assign flows to a traffic class with a maximum sending rate and to embed the traffic class in the packet header. A router on the path can then detect misbehaving flows by finding the largest ones within a traffic class. In

This dissertation reuses some parts including text and figures from Wu, H. et al. "Efficient Large Flow Detection over Arbitrary Windows: An Algorithm Exact Outside An Ambiguity Region", IMC '14 Proceedings of the 2014 Conference on Internet Measurement Conference, pp. 209-222 [1], ©2014 Association for Computing Machinery, Inc. Reprinted by permission. http://doi.acm.org/10.1145/2663716.2663724.

[1]As in prior literature [1,2], the term "large flow" denotes a flow that sends more than its allocated bandwidth.

Chapters 3 and 4, we analyze and design algorithms for *large-flow detection* that can scale to high-capacity core routers.

In Chapter 3, we first 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. Existing large-flow detectors that only check the average throughput over a certain time period cannot detect bursty flows and are therefore easily fooled by attackers. To achieve exactness outside the ambiguity region, we propose a deterministic large-flow detection algorithm, EARDet, that detects *all* large flows (including bursty flows) and avoids false accusation against *any* small flows, regardless of the input traffic. The core idea of EARDet is to monitor flows over arbitrary time windows built on a frequent items finding algorithm based on average frequency. Despite its strong properties, EARDet takes very low storage overhead regardless of input traffic and is surprisingly scalable because it focuses on accurate classification of large flows and small flows only.

In Chapter 4 we propose a novel randomized *Recursive Large-Flow Detection* (RLFD) algorithm to complement EARDet to detect large flows missed by EARDet in the ambiguity region. Unlike existing detectors, RLFD efficiently distinguishes large flows from legitimate flows by evaluating one set of flows at a time, and recursively shrinking the set of suspected large flows. Larger flows are detected with higher probability in RLFD, so the expected detection time decreases in the level of overuse, resulting in limited damage (to legitimate flows) across a wide range of flow overuse amounts. Because the immediate detection of large flows is not possible due to memory constraints, the goal should be to minimize the damage caused to legitimate flows; in such an environment, it is likely more important to rapidly catch very high-rate flows than it is to quickly catch mildly misbehaving flows. Our damage model considers two causes of packet loss: loss caused by large flows, and false positives that punish legitimate flows.

We further propose a hybrid scheme CLEF, short for *in-Core Limiting of Egregious Flows* combining the deterministic EARDet mechanism for rapid detection of very large flows with the RLFD algorithm for eventual detection of large flows. We analyze a range of attacks against select detection schemes and find CLEF provides strong resilience against attacks even in its worst-case background traffic, and enables efficient implementation on high-capacity core routers.

CHAPTER 2

LARGE-FLOW DETECTION BACKGROUND

2.1 Large-Flow Detection Problem

2.1.1 Flow Model

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 several ways: *landmark-window*, in which the flow specification is enforced over a limited set of starting times; *sliding-window*, in which the flow specification is enforced over a sliding time window with fixed length; or *arbitrary-window*, in which the flow specification is enforced over every possible starting time.

Flow identifiers. In general, the information associated with a collection of related traffic acts as the flow identifiers to group traffic into different flows. For Internet flows, the information are commonly in the packet header, e.g., 5-tuple (source / destination IP / port, transport protocol).

In Chapters 3 and 4, we aim to design a generic large-flow detection solution for Internet flows, which can be applied in cases without assumption on flow identifiers.

As in prior work in flow monitoring, we assume each flow has a unique and unforgeable ID, e.g., using source authentication techniques such as accountable IPs [8], ICING [9], IPA [10], OPT [11], Passport [12], or with RPKI [13]. Such techniques can be deployed in the current Internet or in a future Internet architecture, e.g., Nebula [14], SCION [15], or XIA [16].

Internet flow: Packet streams. Internet flows are traffic of packet streams. In the packet space $\mathcal{X}$, we consider that the large-flow detector processes a packet stream $X = \langle x_1, \cdots, x_k \rangle$ coming in sequence through a link with capacity ρ, where $x_i \in \mathcal{X}\ \forall i = 1 \cdots k$. Due to the high capacity of the link and the limit of the memory of the detector, the detector can only process the packets once (i.e. only making one pass over the packet stream).

For a packet x, we make the following denotation for later discussion. The time at which the large-flow detector observes the packet x is denoted as $\mathsf{time}(x)$; the flow ID of the packet x is denoted as $\mathsf{fid}(x)$; and the size of packet x is denoted as $\mathsf{size}(x)$. Then we denote the traffic volume of a flow f during time $[t_1, t_2)$ as $\mathsf{vol}(f, t_1, t_2) \triangleq \sum_{x \in \mathcal{X}, \mathsf{fid}(x)=f, t_1 \leq \mathsf{time}(x) < t_2} \mathsf{size}(x)$.

2.1.2 Large Flows and Legitimate Flows

The large flow here is the flow which occupies high bandwidth or consumes a large volume of link bandwidth over some short time window. Therefore, we defined a threshold function $\mathsf{TH}(t_2 - t_1)$ for the limit of bandwidth. The t_2 and t_1 in the function indicate that the function only depends on the length of the time window $[t_2, t_1)$.

For a flow f, if there exists a time window $[t_1, t_2)$ over which the volume of flow $\mathsf{vol}(f, t_1, t_2)$ exceeds a threshold function $\mathsf{TH}(t_2 - t_1)$, then the flow f is classified as large flow; otherwise, the flow f is considered as legitimate flow. Namely, (i) when $\mathsf{vol}(f, t_1, t_2) > \mathsf{TH}(t_2 - t_1)$, f is a large flow; conversely, (ii) when $\mathsf{vol}(f, t_1, t_2) \leq \mathsf{TH}(t_2 - t_1)$, f is considered as a legitimate flow.

Leaky bucket model. Ideally, people want to define large flow based on the leaky bucket model. The leaky bucket model is widely used in the packet switched computer network for checking the traffic of data packets and defining the bandwidth limits and burstiness. In the leaky bucket model, the bucket is actually a counter with as fixed rate to decrease its value when the counter is larger than zero. When new packets of a flow arrive at the bucket, it increases the value by the volume of the packets and checks whether the value exceeds the threshold of the leaky bucket. If the threshold is exceeded, then the flow exceeds the bandwidth limit, i.e. the decreasing rate of the bucket.

Then, the threshold function based on the form of the leaky bucket descriptor is: $\mathsf{TH}(t) = \gamma\, t + \beta$, where $\gamma > 0$, $\beta > 0$. The γ and β here are the decrease rate and threshold of the leaky bucket. However, utilizing the leaky bucket algorithm to check large flow is impractical. This is because network links contain numerous flows and usually run at high speed (e.g. the rate of backbone line is above gigabytes/s), it is very hard to keep the per-flow state as the leaky bucket model does. Thus, catching the large flow defined by the leaky bucket model is challenging.

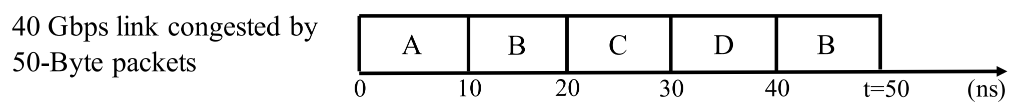
2.1.3 Time Window Models

The time window $[t_1, t_2)$ is a range of time over which the large-flow detector considers the volume of the flow. For example, in some approaches, if the volume of the large flow in $[t_1, t_2)$ exceeds some threshold, then, it is judged as large flow. To identify the large flows defined by the leaky bucket model, the algorithm has to use the arbitrary window model [17]. However achieving the arbitrary window model in practice is challenging, therefore, people usually use some approximate approaches to roughly identify large flows. Thus we have two more typical time window models: the landmark window model [2, 18–24] and the sliding window model [25–27].

Landmark window model. The landmark window model takes the closest landmark in the past as the starting time and the current time as the ending time for each time window (e.g. the landmark could be placed in each 10 seconds). In other words, the landmark window model checks every time window in $\{[t_i, t_i + \Delta_i) | \Delta_i < t_{i+1} - t_i\}$.

Sliding window model. The sliding window model considers the recent traffic as more important than the old traffic, thus the time window starts at some recent time in the past and ends at the current time. Once a new packet arrives, the sliding window model will exclude the oldest packet and keep the newest one. We can state the sliding time window as $\{[t - \Delta, t) | t \in \mathbb{R}\}$.

Arbitrary window model. The arbitrary window model is the stronger time window model to detect the large flows. It monitors each possible time-scales that starts at every instant in time and ends at the current time. Namely, for a flow f, the arbitrary window model monitors it over windows $\{[t_1, t_2) | \forall\, t_1, t_2 \in \mathbb{R}, t_1 < t_2\}$. Therefore, it is more difficult for large flows to evade the detection in front of the arbitrary window model, as demonstrated



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 2.1: In this example, if a flow's volume in time window w with any size is larger than 40 Mbps$\cdot w + 500$ Kb, then it is a large flow. The flow B exceeds the threshold over time window $w = [10, 50)$, however, only the arbitrary time window can catch it [1].

in Figure 2.1.[1]

2.2 Related Work

In this section, we review prior works by the techniques they use in the algorithm as presented in the survey by Cormode and Hadjieleftheriou [28]: counter-based technique, sketch-based technique, and sampling-based technique. We pick and summarize some typical algorithms in each category and discuss their drawbacks.

2.2.1 Counter-Based Algorithm

There are many counter-based algorithms working to find the frequent item, which is closely related to our large-flow identification problem. In a stream with m items, the frequent item is the item that presents more than $\frac{m}{n+1}$ times, where the n is the number of counters. The Misra-Gries (MG) algorithm [18] takes a stream of items as input and find the set of frequent items exactly. The MG algorithm extends the majority algorithm [29, 30], which only considers finding the majority vote.

[1]Figure 2.1 is taken from the paper written by Wu et al. [1] ©2014 Association for Computing Machinery, Inc. by permission.

2.2.2 Sketch-Based Algorithm

The sketch-based algorithm takes a stream as input, applies linear projection or hashing on the input, and produces a matrix. The matrix usually consists of a small number of bits.

Fixed-window-based Multistage Filters (FMF). A multistage filter algorithm is proposed by Estan and Varghese [2] to detect large flows over the fixed window model[2] (called FMF in this chapter). The FMF has multiple stages, and each of the stages contains an array with the same number of counters. When a packet arrives at the multistage filter, its flow identifier is hashed to one counter in each stage (each stage has a different hash function). For each counter, the value increases by the size of the packet assigned to it. Once all of the corresponding counters of a flow f violate the pre-defined threshold, the flow f is judged as large flow.

Arbitrary-window-based Multistage Filters (AMF). One of the obvious drawbacks of FMF is that the fixed window model cannot catch the bursty flow[3] spanning two measurement intervals. To address this, Estan [17] proposed an improved algorithm of multistage filters based on the arbitrary window model. The counters in each stage are replaced by leaky buckets according to the large flow threshold (i.e. $\mathsf{TH}(t) = \gamma t + \beta$). The same applies to FMF, a flow is judged as a large flow if its corresponding leaky buckets are all violated.

2.2.3 Sampling-Based Algorithm

By sampling the packets in the link, the overhead of the algorithm can be reduced effectively. The Sampled NetFlow [31] is a classic sampling-based algorithm which samples packets with a rate of $1/\gamma$ and estimates the frequency of flows by multiplying the count by γ. To improve the Sampled NetFlow, Estan and Varghese [2] propose the sample and hold method which examines every incoming packets: if the flow of the packet is monitored, then increase the corresponding count; otherwise add the flow of the packet into the monitoring list with certain probability.

[2]The fixed window model is a special case of the landmark window model with a fixed measurement interval.

[3]Bursty flow is a kind of large flow which sends very high volume traffic in a short time.

2.2.4 Top-k Detection

Top-k heavy hitter algorithms can be used to identify flows that use more than $1/k$ of bandwidth. The space saving algorithm [32] finds the top-k frequent items by evicting the item with the lowest counter value. HashPipe [33] improves upon space saving so that it can be practically implemented on switching hardware. However, HashPipe still requires keeping 80 KB 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 10 KB of memory. Tong et al. [34] 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.

2.2.5 Other Related Algorithms

Cardinality estimation. Chen et al. [35] and Xiao et al. [36] 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.

UniMon framework. Liu et al. [37] 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.

2.2.6 Drawbacks of Current Algorithms

We mainly discuss the MG algorithm, FMF, AMF, and sample and hold method. The main drawback among the first three algorithms is that they cannot avoid false accusation on the legitimate flows (non-frequent items). For the MG algorithm, it can make sure all the frequent items are stored in the counter at last, but cannot exclude non-frequent items in the one-pass process. The FMF and AMF are the multistage filters algorithms, whose counter could be shared by both large flows and legitimate flows. With some probability, the hash function could map a large flow and legitimate flow to exactly the same counter in each stage. This problem is the nature of multistage filters and cannot be avoided. In Chapter 4, we even show that the multistage filters suffer very high false positives when the memory resource is limited.

Since the sample and hold algorithm just samples the flows to measure, it may not check some large flows. Therefore the sample and hold algorithm has a false detection rate on the large flows. That is, a large flow could evade detection with some probability.

Because the MG algorithm, sample and hold algorithm, and FMF are based on the landmark window model, they cannot catch bursty flows as Section 2.1.3 illustrated.

Although AMF can guarantee a rate of catching all the large flows by its arbitrary window model, it introduces more false detections on the legitimate flows than FMF. Because the leaky bucket is more sensitive to being violated, there are more flows that could exceed the threshold of the leaky bucket model than the fixed window model.

CHAPTER 3

EXACT LARGE-FLOW DETECTION OUTSIDE AN AMBIGUITY REGION

3.1 Chapter Overview

In this chapter, 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 large flow is defined to be a flow whose volume ever exceeds a high-bandwidth threshold function over any arbitrary window. 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, namely the flows in the ambiguity region. Exactness outside ambiguity region guarantees no missed detection on 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. Furthermore, prior works [2, 17] also involves a region similar to our ambiguity region, but they only provide probabilistic bounds and therefore are not exact outside that 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 a 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:

- *Bandwidth guarantees*: To enforce bandwidth allocation, schemes such as IntServ make impractical assumptions that either every router keeps per-flow state [3] or first-hop routers are trusted to regulate traffic on a per-flow basis [38] on behalf of intermediate routers [39]. Although a scalable and robust approach was proposed [40], it causes collateral damage due to the detection delay caused by recursion; moreover it cannot catch bursty flows. Our efficient identifier with these two new models can help enforce bandwidth limits on flows because of its fast detection with no false accusation on the legitimate small flows, and no false circumvention on large flows including bursty flows.

- *Detecting various DoS flows*: Denial-of-Service (DoS) attacks can use either large attack flows or bursty attack flows; however, most existing algorithms check average throughput, which cannot catch large bursty flows. On the contrary, using the arbitrary window model, the detector can effectively detect DoS by bursty flows.

To the best of our knowledge, none of the existing algorithms provide exactness outside an ambiguity region under the arbitrary window model. Because prior algorithms detect average throughput rather than violations of the arbitrary window model, they cannot detect bursty attacks (e.g. Shrew attack [41]). For example, in a large-flow detection system that resets state and starts a new measurement interval periodically [2], 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 above, randomized algorithms are inherently probabilistic and thus may be unable to provide strong deterministic guarantees. Without exactness outside an ambiguity region, most existing algorithms cannot provide guarantees to protect the legitimate flows. Existing randomized algorithms are not exact, because their detection is probabilistic. As a result, they are often configured to provide a lower detection rate for misbehaving flows in order to reduce the false alarm rate for well-behaved flows.

Besides 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, adversaries can perturb their flows' traffic patterns by varying the size and timing of packets sent by those flows so as to cause algorithmic complexity attacks [42], as many algorithms bound their storage and computational overhead by assuming 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 ambiguity region regardless of the input traffic. Built on the Misra-Gries algorithm (a frequent items finding algorithm based on average frequency) [18], EARDet is a 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 accurate classification of large flows and small flows; it does not aim to estimate flow volumes or identify the medium flows, which several prior approaches achieve. In addition to theoretical analysis, we 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 [41, 43].

Our main contributions in this chapter 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 supports exact detection outside an ambiguity region.
- We rigorously prove the two guarantees – catching all large flows and protecting all small flows – without making assumptions about the input traffic.
- Our numerical analysis shows EARDet can operate at 40 Gbps high-speed links using only hundreds of bytes of on-chip SRAM, substantially smaller than the memory consumption in many existing systems. We also provide guidelines on how to configure EARDet to satisfy requirements of specific applications.
- We compare EARDet with two closely related proposals [2, 17] via comparative analysis and extensive simulations based on real and syn-

thetic 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.

3.2 Problem Definition

To make progress in large-flow detection, our goal is to design an efficient arbitrary-window-based large flow algorithm which is exact outside an ambiguity window. In this section, we present our novel model and clarify our goals.

3.2.1 Exact-Outside-Ambiguity-Region Large-Flow Problem

Large, medium, and small flows. To formulate the large-flow problem that is exact-outside-ambiguity-region, we re-define the flows as follows. For a flow f, it is judged as a *large flow*, if there exists a time window $[t_1, t_2)$ over which the volume of f, $\mathsf{vol}(f, t_1, t_2)$ is higher than the high-bandwidth threshold function $\mathsf{TH}_\mathsf{h}(t_2 - t_1)$; the flow f is judged as a *small flow*, if flow f's volume $\mathsf{vol}(f, t_1, t_2)$ over all the possible time window $[t_1, t_2)$ is lower than a low-bandwidth threshold $\mathsf{TH}_\ell(t_2 - t_1)$. The rest of flows are considered as flows in an ambiguity region, which we call *medium flows*.

Considering the arbitrary window model, we defined the threshold function based on the leaky bucket model: $\mathsf{TH}_\mathsf{h}(t) = \gamma_h t + \beta_h$ and $\mathsf{TH}_\ell(t) = \gamma_\ell t + \beta_\ell$, where $\gamma_h > \gamma_\ell > 0$ and $\beta_h > \beta_\ell > 0$.

Exactness outside an ambiguity region. Instead of considering inefficient exact approaches, we propose a relaxed notion of exactness as follows:

Definition 1 *Given a packet stream, the large-flow problem of exactness outside an ambiguity region 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.*[1]

According to the definition above, we also define a *positive* as a flow that is inserted into $\mathcal{F}$, a *negative* is a flow that is not inserted into $\mathcal{F}$. Therefore, we

[1]The Definition 1 is taken from the paper written by Wu et al. [1] ©2014 Association for Computing Machinery, Inc. by permission.

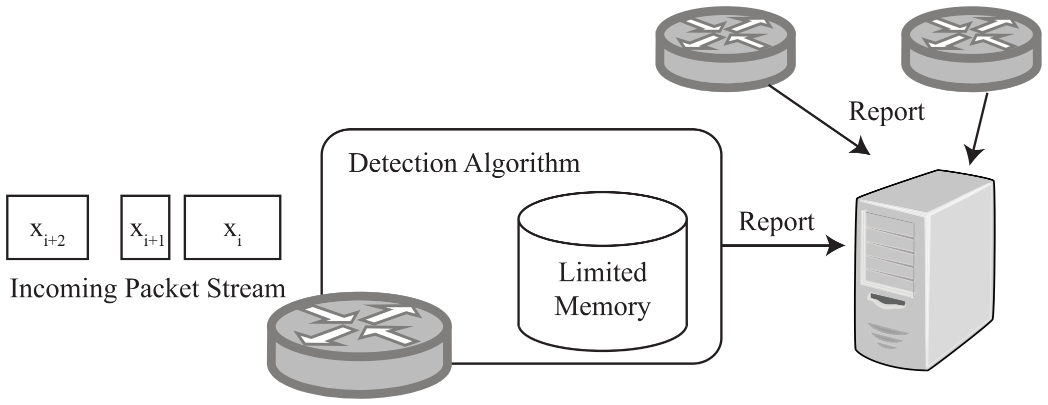


Figure 3.1: A general framework for a large-flow-detection algorithm. The detection algorithm processes incoming flows and keeps limited states in memory. Results may be reported to a remote server for further analysis [1].

have: (1) False Positive of small flows (FP_s) means the detection algorithm added small flows into $\mathcal{F}$ by mistake; and (2) False Negative of large flows (FN_ℓ) means the detection algorithm fails to add large flows into $\mathcal{F}$.

This novel exactness model is reasonable, because the damage caused by large flows is confined by it, and the medium flows can still be handled by existing approaches (e.g. sample and hold algorithm [2], Sampled Netflow [31], etc.).

One thing necessary to mention is that the size of flow set $\mathcal{F}$ is increasing indefinitely over time, thus such a large-flow detection algorithm usually periodically reports results to some report servers with a large amount of storage to maintain a copy of $\mathcal{F}$, as demonstrated in Figure 3.1.[2] Therefore such a large-flow detection algorithm has to correctly make responses without knowledge from the flow set $\mathcal{F}$.

3.2.2 Design Goal

Exactness outside an ambiguity region. We want to design a deterministic large-flow detector which can accurately identify every large flow (including the bursty flow) (i.e. no-FN_ℓ) and never wrongly judges a small flow as a large flow (i.e. no-FP_s) with any input traffic or attack pattern (i.e. we make no assumption on the input traffic).

[2]Figure 3.1 is taken from the paper written by Wu et al. [1] ©2014 Association for Computing Machinery, Inc. by permission.

Scalability. In front of a high rate link, the large-flow detector should maintain a low per-packet operation and small router state so that the algorithm can be implemented in some fast but scarce storage devices (e.g. on-chip cache) regardless of the input traffic and attack pattern.

Fast detection. To minimize the collateral damage, we desire that the large-flow detector can catch the large flow as soon as possible once it violates the high-bandwidth threshold. Thus, for a large flow which violates the high-bandwidth threshold over $[t_1, t_2)$, the detector should be able to detect this flow before an upper bound time $t_2 + t_{process}$, where the $t_{process}$ is the time needed in processing a packet.

3.3 Algorithm

According to design goals described in Section 3.2.2, we proposes EARDet, an arbitrary-window-based algorithm, which resolves the large-flow problem with exactness outside an ambiguity window. Inspired by the MG algorithm [18], EARDet takes the no-FN_ℓ advantage of the MG algorithm and extends it from the landmark window model to the arbitrary window model. Moreover, EARDet achieve the no-FP_s property with only processing packets in one pass. Interestingly, despite such amazing properties achieved by EARDet, it only needs some simple modifications over the original MG algorithm.

3.3.1 EARDet Overview

At the high level, EARDet has the following three main differences compared to the MG algorithm:

Virtual traffic. Different from frequent-item finding, the large-flow problem works on each time slot in the link. Hence, we should not only consider the real packets, but also the idle time gap between two consecutive packets. In EARDet, we virtually fill these idle time gaps with virtual traffic. The virtual flows in the virtual traffic are designed as small flows to avoid unnecessary alarms.

Blacklist. We maintain a local blacklist $\mathcal{L}$ in EARDet to keep the recently identified large flows. The main reason to use the blacklist is to avoid increasing a counter of a flow when the counter value has reached a *counter threshold*, β_{TH}. Once a counter exceeds β_{TH}, EARDet moves the associated flow to the blacklist, and the counter will no longer be updated by the flows in the blacklist. In paper by Wu et al. [1], we have some techniques to bound the size of the blacklist to avoid spending too many resources on blacklist.

Counter threshold. As described above, each counter has a threshold β_{TH} to limit the value. The flows exceeding the threshold will be sent to the blacklist, which enable us to confine the size of each counter by the upper bound of $\beta_{TH} + \alpha$, where the α is the maximum packet size.

3.3.2 Algorithm Description

We show how EARDet works in Algorithm 1.[3] In the algorithm, we treat a packet (including virtual packets) of w size as w uni-size items, and apply a mechanism similar to the one in the MG algorithm to increase and decrease the n counters which are indexed by flow identifiers. There are at most n non-zero counters (the set of non-zero counters is denoted as C), and each counter is at most associated with a flow at the same time. We use S in the algorithm to denote the state of the counters.

To clearly illustrate Algorithm 1, we introduce an example in Figure 3.2[4] to show details of how EARDet updates its status (i.e. counters). At the beginning of the example, there is an empty counter, hence when flow g with a size of 2 arrives, EARDet assigns the empty counter to flow g and increases it by 2. Then, when flow b comes to EARDet, its size is added to the counter associated with flow b, so that the value of flow b's counter violates the threshold hold β_{TH} and flow b is added into blacklist $\mathcal{L}$. At this time, flow b will not be considered for increasing or decreasing the counter anymore. Then, since no empty counters remain, each counter decreases by the size of flow e's packet. At last, EARDet treats the virtual traffic as two packets with a size of 3 and reaches the final state.

[3]This algorithm is taken from the paper written by Wu et al. [1].

[4]Figure 3.2 is taken from the paper written by Wu et al. [1] ©2014 Association for Computing Machinery, Inc. by permission.

Algorithm 1 EARDet [1]

```
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 ($c_f > \beta_{TH}$)
```

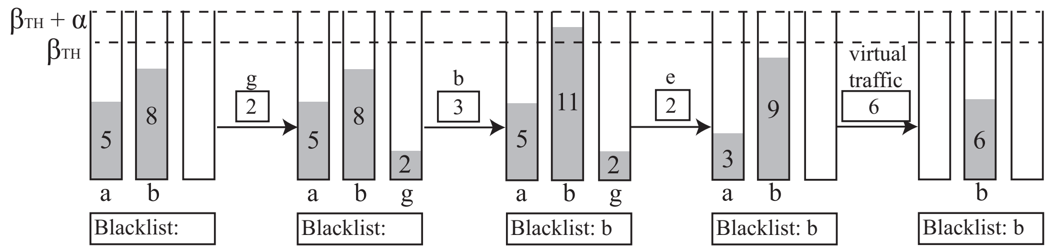


Figure 3.2: Example of updating EARDET status, $\beta_{TH} = 10$, $\alpha = 3$, and $n = 3$ [1].

3.3.3 Optimization in Data Structure

To make EARDET efficient and scalable, we must do some optimization to reduce the counter access. A naive implementation of EARDET has to access each counter once a packet passes through the system, and access each counter numerous times for processing virtual packets if we use 1 byte as the virtual packet size. We are not able to afford such computation consumption in high-speed links, e.g. backbone links. Therefore, we optimize our algorithm as follows.

Balanced binary search tree. To save computation consumption in EARDET, the first thing we need to do is to have a proper data structure which can support insertion, deletion, and retrieval of the minimum counter among all counters. A balanced binary search tree is a good choice in this case, because it can achieve these operations in $O(\log n)$ time.

Float ground for decrement operation. To avoid retrieving and decreasing all counters when one packet arrives, we consider the counter value relative to *floating ground* c_{ground} instead of recording the absolute counter values. In this way, once a packet comes in, we do not have to decrease each counters, but just need to elevate the floating ground. Finally, to judge whether the counter exceeds the threshold, we only need to check whether $c_f - c_{ground} > \beta_{BF}$ is true. To prevent overflow in counters, we periodically reset the floating ground to zero and accordingly reduce the value of each counter.

Efficiently process virtual flows. As mentioned, if we set the packet size too small for flow virtual flows, we are going to update the counters too many times. We noticed that we actually expect to divide virtual traffic to multiple virtual flows to make virtual flows comply with the low-bandwidth threshold (i.e. we will not mistake virtual flows as large flows), meanwhile we want to minimize the packet processing cost for virtual flows.

Thus, the maximum packets size is the counter threshold β_{TH} bytes. Because the counter threshold β_{TH} has to be larger than minimum packet size (i.e. 40 bytes), the overhead is bounded by that of the worst case when the link is congested by minimum-sized packets.

Implement counters with integers. To make the system more efficient, we use integers to implement counters rather than using float numbers. In this way, we not only save storage space but also modify counters faster. However, we should be careful here, because the size of virtual traffic is not always an integer. For example, for a link with 800 Mbps capacity and an idle interval 1 ns, we have 0.1 byte virtual traffic. To handle this issue, we have a little change in our thresholds: EARDET can catch all large flows violating $\mathsf{TH_h}(t) = \gamma_h t + (\beta_h + 1)$ and no false positive for small flows complying $\mathsf{TH}_\ell(t) = \gamma_\ell t + (\beta_h - 1)$. We derive the proof sketch from Wu et al.'s paper [1].

Proof sketch: We bound such biases with a slightly modified algorithm that adjusts virtual traffic. Let us use $\{v_1, v_2, \cdots\}$ to denote the sizes of a sequence of virtual traffic and $\{v'_1, v'_2, \cdots\}$ to denote the adjusted sizes. We maintain an extra field called "carryover", co, which keeps the amount of uncounted virtual traffic. The co is initialized to zero, and we ensure that $-0.5 \leq co < 0.5$ all the 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 catching flows that violate $\mathsf{TH_h}(t) = \gamma_h t + (\beta_h + 1)$ and guarantees not catching any flow that conforms to $\mathsf{TH}_\ell(t) = \gamma_\ell t + (\beta_h - 1)$. [1] ■

Run EARDet in parallel. A straightforward way to scale a large-flow detection algorithm is to parallelize it with multiple detectors. We could randomly distribute the input flows into k EARDET, and each EARDET detector only has approximately $1/k$ of overhead. However, this approach also has some drawbacks: (1) it may not be able to evenly distribute overhead, because $1/k$ flows does not mean $1/k$ packets; and (2) randomness weakens the deterministic property, so attackers could manipulate the flows based on the random seed to escape detection.

3.4 Algorithm Analysis

In this section, we analyzed the unique properties in EARDet, and theoretically proved them. Moreover, we analyzed the tradeoff in the tuning parameters of EARDet. Please refer to the List of Symbols to understand the notation used in the analysis.

In the analysis, we consider an n-counter EARDet is running over a network link and its link capacity is ρ. We use β_{TH} to denote the threshold of the counter, and use α as the maximum packet size. Thus, $\beta_{TH} + \alpha$ is the maximum possible value of each counter. Table 3.1 summarizes the notations used in this section.

Table 3.1: 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

3.4.1 Property 1: No False Negative on Large Flows

To analyze the false negative issue of this filter, we consider the performance of our filter under the worst case (namely, the best case for the attacker). To have the worst case for us, the attackers expect their counter's value can decrease as much as possible to make the attacker's flow have the smallest possibility to be caught by the filter.

To consider the decrement of the counters, firstly, we describe all the ways to decrease and increase the value counters:

1. When the incoming flows are virtual flows and there are l empty counters in the filter, then, in time interval 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 the filter, then, in time interval t the decrement is ρt on all counters and the increment is 0. (This is the same as the first situation when $l = 0$.) The new real flows means there is no associated counter in the filter for this flow.

3. When the incoming flows are old real flows, or new real flows and there are some empty counters, then, in time interval t, the decrement is 0 and the increment is $\rho\, t$ on one counter. The old real flow means there is an associated counter in the filter for this flow.

Thus, in the first and second situations, when there are l empty counters in the filter, the decrement is always $\frac{\rho}{l+1}t$ in the interval of t; and in the third situation, the increment is always $\rho\, t$ in the interval of t. The increment and decrement cannot happen at the same time.

We proved Lemma 2 as follow, and the proof sketch of it is in Appendix A.1.3.

Lemma 2 *In any time interval* $[t_1, t_2]$*, the upper bound of decrement of all the counters is* $\frac{\rho}{n+1} \cdot (t_2 - t_1) + \alpha + \beta_{TH}$.[5]

With Lemma 2, we proved that EARDET can detect any large flows which violate the high-bandwidth threshold. We theoretically proved this property and conclude it in the following theorem.

[5]Lemma 2 is taken from Wu et al.'s paper [1].

Theorem 3 ***No-FN$_\ell$ property.*** EARDET *detects every flow violating the high-bandwidth threshold $TH_h(t) = \gamma_h t + \beta_h$ over a time window of length t, when $\gamma_h \geq R_{NFN} = \frac{\rho}{n+1}$ and $\beta_h \geq \alpha + 2\beta_{TH}$.*[6]

Proof sketch: According to Lemma 2, in time interval $[t_1, t_2]$, the decrement of a counter will not exceed $\frac{\rho}{n+1} \cdot (t_2 - t_1) + \alpha + \beta_{TH}$. Because any flow cannot be associated with two or more counters at the same time, therefore, in any $[t_1, t_2]$, for any flow f passing the filter the decrement $dec_f < \frac{\rho}{n+1} \cdot (t_2 - t_1) + \alpha + \beta_{TH}$. Thus, if there is a flow f with rate of $R(t)$, and it violates the high-bandwidth threshold, then:

$$\int_{t_1}^{t_2} R(t)\,\mathrm{d}t \geq TH_h(t_2 - t_1) \geq \frac{\rho}{n+1}(t_2 - t_1) + \alpha + 2\beta_{TH} \tag{3.1}$$

Then, the remaining value of f's counter is:

$$Remains = \int_{t_1}^{t_2} R(t)\,\mathrm{d}t - dec_f > \beta_{TH} \tag{3.2}$$

Because β_{TH} is the threshold of the filter, the flow f will be caught before time t_2. Therefore, for any flow which violates the high-bandwidth threshold, it will be caught by the filter. Namely, there is no false negative in the filter on detecting the flows violating the high-bandwidth threshold. Thus, this theorem is proved now. ■

Another way to prove Theorem 3 is presented in Wu et al.'s paper [1].

From Theorem 3, EARDET can be applied to enforce that all flows violating the high-bandwidth threshold, $TH_h(t) = \gamma_h t + \beta_h$, where $\gamma_h = \frac{\rho}{n+1}$ and $\beta_h = \alpha + 2\beta_{TH}$, will be caught by the filter and cut off. In this way, we can largely protect a network link from the large-flow attack and the burst attack, especially when the number of attackers (or attack flows) is fewer than n. That means, if the attackers want to attack this link successfully, they should have more than n machines to send floods. Therefore, this filter effectively limits the DoS attacks.

[6] Theorem 3 is taken from Wu et al.'s paper [1].

3.4.2 Property 2: No False Positive on Small Flows

EARDET will not wrongly catch any small flow complying the low-bandwidth threshold. To demonstrate this point, we first proposed Lemma 4 [1] as follows. The proof of this lemma is in Appendix A.1.4.

Lemma 4 *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 always be 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 $t_{\beta_\ell} = \frac{(n-1)\alpha+(n+1)\beta_\ell}{[1-(n+1)\gamma_\ell/\rho]\rho}$.*[7]

Then, we proposed Theorem 5 [1] which illustrates the property of no false positives on small flows.

Theorem 5 ***No-FP$_s$ property.*** EARDET *will not catch any flow complying with the low-bandwidth threshold $TH_\ell(t) = \gamma_\ell t + \beta_\ell$ for all time windows of length t, when $0 < \beta_\ell < \beta_{TH}$, $\gamma_\ell < R_{NFP}$, $R_{NFP} = \frac{\beta_\Delta}{(n-1)\alpha+(n+1)\beta_\ell+(n+1)\beta_\Delta} \cdot \rho$.*[8]

Proof sketch: According to Lemma 4, 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, $\frac{(n-1)\alpha+(n+1)\beta_\ell}{[1-(n+1)\gamma_\ell/\rho]\rho} < \frac{\beta_{TH}-\beta_\ell}{\gamma_\ell}$,

$$\Leftrightarrow \gamma_\ell < \frac{\beta_\Delta}{(n-1)\alpha + (n+1)\beta_\ell + (n+1)\beta_\Delta} \cdot \rho \tag{3.3}$$

The theorem is proved. ■

3.4.3 Property 3: Large-Flow Incubation Period

Considering a large flow f violates a high-bandwidth threshold over time window $[t_1, t_2)$, we assume the detection is triggered by the packet at t_a. Then, we define the incubation period as $t_a - t_1$, where $t_a \leq t_2$ is due to the no-FN$_\ell$ property of EARDET. According to theoretical analysis, we proved there is an upper bound of the incubation period for the large flow. The upper bound depends on the rate of the large flow over $[t_1, t_2)$.

[7]Lemma 4 is taken from Wu et al.'s paper [1].

[8]Theorem 5 is taken from Wu et al.'s paper [1].

Theorem 6 *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 the 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 $t_{incb} < \frac{\alpha+2\beta_{TH}}{R_{atk}-\frac{\rho}{n+1}}$.*[9]

Proof sketch:[10] Because $R(t_1, t_a) > R_{atk}$, intuitively the t_{incb} of flow with an average rate of $R(t_1, t_a)$ must be shorter than the t'_{incb} of flow with a 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

$$\begin{aligned} R_{atk}(t'_2 - t'_1) &= \frac{\rho}{n+1}(t'_2 - t'_1) + \alpha + 2\beta_{TH} \\ \Rightarrow t_{incb} < t'_{incb} = t'_a - t'_1 \leq \ t'_2 - t'_1 &= \frac{\alpha + 2\beta_{TH}}{R_{atk} - \frac{\rho}{n+1}} \end{aligned} \tag{3.4}$$

Thus, the theorem is proved. ■

3.4.4 Property 4: Deterministic Performance

The proofs of the three properties above do not make any assumptions on the input traffic, which means EARDet will keep these properties regardless of the type of the input traffic or attack pattern. The attackers are not able to escape the detection through manipulating the flows and playing with timing. Thus, we say EARDet provides deterministic performance over large-flow detection.

3.5 Evaluation

In this section, we present the theoretical analysis and real-traffic simulation results of EARDet and another two related large-flow detection algorithms, Fixed-window-based Multistage Filters (FMF) [2] and Arbitrary-window-based Multistage Filters (AMF) [17], to evaluate performance of EARDet. In terms of the exactness outside the ambiguity region, the evaluation shows

[9]Theorem 6 is taken from Wu et al.'s paper [1].
[10]The proof sketch is taken from Wu et al.'s paper [1].

that EARDet outperforms the prior work in both large rate flow detection and burst flow detection.

3.5.1 Theoretical Evaluation

As introduced in Section 2.2.2, multistage filter maintains an array of counters to record the size of flows. For an incoming flow, the filter will hash map the flow identifier to a counter in the array, and whenever a packet of this flow arrives in the filter, we increase the counter by the size of the packet. Once the value of the counter exceeds the threshold of a large flow, multistage filter catches all flows associated to this counter as a large flow. The difference between AMF and FMF is that AMF uses leaky buckets instead of regular counters.

We can easily observe that FMF and AMF have no false negative over large flows, because if a flow is a large flow, its counter must exceed the large flow threshold. However, there are some false positives resulting from these two algorithms. For example, if a large flow and a small flow are mapped to the same counter, the small flow will be detected as a large flow too. To lower the false positive rate, FMF and AMF must increase the number of counters. But this introduces more overhead in storage space. To understand the performance of three large-flow detector algorithms, we present a concrete example here. Considering the case with $\gamma_h = 1\%\rho$, $\gamma_l = 0.1\%\rho$, where ρ is the link capacity. The performance of three detectors are described in Table 3.2.

Table 3.2: Numerical Example for FMF, AMF, and EARDet.

Detector	Number of Counters (n)	Rate of FP$_s$	Rate of FN$_\ell$
EARDet	101	0	0
FMF	101	no guarantee	0*
FMF	1000	$\leq$ 0.04*	0*
AMF	101	no guarantee	0
AMF	2000	$\leq$ 0.04	0

*The result for FMF is not applicable for large burst flows. Because FMF is based on the landmark window model, it provides no guarantee for detecting large burst flows.

Table 3.2 shows that with the same amount of memory space that EARDet uses (i.e. 101 counters), FMF and AMF cannot guarantee there will be no false positives for small flows at all; on the contrary, EARDet can guarantee both no false positives for small flows and no false negatives for large flows. Even using 10x (20x) memory, FMF (AMF) can only guarantee a 0.04 false positive rate for small flows. Moreover, as we mentioned, FMF has no guarantee for large burst flows, however, EARDet and AMF are able to guarantee this. To make the result clearer, we summarize the comparison result in Table 3.3. We say FMF and AMF are not deterministic, because they are dependent on input traffic that can be manipulated by an attacker to result in false positives.

Table 3.3: Comparison Summary for FMF, AMF, and EARDet.

Detector	Storage Cost	No-FP$_s$	No-FN$_\ell$	Deterministic
EARDet	low	guarantee	no guarantee	yes
FMF	high	no guarantee	no guarantee	no
AMF	high	no guarantee	guarantee	no

3.5.2 Experimental Evaluation Environment

Traffic datasets. To make the experiment more convincing, we use real network traffic datasets Federico II [44–46] and CAIDA [47], and we use the first 30 seconds of traffic to run FMF, AMF, and EARDet. Under the flow ID defined by the pair of source IP and destination IP, we summarize each dataset as follows:

- Federico II dataset contains 2911 flows which are collected from a 200 Mbps link. The average link rate is 1.85 MB/s and the average flow size is around 19.9 KB.
- CAIDA dataset contains around 2.5 million flows from a 10 Gbps link. The average link rate is about 280 MB/s and the average flow size is about 3.3 KB.

Attack flows. To comprehensively evaluate performance of EARDet compared to FMF and AMF, we artificially generated two kinds of attack flows: *flooding attack* flows and *shrew DoS attack* flows [41,43], and mix the generated attack flows with the real traffic as the experiment input traffic.

Then we test (1) how many attack flows escape the detection, and (2) how many legitimate flows are falsely caught as large flows.

Flooding attack flows are the flows with a high rate, thus we generate such flows second by second. For each second interval, we randomly distribute $\gamma_{large}/packetSize$ packets in this one second, where γ_{large} is the flooding flow rate. Then we do the same work for all 30 seconds.

Shrew DoS attack flows consist of some periodic bursts, and attackers use such bursty traffic to block TCP traffic by exploiting the TCP congestion control mechanism. To generate shrew DoS attack flows, we randomly pick up an initial timestamp (from 0 to 29 seconds) for each flow, and then generate a burst with size $\gamma_{burst} \cdot l_{burst}$ every T seconds, where γ_{burst} is the rate of the burst traffic, the l_{burst} is the duration of each burst, and T is the period of the burst.

Configure EARDet. We configure EARDet's parameters as shown in Table 3.4. With this configuration, EARDet is able to catch all large flows which violates the high-bandwidth threshold $TH_h(t) = 0.01\rho t + 15.5$ KB, while not hurting any legitimate flows which comply with the low-bandwidth threshold $TH_\ell(t) = 0.001\rho t + 6072$ B for flows in dataset Federico II. For the dataset CAIDA, there is only a slight difference in β_h, n, and t_{upincb}. The congested link status means the link is fully congested by flows; the non-congested link status means the link still contains many idle time intervals. For a detailed description about how to come up such parameters, please refer the technical report by Wu et al. [48].

Table 3.4: Parameters of EARDet.

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

Table 3.5: Parameters of FMF.

Parameters	Federico II	CAIDA
b	55/250	55/250
d	2	2
n	110/500	110/500
T	250 KB	12.5 MB

Table 3.6: Parameters of AMF.

Parameters	Federico II	CAIDA
b	55/250	55/250
d	2	2
n	110/500	110/500
u	15.5 KB	15.4 KB
r	250 KB/s	12.5 MB/s

Configure FMF and AMF. We set the number of stages for FMF and AMF as $d = 2$, and the number of counters in each stage as $b = 250$. For FMF, we set the window size as 1 second, namely, it checks whether the counter exceeds the threshold every second. Therefore, the threshold of FMF is $T = \gamma_h$. For AMF, we set the leaky bucket threshold as $u = \beta_h$ and the leaky bucket rate as $r = \gamma_h$. We are also interested in investigating how these two large-flow detectors perform with the same amount of storage cost used by EARDet, thus, we also consider the case that $b = 55$ and $d = 2$. The configuration is summarized in Table 3.5 and Table 3.6.

3.5.3 Experimental Evaluation Results

We found that the experiment result of the experiments using CAIDA dataset shows a similar result to the one of the experiments using Federico II, thus, we just present the result of the experiments running with Federico II dataset.

To measure the performance of FMF, AMF, and EARDet, we mainly focus on three metrics: false positive probability for small flows, and detection probability and incubation period for large flows. The false positive probability measures the probability for the detector to wrongly detect a small flow as a large flow. The detection probability is the probability that a detector can successfully catch large flows. The large-flow incubation period shows the time needed to catch a large flow since the flow appears in the link.

To illustrate the experiment result, Figure 3.3, Figure 3.4, Figure 3.5(a) to 3.5(h), and Figure 3.6 are taken from a paper written by Wu et al. [1] ©2014 Association for Computing Machinery, Inc. by permission.

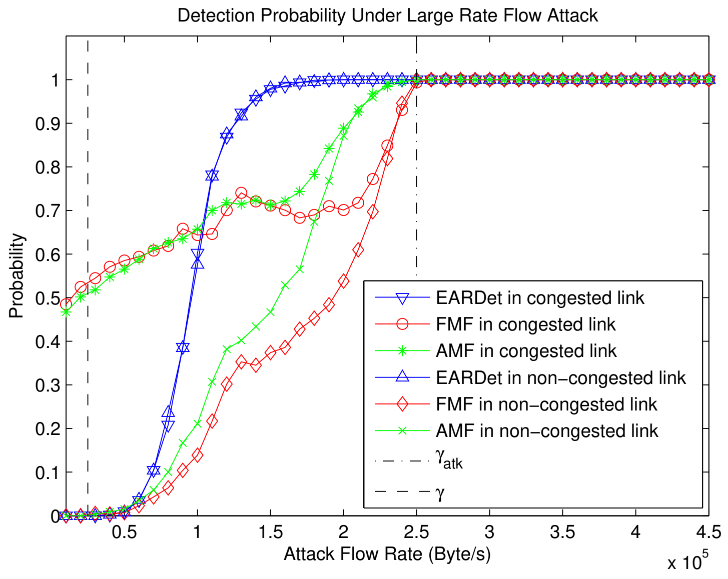


Figure 3.3: Detection probability under flooding DoS. FMF and AMF use 55*2 counters [1].

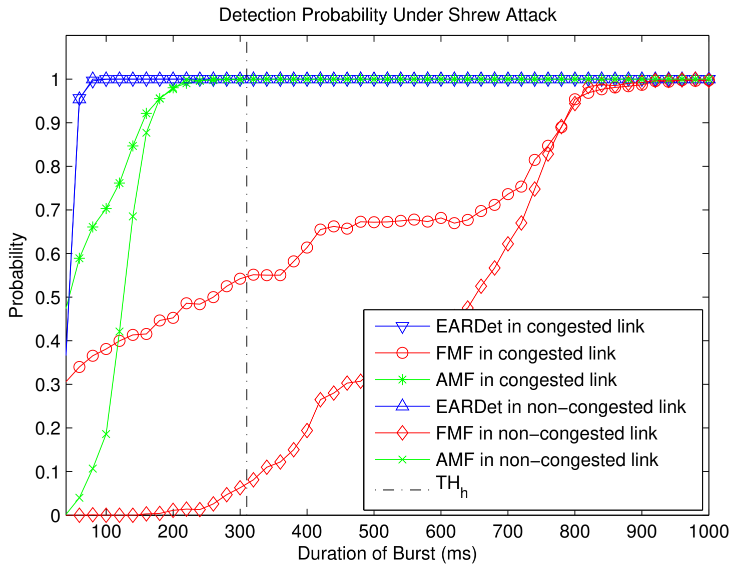
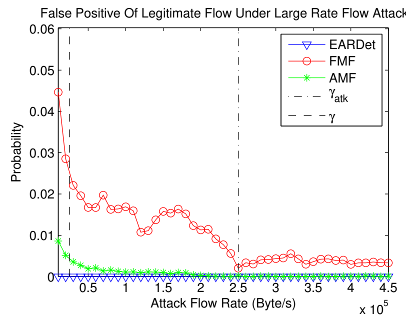
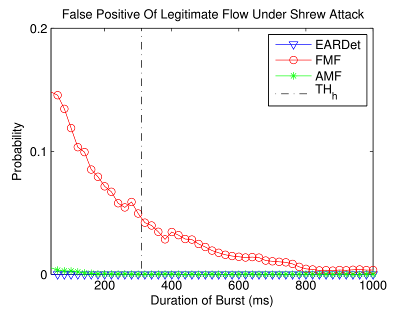


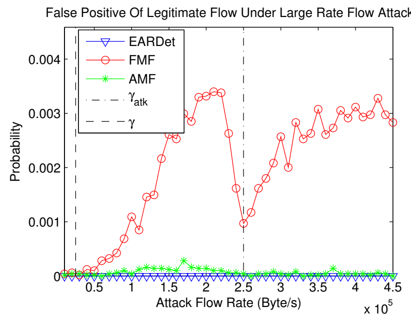
Figure 3.4: Detection probability under shrew DoS. FMF and AMF use 55*2 counters [1].



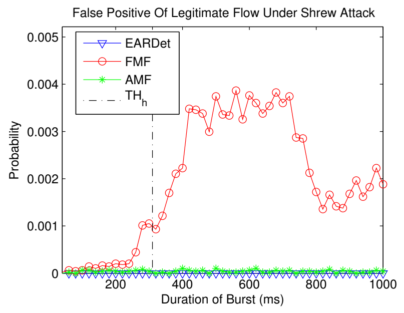
(a) 55*2 counters - Congested Link



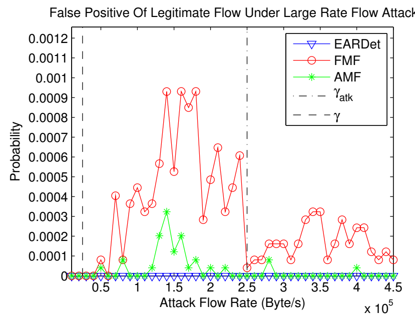
(b) 55*2 counters - Congested Link



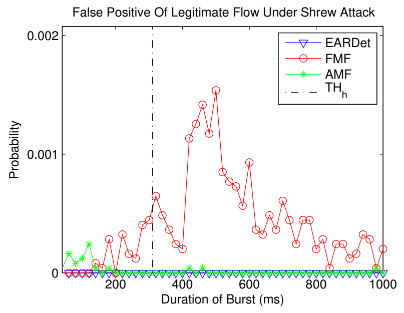
(c) 55*2 counters - Non-congested Link



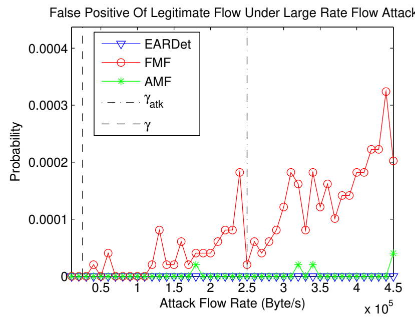
(d) 55*2 counters - Non-congested Link



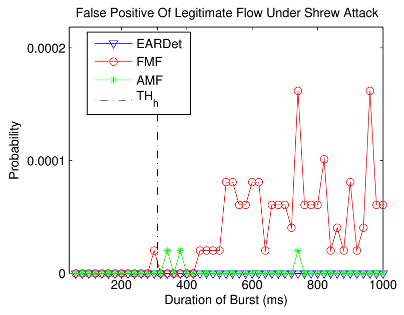
(e) 250*2 counters - Congested Link



(f) 250*2 counters - Congested Link



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



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

Figure 3.5: False positive for small flows [1].

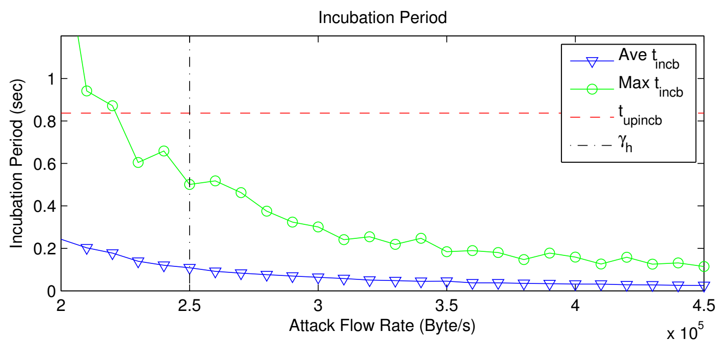


Figure 3.6: Incubation period for large flows [1].

Figure 3.3 shows the detection probability of three detectors in front of flooding DoS attack. We can see all of three flows can perfectly catch all large flows which violate the large-flow threshold. However, FMF and AMF cannot guarantee that there are no false positives all the time. Especially, when the link is congested, FMF and AMF falsely caught a lot of flows below the low-bandwidth threshold.

Figure 3.4 represents the detection probability of three detectors when shrew DoS attack happens. As we expected, EARDET and AMF can catch all bursty attack flows, however, FMF missed a lot of such attack flows because it is only based on the fixed window model.

For false positive probability over small flows, we take a look at Figures 3.5(a) to 3.5(h). The result shows no false positives in the result from EARDET. However FMF and AMF cannot avoid the false positives. When FMF and AMF are using the same number of counters as used by EARDET, we can find many false positives, especially in the congested link. Figure 3.5(a) and Figure 3.5(b) indicate that in the congested link, FMF suffers more 1% and 4% false positives under shrew DoS attack and flooding DoS attack respectively. Increasing the number of counters can reduce the false positives for FMF and AMF, but it is impossible to guarantee no false positive.

The results also reflect that EARDET is deterministic regardless of what input traffic is used. It is even more interesting that in the ambiguity region, the curves of detection probability of EARDET are exactly the same. Maybe we could discover more in the ambiguity region in the future.

Figure 3.6 perfectly supports Theorem 6. The figure shows that the maximum incubation period of attack flows is always below the theoretical upper bound no matter what the attack flow rate is. Moreover, we observed that usually the maximum incubation period is much smaller than the theoretical upper bound and the average incubation period is even much smaller.

3.6 Chapter Summary

In this chapter, we proposed EARDet, a novel arbitrary-window-based algorithm, which is exact outside an ambiguity window. Inspired by the MG algorithm, EARDet not only keeps the property of no false detection over large flows exceeding a high-bandwidth threshold, but also achieves the no false accusation on small flows complying a low-bandwidth threshold with no assumption on the input traffic or attack pattern. We demonstrate this both in theoretical analysis and experimental evaluation.

CHAPTER 4

CLEF: LIMITING THE DAMAGE CAUSED BY LARGE FLOWS IN THE INTERNET CORE

4.1 Chapter Overview

In this chapter, we aim to design algorithms to detect large flows in the Internet core routers with high capacity, e.g. aggregate capacity of several Terabits per second (Tbps). Thus, we require the algorithm to be highly efficient and scalable.

Several approaches for large-flow detection have been proposed; they can be categorized into deterministic (i.e., not relying on random sampling or random binning) and probabilistic algorithms. Examples of deterministic algorithms are EARDet [1] and Space Saving [32], while Sampled Netflow [31] and Multistage Filters [2,17] are examples of probabilistic approaches. With the increase of core router bandwidth, reducing overhead and increasing efficiency of large-flow detection algorithms are critical. For example, for EARDet to detect a 1 Mbps flow on a 100 Gbps link, it would require 10^5 counters on that link, which calls for specialized hardware to update the counters on a per-packet basis. Furthermore, mechanisms that require per-flow state, such as Netflow, may encounter as many as 10^7 flows across the same 100 Gbps link. Maintaining 10^5 counters, including the IPv4 or IPv6 metadata,[1] requires about 1.6–4 MB of state for each 100 Gbps of bandwidth. Since the state has to be accessed for each data packet, the memory system has to be highly optimized. In contrast, the scheme designed in this chapter requires only 10 KB per 100 Gbps bandwidth, greatly reducing hardware cost and complexity.

Because core routers process packets in hardware, and because the designers of core routers add hardware as necessary to accomplish high-speed

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

processing, it is difficult to provide a conclusive upper bound on the computation and memory that a core router can deploy across its links. Thus, to analyze a budget for in-router memory, we consider three levels of memory requirement: the lowest level is the size of a commodity CPU L1 cache, or about 32 KB; the intermediate level is the size of a commodity CPU L2 cache, or about 256 KB; the largest level is the size of a commodity CPU L3 cache, or about 8 MB.

With such a limited amount of memory, an argument based on the pigeonhole principle shows that flows cannot be consistently detected as soon as they exceed certain large-flow thresholds. When the immediate detection of large flows is not possible due to memory constraints, the goal should be to minimize the damage caused to legitimate flows; in such an environment, it is likely more important to rapidly catch very high-rate flows than it is to quickly catch mildly misbehaving flows. Our damage model considers two causes of packet loss: loss caused by large flows, and false positives that punish legitimate flows.

Of particular importance in the damage metric is that an attacker can perform bursty attacks with periodic traffic bursts. Our damage metric can also reflect the damage caused by attacks such as the Shrew attack [49] because we count every dropped legitimate packet.

In this chapter, we propose a hybrid scheme called CLEF, short for *in-Core Limiting of Egregious Flows*, which combines the deterministic EARDet mechanism for rapid detection of very large flows with a novel, randomized algorithm, called *Recursive Large-Flow Detection* (RLFD), that eventually detects large flows not caught by EARDet. Unlike existing detectors, RLFD efficiently distinguishes large flows from legitimate flows by evaluating one set of flows at a time, and recursively shrinking the set of suspected large flows. Larger flows are detected with higher probability in RLFD, so the expected detection time decreases in the level of overuse, resulting in limited damage across a wide range of flow overuse amounts.

In this chapter, our main contributions are: the definition of a damage metric to analyze the performance of large-flow detection; 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; and the analysis of worst-case attacks against proposed large-flow detectors.

4.2 Problem Definition

This chapter 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 proposes 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.

4.2.1 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 [50]), our goal is to perform effective detection with much fewer counters.

- **Low damage:** Because the overhead of exact detection (i.e., no false positives and no false negatives) is excessive in our environment, our schemes trade timely detection for space efficiency, aiming to limit the impact of overusing flows. Section 4.2.2 introduces our damage metric, which quantifies the impact of overuse on legitimate flows.

4.2.2 Damage Metric

Rather than relying on memory-intensive exact enforcement algorithms, this chapter considers the *damage* inflicted by a large flow before it is detected. Specifically, we propose a new metric, which we call *damage*, to assess the effectiveness of a large-flow detection algorithm. Intuitively, damage quantifies the combined impact of large flows and detection-algorithm-induced false positives on legitimate traffic (that is, traffic from flows other than large flows).

In this chapter, we evaluate the damage metric over a link that is at capacity, such that any large-flow traffic causes packet loss. In general, damage could be modified to contemplate links that are not at capacity, or traffic of different priority classes. We thus characterize large-flow impact to be equal to large-flow overuse. The damage metric, then, is large-flow overuse (intuitively, the losses resulting from the large flow) plus false positives (intuitively, the losses resulting from an incorrect detection by an erroneous detection algorithm). Symbolically, our damage metric $D = D_{over} + D_{fp}$ where

- Overuse Damage (D_{over}): the total amount by which all large flows exceed the flow specification.
- False Positive Damage (D_{fp}): the amount of legitimate traffic incorrectly blocked by the detection algorithm.

Relationship with other metrics. Previous work often uses false positive, false negative, and detection delay metrics; the damage metric reflects these three, but is more expressive than each metric individually.

- False Positive (FP): a flow incorrectly detected as a large flow.
- False Negative (FN): a large flow missed by the detection algorithm.
- Detection Delay: the latency between a large flow's first threshold-exceeding packet and the detection of that flow.

Overuse damage reflects both FNs and detection delay; it also distinguishes between flows that significantly exceed the flow specification and those that barely exceed the flow specification. For example, if large flow A is much larger than large flow B, a missed detection on A or B results in the same FN rate; however, the damage metric reflects the increased cost of missed detection on the larger flow A. The damage metric elegantly unifies common metrics like FP, FN, and detection delay, simplifying the comparison of detection algorithms.

4.2.3 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 [2], 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 [2, 18–21, 23, 24, 32], and random numbers used for packet/flow sampling [2]. This assumption prevents the attacker from 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 4.1 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.

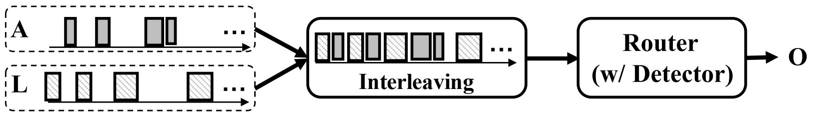


Figure 4.1: Adversary model.

4.3 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.

4.3.1 Existing Detection Algorithms

We review the three most relevant large-flow detection algorithms, summarized in Table 4.1. 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 [1] guarantees exact and instant detection of all flows exceeding a high-rate threshold $\gamma_h = \frac{\rho}{m}$, where ρ is the link capacity and m is the number of counters. However, EARDet may fail to identify a flow whose rate stays below γ_h.

Multistage Filters (AMF). Multistage filters [2, 17] 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. [1], uses leaky buckets as counters. AMF guarantees no-FN and immediate detection for any flow specification; however, AMF has FPs, which increase as the link becomes congested (as shown in Appendix B.1.2).

Flow Memory. Flow Memory (FM) [2] 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 [2]'s random eviction. If the flow memory is large enough to avoid eviction, it provides exact detection. In practice, however, FM is unable to handle a large number of flows in the network, resulting in frequent flow eviction and potentially high FN. The analysis in Appendix B.1.1 shows that 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 4.1: Comparison of Three Existing Detection Algorithms. None of Them Achieve All Desired Properties.

Algorithm			EARDet	AMF	FM
No-FP			yes	no*	yes
No-FN	Flat large flow	low-rate	no**	yes	no*
		high-rate	yes	yes	yes***
	Bursty large flow	low-rate	no**	yes	no*
		high-rate	yes	yes	yes***
Instant detection			yes	yes	yes
Deterministic			yes	no	no

*Appendix B.1.1 and B.1.2 show that FM 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.

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

4.3.2 Advantages of Hybrid Schemes

As Table 4.1 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 chapter.) 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 chapter considers the parallel approach that identifies a flow whenever it is detected by either detector.

The existing EARDet and FM 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 FM 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 chapter, 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.

4.4 RLFD Algorithm and CLEF Hybrid Scheme

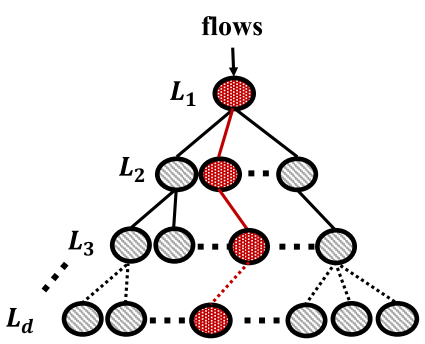
In this section, we present our new large-flow detectors. First, we describe the *Recursive Large-Flow Detection* (RLFD), a novel approach, which is designed to use very little memory but provide eventual detection for large flows, and present the data structures, runtime analysis, and advantages and disadvantages of RLFD. Next, we develop a hybrid detector, CLEF, that addresses the disadvantages of RLFD by combining it with the previously proposed EARDet [1]. 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.

4.4.1 RLFD Algorithm

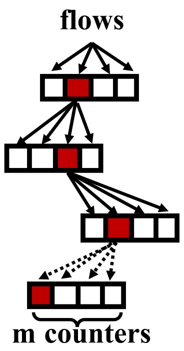
RLFD is a randomized algorithm designed to perform memory-efficient detection of low-rate large flows; it is designed to scale to a large number of flows, as encountered by an Internet core router. RLFD is designed to limit the damage inflicted by low-rate large flows while using very limited memory. 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 (monitoring 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.

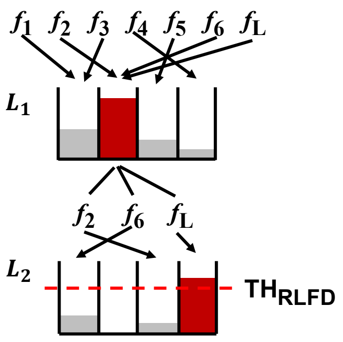
Recursive subdivision. To operate on extremely limited memory, RLFD



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



(b) A Tree Branch.



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

Figure 4.2: RLFD structure and example.

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*[2] (Figure 4.2(a)) of depth k. 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 try 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 4.2(b). This mapping is determined by hashing the flow ID with a keyed hash function, where the key is randomly generated by each router. Section 4.4.2 explains how RLFD efficiently implements this random mapping.

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}. Section 4.4.2 describes how to configure RLFD to guarantee no-FP and low detection delay. We choose $d \geq \lceil \log_m n \rceil$ so that a leaf node is likely to contain a single flow.

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

[2]The terms "counter tree" and "virtual counter" are also used by Chen et al. [35], 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 bad behavior.

level L_k. Though these periods need not be of equal length, in this chapter we consider periods of equal length T_ℓ, which results in a RLFD detection cycle $T_c = d \cdot T_\ell$.

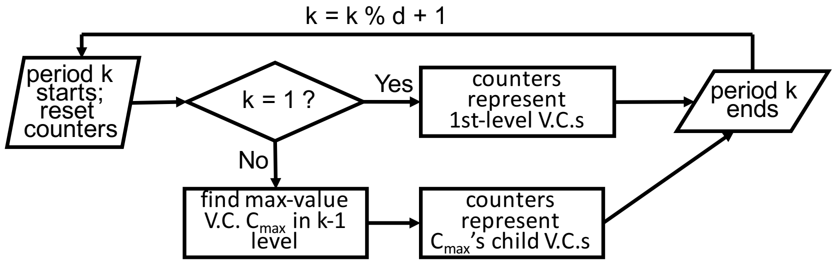
RLFD always chooses the root node to monitor at level L_1; after monitoring at level L_k, RLFD identifies the largest counter C_{max} among the m counters at level L_k, and uses the node corresponding to that counter for level L_{k+1}. Section 4.5.3 shows that choosing the largest counter detects large flows with high probability.

Figure 4.2(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}.

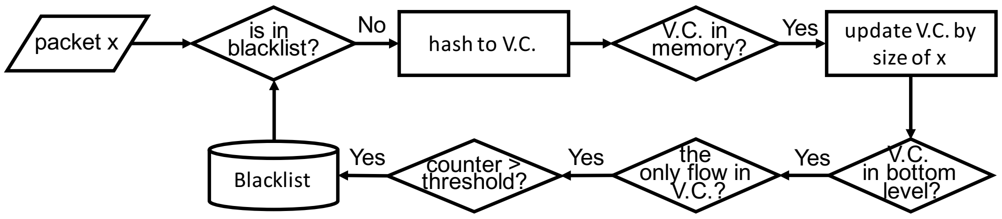
Algorithm description. As shown in Figure 4.3(a), the algorithm starts at the top level L_1 so each counter represents a child of the root node. At the beginning of each period, all counters are reset to zero. At the end of each period, the algorithm finds the counter holding the maximum value and moves 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 4.3(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 [51] at the bottom level to reduce collisions (as described in Section 4.4.2).

- **Hashing to Counters.** RLFD hashes each packet's flow ID to one virtual counter at each level of the virtual counter tree. Each incoming packet must be hashed so that RLFD can determine whether the flow is monitored in the current time period. Though a naïve approach


k = k % d + 1
period k starts; reset counters
k = 1 ?
Yes
No
counters represent 1st-level V.C.s
period k ends
find max-value V.C. C_{max} in k-1 level
counters represent C_{max}'s child V.C.s

(a) Level Change Diagram.



(b) Packet Processing Diagram.

Figure 4.3: RLFD Decision Diagrams. "V.C." stands for virtual counter.

would be to perform one hash for each level (total run time of $O(d)$), we propose an $O(1)$ implementation in Section 4.4.2.

- **Counter Threshold.** 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_ℓ 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_ℓ must violate the large-flow threshold $\mathsf{TH}(t)$, so RLFD has no FPs.

- **Blacklist.** As in EARDET [1], RLFD includes a blacklist to record detected large flows for future analysis and action. We calculate the damage D with the assumption that CLEF blocks large flows right after they are put into the blacklist.

4.4.2 RLFD Data Structure and Optimization

In this section, we describe how to efficiently implement RLFD and propose additional optimizations to the basic RLFD described in Section 4.4.1.

Hashing and counter checking. Hashing each flow f into a path of virtual counters $PATH(f)$ and checking whether any of these counters are loaded in the memory are two performance-critical operations of RLFD.

For each packet, our implementation only requires three bitwise operations (a hash operation, a bitwise AND operation, and a comparison over 64 bits), thus requiring only $O(1)$ time and $O(1)$ space on a modern 64-bit CPU.

A naive implementation of hashing could introduce unnecessary cost in computation and space. For example, a naive implementation may maintain one hash function per virtual counter array. To check whether an incoming flow needs to be monitored, it needs to check whether the incoming flow is hashed into every maximum-value counter in each level above the current level. However, this took $O(d)$ time for checking level by level and $O(d)$ space for hash functions, where d is the depth of the virtual counter tree. $O(d)$ computational complexity per packet is too high due to the limited number of per-packet memory access in a typical modern CPU.

$$H(f) \triangleq |\underbrace{b_0 b_1 b_2 \dots b_{s-1}}_{\text{For } L_1} \dots \underbrace{b_{s(k-1)} \dots b_{sk-1}}_{\text{For } L_k} \dots b_{sd-2}\, b_{sd-1}|$$

$$M(L_k) \triangleq |\underbrace{1111111 \dots 111111}_{s(k-1) \text{ bits}}\, \underbrace{00000 \dots 000000}_{s(d-k+1) \text{ bits}}|$$

$$A(l_{i,k}) \triangleq |\underbrace{a_0 a_1 a_2 \dots a_{s-1}}_{\text{For } L_1} \dots \underbrace{a_{s(k-2)} \dots a_{s(k-1)-1}}_{\text{For } L_{k-1}}\, 00 \dots 000|$$

$$H(f) \text{ AND } M(L_k) \begin{cases} = A(l_{i,k}) & f\text{'s V.C. is in the loaded counter array } l_{i,k} \\ \neq A(l_{i,k}) & f\text{'s V.C. is } \textbf{not} \text{ in the loaded counter array } l_{i,k} \end{cases}$$

Figure 4.4: RLFD counter hash and in-memory check. $H(f)$ reflects the hash-generated bin number for all levels, $M(L_k)$ reflects a mask that includes the first $k-1$ levels, and $A(l_{i,k})$ reflects the bins selected in each of the first $k-1$ levels. Flow f is in the i,k counter array exactly when $H(f)$ & $M(L_k) = A(l_{i,k})$.

Inspired by how a network router finds the subnet of an IP address, as Figure 4.4 illustrates, we map a flow to a virtual counter per level based on a single hash value. Specifically, given an incoming flow f, we compute $H(f)$, and then do a bitwise AND operation of $H(f)$ and a mask value $M(L_k)$ of the current level L_k. We then check whether the result is equal to the hash value $A(\ell_{i,k})$ of the current loaded counter array $\ell_{i,k}$ (the ith counter array in the kth level). If the $H(f)$ AND $M(L_k) = A(\ell_{i,k})$, then the virtual counter of f in the level L_k is in the currently loaded counter array $\ell_{i,k}$.

Assuming RLFD has d levels and m counters in each counter array, we hash a flow ID $f = \mathsf{fid}(x)$ into $H(f)$ with $s \cdot d$ bits, where $s = \log_2 m$. We require the system designer to only choose the base-2 exponential value for m, so that the s is an integer.

The bits $[b_{s(k-1)} : b_{sk-1}]$ [3] of $H(f)$ is the index of the virtual counter in its counter array in the kth level L_k. As each counter array is determined by its ancestor counters as Figure 4.2(b) describes, the bits $[b_0 : b_{s(k-1)-1}]$ can uniquely determine the counter array in the level L_k for the flow f. Thus, to check whether the virtual counters of a flow is in memory, we just need to track the ancestor counters of the currently loaded counter array $l_{i,k}$. We track the ancestor counters by $A(\ell_{i,k})$, which is also a value of $s \cdot d$ bits. The bits $[a_0 : a_{s(k-1)-1}]$ record the index of ancestor counters of $\ell_{i,k}$, and the rest of bits are all 0s. To track $A(\ell_{i,k})$, we just simply set the bits $[a_{s(k-1)} : a_{sk-1}]$ as the index of the selected counter at the end of the period of L_k. The mask value for the level L_k is also a value of $s \cdot d$ bits, whose first $s(k-1)$ bits are 1s and the rest are 0s. By $H(\mathsf{fid}(x))$ AND $M(L_k)$, we extract the ancestor bits $[b_0 : b_{s(k-1)-1}]$ of the flow $\mathsf{fid}(x)$, and compare it with the ancestor bits $[a_0 : a_{s(k-1)-1}]$ of the loaded counter array. If they match, then the flow $\mathsf{fid}(x)$'s counter is in the memory, and we update the counter with index $[b_{s(k-1)} : b_{sk-1}]$ by the size of the packets of the flow $\mathsf{fid}(x)$.

For each packet, our implementation above only needs three bitwise operations: a hash operation, an AND operation, and a comparison over $d \log_2 m$ bits. Although the number of bits used in this implementation depends on d and m, a 64-bit long integer is enough in most of the cases, thus those bitwise operations only take O(1) CPU cycles in a modern 64-bit CPU. Therefore, we say this implementation only takes $O(1)$ time and $O(1)$ space.

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 counter assignments between each top-to-bottom detection cycle are approximately independently random. For simplicity, in this chapter we analyze RLFD with uniformly random hash functions. 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

[3] $[b_i : b_j]$ denotes a block of bits $\{b_k\}$, $i \le k \le j$.

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 usually small, 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.

No-FP guarantee. To guarantee no FP, we only identify large flows whose counters have no second flow. At the bottom level L_d, we randomly pick m flows from among the n_d flows in L_d, and use Cuckoo hashing [51] to assign each chosen flow to a counter. By using Cuckoo hashing, the expected flow insertion time is constant and the worst-case lookup and update time are constant. Furthermore, there are nearly no hash collisions.

To guarantee no FP, we only identify large flows whose counter has no second flow, i.e. no flow hash collision C_{free}. If we randomly hash flows into counters at the bottom level L_d, the no-collision probability for a counter is $Pr(C_{\mathit{free}}) = [\frac{m-1}{m}]^{n_d-1}$, where n_d is the number of flows selected into L_d. Because we want to have d as small as possible, thus, we usually may choose $d = \lceil \log_m n \rceil$, where n is the total number of flows in the link. Thus, $n_d \leq m$ on average. Thus,

$$Pr(C_{\mathit{free}}) = \left[\frac{m-1}{m}\right]^{m\frac{n_d-1}{m}} \approx e^{-\frac{n_d-1}{m}} \tag{4.1}$$

When $n_d \approx m$, the no-collision probability $Pr(C_{\mathit{free}}) \approx \frac{1}{e} = 0.368$, which gives a collision probability for each flow of 0.632.

To avoid the high collision probability in the regular hash above, we randomly pick m flows (out of n_d flows) instead. Each of m flows is monitored by a dedicated counter (which does not introduce additional FNs, because $n_d \leq m$). To efficiently implement this counter assignment, we can use Cuckoo hashing [51] to achieve constant expected flow insertion time and worst-case constant lookup and update time. Cuckoo hashing resolves collisions by using two hash functions instead of only one in regular hashing. As in [52], Mitzenmacher shows that, with three hash functions, Cuckoo hashing can achieve expected constant insertion and lookup time with load factor of 91%. Thus, when $m = n_d$, the Cuckoo hashing can achieve $Pr(C_{\mathit{free}}) \approx 0.91$, which is still much larger than $Pr(C_{free}) \approx 0.368$ in the regular hashing. As n_d is usually less than m (because we set d to be the ceiling of $\log_m n$), it is reasonable to treat the $Pr(C_{free}) \approx 1$ in our later analysis. Cuckoo hashing requires to store both the key (48 bits for IPv4, 144 bits for IPv6) and value (32 bits) of an entry, thus, for each counter, we need space for the flow ID and the counter value.

4.4.3 Shrinking Counter Entry Size

As we discussed, the number of flows hashed into the bottom level is much less than m (e.g. at most 2^{10}). A key space of 96 bits (288 for IPv6) is too large for less than 2^{10} keys. We can hash the flow IDs into a smaller key space, e.g. 48 bits to save memory size. For each flow, although hash collision could happen and may result in FP in the detection in the bottom level, the probability is less than $1 - [\frac{2^{48}-1}{2^{48}}]^{2^{10}-1} \approx 2^{-38}$ which is very small. For systems can tolerate such extremely low FP probability, we recommend it to do so.

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.

4.4.4 RLFD Runtime Analysis

We analyze the runtime using the same CPU considered in EARDet [1]. An OC-768 (40 Gbps) high-speed link accommodates 40 million mid-size (1000 bit) packets per second. To operate at the line rate, a modern 3.2 GHz CPU must process each packet within 76 CPU cycles. A modern CPU might contain 32 KB L1 cache, 256 KB L2 cache, and 20 MB L3 cache. It takes 4, 12, and 30 CPU cycles to access L1, L2, and L3 CPU cache, respectively; accessing main memory is as slow as 300 cycles.

If, over a 40-Gbps link, we conservatively pick a large-flow threshold rate $\gamma = 100$ kbps (a low-rate threshold in today's networks), a maximum of $400,000$ flows can use the threshold rate. An RLFD with $400,000$ flows and

128 counters per level only needs $d = 3$ levels to get an average of 24.4 flows at the bottom level, causing only a few collisions for the 128 counters at the bottom level which will be handled by the Cuckoo hashing approach. Even if we consider a much larger number of flows, such as 40 million, $d = 4$ levels results in around 19.1 flows at the bottom level. In such a four-level RLFD, a flow's path through the tree will require only $4 \cdot \log_2 128 = 28$ bits, so a 64-bit integer is large enough for the hash value. In practice, the threshold rate may be higher than 100 kbps, and the number of flows is likely to be under 40 million.

Computational complexity. Based on the implementation and optimizations in Section 4.4.2, RLFD performs the following steps on each packet: (1) *a hash computation* to find the flow's path in the tree, (2) *a bitwise AND operation* to find the subpath down to the depth of the current period, (3) *an integer comparison* to determine if the flow is part of an active counter, and (4) *a counter value update* if the flow is hashed into the loaded counter array. Each of these operations is $O(1)$ complexity and fast enough to compute within 76 CPU cycles.

At the bottom level, after operations (1) to (3), RLFD performs the following steps: (5) *a Cuckoo lookup/insert* to find the appropriate counter, (6) *a counter value update* to represent the usage of a flow, (7) *a large-flow check* that compares the counter value with a threshold, and (8) *an on-chip blacklist write* if the counter has exceeded the threshold. Steps (5)–(7) are only performed on packets from the small fraction of flows that are loaded in the bottom-level array; step (8) is only for packets of the flows identified as large flows in step (7), and this only happens once for each flow (if we block the large flows in the blacklist). Thus steps (5)–(8) are executed much less frequently than steps (1)–(4). Even so, steps (5)–(8) have a constant time in expectation, and are likely negligible in comparison with steps (1)–(4).

Storage complexity. RLFD only keeps a small array of counters and a few additional variables: the hash function key, the 64-bit mask value for the current level, and the 64-bit identifier of the currently loaded counter array. Because we use Cuckoo hashing at the bottom level, besides a 32-bit field for the counter value, each counter entry needs to have a field for the associated flow ID key, which is 96 bits in IPv4 and 288 bits in IPv6. An array of 128 counters requires 2 KB in IPv4 and 5 KB for IPv6, which readily fits within the L1 cache. As discussed in Section 4.4.3, we can further shrink the flow ID field size to 48 bits (with FP probability $\leq 2^{-38}$ for each flow); if deployed, a 128 counter array is 1.25 KB and a 1024 counter array is 10 KB for both IPv4 and IPv6, quite a bit less than the size of the L1 cache (32 KB).

4.4.5 RLFD's Advantages and Disadvantages

Advantages. With recursive subdivision and other 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 (Section 4.5.3 analyzes RLFD's detection probability); and (2) guarantee no-FP, eliminating damage from FP.

Disadvantages. First, a landmark-window-based RLFD cannot guarantee exact detection over large-flow specification based on arbitrary time windows [1].[4] However, this approximation results in limited damage, as mentioned in Section 4.3. 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 sends over 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).

4.4.6 CLEF Hybrid Scheme

We propose a hybrid scheme, CLEF (in-Core Limiting of Egregious Flows), 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 do not include flow memory in this hybrid scheme because its detection is not as deterministic as EARDet's.

[4]Landmark window and arbitrary window are introduced in Section 2.1

Parallel composition of EARDet and RLFD. As described in Section 4.3.2, 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 a 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 $T_c^{(2)}$ (denoting the first RLFD and its detection cycle by RLFD$^{(1)}$ and $T_c^{(1)}$, respectively), 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. 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 γ_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 $T_c^{(2)}$.

4.5 Analysis

In this section, we discuss RLFD's performance and its large-flow detection probability. We then compare CLEF with state-of-the-art schemes, considering various types of large flows under CLEF's worst-case background traffic. Due to limited space, some derivations are in Appendix B. The notations used in the rest of sections in this chapter are introduced in Table 4.2.

Detection probability. *Single-level detection probability* is the probabil-

Table 4.2: Table of Notations.

Generic notations:		
ρ	$\triangleq$	Rate of (outbound) link capacity
γ, β	$\triangleq$	Rate and burst threshold flow specification
θ	$\triangleq$	Duty cycle of bursty flows ($\theta \leq 1$)
T_b	$\triangleq$	Period of burst
R_{atk}, α	$\triangleq$	Average large-flow rate, and $R_{atk} = \alpha\gamma$
n	$\triangleq$	Number of legitimate flows
n_γ	$\triangleq$	$\frac{\rho}{\gamma}$; Maximum number of legitimate flows at rate γ
m	$\triangleq$	Number of counters available in a detector
γ_h	$\triangleq$	$\frac{\rho}{m+1}$; EARDet high-rate threshold rate
D_{fp}	$\triangleq$	Damage caused by false accusation
D_{over}	$\triangleq$	Damage caused by overuse by large flows
$E(D_{over})$	$\triangleq$	Expected overuse damage
RLFD notations:		
d	$\triangleq$	Number of levels
$n^{(k)}$	$\triangleq$	Number of legitimate flows in the level k
T_ℓ	$\triangleq$	Time period of a detection level
T_c	$\triangleq$	Detection cycle $T_c = d \cdot T_\ell$
$T_c^{(1)}$	$\triangleq$	Detection cycle T_c of the first RLFD in CLEF
$T_c^{(2)}$	$\triangleq$	Detection cycle T_c of the second RLFD in CLEF
$Pr(A_\alpha)$	$\triangleq$	Detection prob. for flows with $R_{atk} = \alpha\gamma$
$\alpha_{0.5}$	$\triangleq$	When $\alpha \geq \alpha_{0.5}$, approximately $Pr(A_\alpha) \geq 0.5$
$\alpha_{1.0}$	$\triangleq$	When $\alpha \geq \alpha_{1.0}$, approximately $Pr(A_\alpha) = 1.0$
Timing-Randomized RLFD notations:		
$T_{\ell,min}$	$\triangleq$	The minimum T_ℓ
$T_{c,min}$	$\triangleq$	$T_{c,min} = dT_{\ell,min}$
σ	$\triangleq$	$\sigma T_{c,min}$ is the maximum T_c
P	$\triangleq$	Maximum period in brute-force search (for optimal attacks)
W_{in}	$\triangleq$	The input size of reinforcement learning (for optimal attacks)
p	$\triangleq$	The punishment to reinforcement learner when the large flow is caught
$Pr(p)$	$\triangleq$	The probability to apply the punishment p

ity that a RLFD selects a correct counter (containing at least one large flow) for the next level. *Total detection probability* is the probability that one copy of RLFD catches a large flow in a cycle T_c (which can be estimated by the product of single-level detection probabilities across all levels in a cycle).

4.5.1 RLFD Worst-Case Background Traffic

Since our goal is to minimize worst-case damage, here we discuss the worst-case background traffic against RLFD. Given a large flow, we want to find the legitimate flow traffic pattern that maximizes damage caused by the large flow. We then assume this worst-case cross-traffic in the rest of the analysis.

Since damage increases with expected detection delay (and thus decreases with single-level detection probability) in RLFD, we derive the worst-case background traffic by finding the minimum single-level detection probability for each level of RLFD. Theorem 7 states that the worst-case background traffic consists of threshold-rate legitimate flows fully utilizing the outbound link. The proof of Theorem 7 and further discussion are presented in Appendix B.1.3.

Theorem 7 *On a link with a threshold rate γ and an outbound link capacity ρ, given an attack large flow f_{atk}, RLFD runs with the lowest probability to select the counter containing f_{atk} to the next level, when there are $n_\gamma = \rho/\gamma$ legitimate flows, each of which is at the rate of γ.*

Figure 4.5 presents single-level detection probabilities for several different background traffic patterns, which empirically validates our theorem.

4.5.2 Characterizing Large Flows

To systematically compare CLEF with other detectors under various types of attack flows, we categorize large flows based on three characteristics, as Figure 4.6 illustrates:

1. *Burst Period* (T_b). A large flow sends a burst of traffic in a period of T_b.

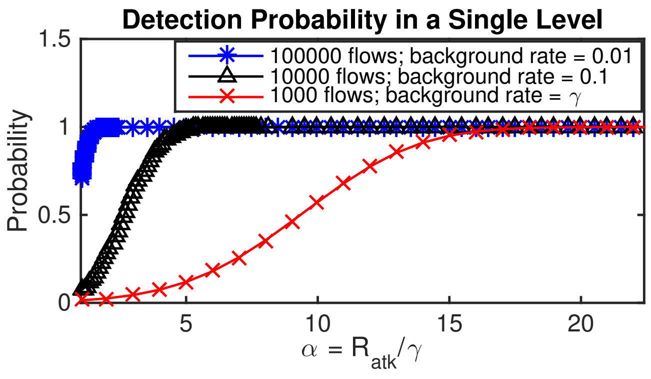


Figure 4.5: RLFD's single-level detection probability of the first level against a large flow at different rate $R_{atk} = \alpha\gamma$, when background legitimate flows at various rates (0.01γ, 0.1γ, and γ) fully use the link capacity of 1000γ. The RLFD suffers the lowest detection probability when the legitimate flows are at the threshold rate γ.

2. *Duty Cycle* ($\theta \in (0, 1]$). In each period of length T_b, a large flow only sends packets during a continuous time period of θT_b and remains silent during the rest of the period.

3. *Average Rate* (R_{atk}). This is the average volume of traffic sent from a large flow per second over a time interval much longer than the burst period T_b. The instant rate during the burst chunk θT_b is R_{atk}/θ.

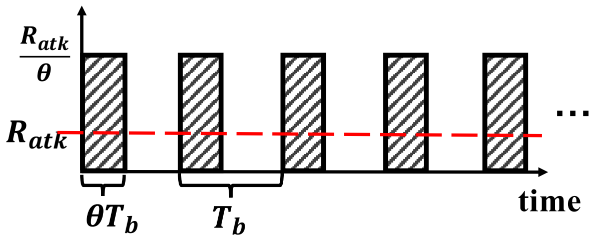


Figure 4.6: Flow with average rate R_{atk}, burst period T_b, duty cycle $\theta \in (0, 1]$.

By remaining silent between bursts, attacks such as the Shrew attack [49] keep the average rate lower than the detection threshold to evade the detection algorithms based on landmark windows [2, 18–21, 23, 24, 32].

A large flow may switch between different characteristic patterns over time, including ones that comply with flow specifications. The total damage in this case can be computed by adding up the damage inflicted by the large flow under each appearing pattern. Hence, for the purpose of the analysis, we focus our discussion on large flows with fixed characteristic patterns.

4.5.3 RLFD Detection Probability for Flat Flows

We discuss the detection probability of RLFD over flat large flows in this section. In order to detect a flat large flow, the traffic of the flat large flow should be observable in each detection level.

The probability that RLFD catches one large flow in a detection cycle increases with the number of large flows passing through RLFD. Because a greater number of large flows implies that more counters may contain large flows in each level, thus RLFD has a higher chance of correctly selecting counters with large flows in the recursive subdivision. Hence, we discuss the worst-case scenario for RLFD where only one large flow is present.

Because the operation in all but the bottom level of RLFD is similar and the only difference is the flows hashed to the counter array, we discuss the detection in a single level first and expand it to the whole detection cycle. Additional numeric examples are provided in Appendix B.1.4.

Single-level detection probability. Given the total number of flows traversing the link is n, we can predict the expected number of flows in the kth level by $n^{(k)} = n/m^{k-1}$, where m is the number of counters. Since $n^{(k)}$ depends only on the total number of flows and not the traffic distribution, we discuss a single-level detection with $n^{(k)}$ legitimate flows, m counters, and a large flow at the rate of $R_{atk} = \alpha\gamma$, where γ is the threshold rate and $\alpha > 1$. When the context is clear, we use n to stand for $n^{(k)}$ in the discussion of single-level detection.

According to Theorem 7, the worst-case background traffic is that all n legitimate flows are at the threshold rate γ; Theorem 8 shows an approximate lower bound of the single-level detection probability $\mathsf{P}_{worst}(m, n, \alpha)$ in such worst-case background traffic. The proof of Theorem 8 and its Corollaries 9 and 10 are presented in Appendix B.2.1.

Theorem 8 *Given m counters in a level, n legitimate flows at full rate γ, and a large flow f_{atk} with an average rate of $R_{atk} = \alpha\gamma$, the probability $\mathsf{P}_{worst}(m, n, \alpha)$ that RLFD will correctly select the counter with large flow f_{atk} has an approximate lower bound of $1 - Q(K, \frac{n}{m})$, where $K = \lfloor \frac{n}{m} +$*

$\sqrt{2\frac{n}{m}\log n} - \alpha\rfloor$; *$Q(K, \frac{n}{m})$ is the cumulative distribution function (CDF) of the Poisson distribution Pois($\frac{n}{m}$).*

Corollary 9 *For a detection level with n legitimate flows, m counters, and a large flow f_{atk} at the average rate of $\alpha_{0.5} \cdot \gamma$, the probability $\mathsf{P}_{worst}(m, n, \alpha_{0.5})$ that RLFD will correctly select the counter of f_{atk} has an approximate lower bound of* 0.5, *where* $\alpha_{0.5} = \sqrt{2\frac{n}{m}\log n}$.

Corollary 10 *For a detection level with n legitimate flows, m counters, and a large flow f_{atk} at the average rate of $\alpha_{1.0} \cdot \gamma$, the probability $\mathsf{P}_{worst}(m, n, \alpha_{1.0})$ that RLFD will correctly select the counter of f_{atk} has an approximate lower bound of* 1.0, *where* $\alpha_{1.0} = 2 \cdot \alpha_{0.5} = 2\sqrt{2\frac{n}{m}\log n}$.

Total detection probability. Theorem 11 describes the total probability of detecting a large flow in one detection cycle. Detailed proof is provided in Appendix B.2.2.

Theorem 11 *When there are n legitimate flows and a flat large flow at the rate of $\alpha\gamma$, the total detection probability of a RLFD with m counters has an approximate lower bound:*

$$Pr(A_\alpha) \geq \begin{cases} \left(1 - Q(K_\gamma, \dfrac{n_\gamma}{m})\right)^{\lfloor \log_m(n/n_\gamma)\rfloor + 1} & \text{, when } n \geq n_\gamma \\ 1 - Q(K, \dfrac{n}{m}) & \text{, when } n < n_\gamma \end{cases} \tag{4.2}$$

where $K_\gamma = \lfloor \frac{n_\gamma}{m} + \sqrt{2\frac{n_\gamma}{m}\log n_\gamma} - \alpha \rfloor$, $K = \lfloor \frac{n}{m} + \sqrt{2\frac{n}{m}\log n} - \alpha \rfloor$, *and* $Q(x, \lambda)$ *is the CDF of the Poisson distribution Pois*(λ).

4.5.4 Twin-RLFD Theoretical Overuse Damage

To evaluate RLFD's performance, we derive a theoretical bound on the damage caused by large flows against RLFD. Recall that there are two sources of damage: FP damage D_{fp} and overuse damage D_{over}. Because RLFD has no FP, there is no need to consider D_{fp}. Thus, we only theoretically analyze D_{over}.

Theorem 12 shows the expected overuse damage for flat flows and bursty flows against a Twin-RLFD. The proof is presented in Appendix B.2.3. Additional numeric examples are in Appendix B.1.5.

Theorem 12 *A Twin-RLFD with RLFD$^{(1)}$ and RLFD$^{(2)}$ whose detection cycles are $T_c^{(1)}$ and $T_c^{(2)} = \frac{2d\gamma_h}{\alpha\gamma} T_c^{(1)}$, respectively, it can detect bursty flows at an average rate $R_{atk} = \alpha\gamma < \theta\gamma_h$, where γ_h is the high-rate threshold rate of the* EARDet. *The expected overuse damage caused by such flows has the following upper bound:*

$$E(D_{over}) \leq \begin{cases} T_c^{(1)}\gamma\alpha/\theta Pr(A_\alpha) & , \textit{when } \theta T_b \geq 2T_c^{(1)} \\ T_c^{(1)} 2d\gamma_h/\theta Pr(A_\alpha) & , \textit{when } \theta T_b < 2T_c^{(1)} \end{cases} \tag{4.3}$$

where

$$Pr(A_\alpha) \geq \begin{cases} \left(1 - Q(K_\gamma, \frac{n_\gamma}{m})\right)^{\lfloor \log_m (n/n_\gamma) \rfloor + 1} & , \textit{when } n \geq n_\gamma \\ 1 - Q(K, \frac{n}{m}) & , \textit{when } n < n_\gamma \end{cases} \tag{4.4}$$

and $K_\gamma = \left\lfloor \frac{n_\gamma}{m} + \sqrt{2\frac{n_\gamma}{m}\log n_\gamma} - \alpha_\theta \right\rfloor$, $K = \left\lfloor \frac{n}{m} + \sqrt{2\frac{n}{m}\log n} - \alpha_\theta \right\rfloor$ *($\alpha_\theta = \alpha/\theta$ when $\theta T_b \geq 2T_c^{(1)}$, and $\alpha_\theta = \alpha$ when $\theta T_b < 2T_c^{(1)}$). The d is the number of levels in RLFD, and $Q(x, \lambda)$ is the CDF of the Poisson distribution Pois(λ). The damage of flat flow is that in the case of $\theta = 1$ and $\theta T_b \geq 2T_c^{(1)}$.*

We can see that a properly configured Twin-RLFD can detect bursty flows unable to be detected by EARDet (i.e., flows at average rate $R_{atk} = \alpha\gamma < \theta\gamma_h$).

4.5.5 Theoretical Comparison

We compare the CLEF hybrid scheme with the most relevant competitor, the AMF-FM hybrid scheme [2], which runs an AMF and a FM sequentially: all traffic is first sent to the AMF and the AMF sends detected large flows (including FPs) to the FM to eliminate FPs. For completeness, we also present the results of individual detectors, including Twin-RLFD, EARDet, AMF, and Flow Memory (FM). Table 4.3 summarizes the damage inflicted by different large-flow patterns when different detectors are deployed. The damage is calculated according to the analyses of AMF (Appendix B.1.2), FM (Appendix B.1.1), EARDet [1], and Twin-RLFD (Section 4.5.4). Figures B.2(c) and B.2(e) in Appendix B.1.5 provide more details about Twin-RLFD's overuse damage presented in Table 4.3.

Comparison setting. To compare detectors in an in-core router setting, we allocate only 100 counters for each detector, and we allocate 50 counters

Table 4.3: Theoretical Comparison. CLEF Outperforms Other Detectors with Lower Large Flow Damage. Damage in Megabyte (MB).

	Algorithm	FP Damage	Overuse Damage (MB)						
			Low-rate Large Flow					High-rate Large Flow	
			$R_{atk} < 10\gamma$	$10\gamma < R_{atk} < 30\gamma$		$30\gamma \leq R_{atk} < 250\gamma$		$250\gamma \leq R_{atk}$	
				$\theta T_b < 2T_c$	$\theta T_b \geq 2T_c$	$\theta T_b < 2T_c$	$\theta T_b \geq 2T_c$	$\theta T_b < 2T_c$	$\theta T_b \geq 2T_c$
Individual	Twin-RLFD	0	$[512, +\infty)$	$[158, 512)$	$[33, 45)$	$[70, 158)$	$[6, 33)$	$[99, +\infty)$	$[6, +\infty)$
	EARDET	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	≈ 0	≈ 0
	FM	0	$+\infty^*$	$+\infty^*$	$+\infty^*$	$+\infty^*$	$+\infty^*$	≈ 0	≈ 0
	AMF	$+\infty$	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0
Hybrid	CLEF	0	$[512, +\infty)$	$[158, 512)$	$[33, 45)$	$[70, 158)$	$[6, 33)$	≈ 0	≈ 0
	AMF-FM	0	$+\infty^*$	$+\infty^*$	$+\infty^*$	$+\infty^*$	$+\infty^*$	≈ 0	≈ 0

Comparison in a 40 Gbps link with threshold rate $\gamma = 400$ Kbps. Each of Twin-RLFD, EARDET, FM and AMF has $m = 100$ counters (each of single RLFD has 50 counters), and thus each of CLEF and AMF-FM has 200 counters. In Twin-RLFD and CLEF, detection cycles $T_c^{(1)} = T_c = 0.1$ sec, $T_c^{(2)} = 7.92$ sec, and number of levels is $d = 4$. Attack flows are busty flows with duty cycle of $\theta = 0.25$. The reasons for this Twin-RLFD configuration are shown in Appendix B.1.5.

*The overuse damage for FM is treated as infinity, due to the extremely low detection probability.

for each RLFD in the Twin-RLFD for a fair comparison. Each hybrid scheme has 200 counters in total to ensure fair comparison between hybrid schemes is fair.

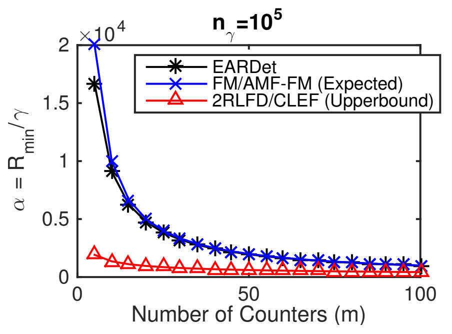
We consider both high-rate large flows ($R_{atk} \geq 250\gamma$) and low-rate large flows ($R_{atk} < 250\gamma$). 250γ is the minimum rate at which detection is guaranteed by EARDET, FM, and AMF-FM: $\frac{\theta\rho}{m} = \frac{0.25\times10^5\gamma}{100}$. Low-rate large flows are further divided into three rate intervals for thorough comparison. For each rate interval, we consider the worst-case ($\theta T_b < 2T_c$) and non-worst-case ($\theta T_b \geq 2T_c$) burst length. The duty cycle of the bursty flow is set to $\theta = 0.25$, which is challenging for CLEF. Given an average rate R_{atk}, if θ is close to 0 (close to 1), a bursty flow is easily detected by EARDET (Twin-RLFD) in CLEF.

CLEF ensures lower damage. As shown in Table 4.3, Twin-RLFD and CLEF outperform other detectors for identifying a wide range of low-rate flows. However, due to limited memory, it remains challenging for Twin-RLFD and CLEF to effectively detect large flows that are extremely close to the threshold.

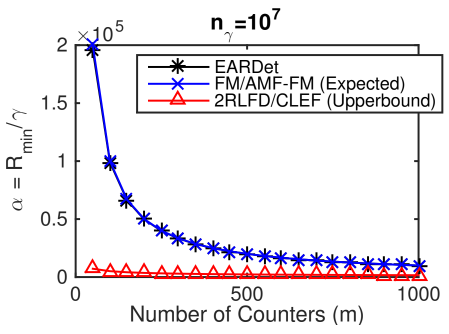
We can see that Twin-RLFD fails to limit the damage caused by high-rate large flows, because the overuse damage is linear in R_{atk} of high-rate flows (due to the minimum detection delay of one cycle). Thus, CLEF uses EARDET to limit the damage caused by high-rate flows. CLEF is better than the AMF-FM hybrid scheme. This is because the FP from AMF (with limited memory) is too high to narrow down the traffic passed to the FM in the downstream, so that the FM's performance is not improved.

CLEF is memory-efficient. We now consider the minimum rate of guaranteed detection (R_{min}) for flat flows (i.e., flat large flows ($\theta = 1.0$) exceeding the rate R_{min}) of these detectors. The R_{min} of Twin-RLFD and CLEF is bounded from above by $4\theta\sqrt{\frac{m \log n_\gamma}{n_\gamma}}\frac{\rho}{m}$ (derived from Corollary 10), which is much less than the $R_{min} = \theta\frac{\rho}{m+1}$ for EARDET and $R_{min} = \theta\frac{\rho}{m}$ for FM and AMF-FM. This is especially true when the memory is extremely limited (i.e. $n_\gamma \gg m$), where n_γ is the maximum number of legitimate flows at the threshold rate γ, and m is the number of counters for each individual detector (each RLFD in Twin-RLFD has $m/2$ counters).

Figures 4.7(a) and 4.7(b) compare the R_{min} among these three detectors given two link capacities: (1) $\rho = 10^5\gamma$ (i.e., $n_\gamma = 10^5$), and (2) $\rho = 10^7\gamma$ (i.e.,



(a) When $\rho = 10^5\gamma$



(b) When $\rho = 10^7\gamma$

Figure 4.7: Minimum Rate of Guaranteed Detection R_{min} (shown as R_{min}/γ in figures), for flat large flows ($\theta = 1.0$), when link capacity $\rho = 10^5\gamma$ and $10^7\gamma$, where γ is threshold rate. Twin-RLFD and CLEF have much lower rate of guaranteed detection than other schemes when the memory is limited.

$n_\gamma = 10^7$). The results suggest that Twin-RLFD and CLEF have a much lower R_{min} than that of other detectors when memory is limited, and the R_{min} is insensitive to memory size because RLFD can add levels to overcome memory shortage.

For bursty flows, CLEF's R_{min} is competitive to AMF-FM, due to EARDet.

4.6 Evaluation

We experimentally evaluate CLEF, RLFD, EARDet, and AMF-FM hybrid scheme with respect to worst-case damage. 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.

4.6.1 Experiment Settings

Link settings. Since the required memory space of a large-flow detector is linear 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 Section 4.4.4). The flow specification is set to $\mathsf{TH}(t) = \gamma t + \beta$, where β is set to 3028 bytes (which is as small as two maximum-sized packets, making bursty flows easier to catch).

The results on this 1 Gbps link allow us to extrapolate detector performance to high-capacity core routers, e.g., in a 100 Gbps 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_γ (as discussed in Section 4.5.3), 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 Appendix B.1.2 and B.1.1). 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, using Theorem 7 to determine worst-case traffic. 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 scheme against large flows with various average rates R_{atk} and duty cycle θ. Their bursty period is set to be $T_b = 0.967$ seconds. To evaluate RLFD and CLEF against their worst-case busty flows ($\theta T_b < 2T_c$), large flows are allotted a relatively small bursty period $T_b = 4T_\ell = 0.967$ seconds, where $T_\ell = \beta/\gamma = 0.242$ seconds is the period of each detection level in the single RLFD. In CLEF, RLFD$^{(1)}$ uses the same detection level period $T_\ell^{(1)} = T_\ell = 0.242$ seconds as well. Since RLFD usually has $d \geq 3$ levels and $T_c \geq 3T_\ell$, it is easy for attack flows to meet $\theta T_b < 2T_c$.

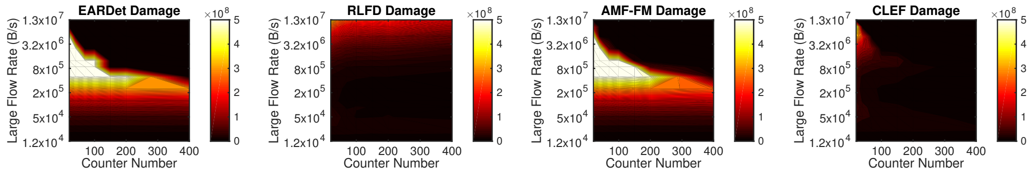
In each experiment, we have 10 artificial large flows whose rate is in a range of 12.5 KB/s to 12.5 MB/s (namely, 1 to 1000 times that of threshold rate γ). The fewer large flows in the link, the longer delay required for RLFD and CLEF to catch large flows; however, the easier it is for AMF-FM to detect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

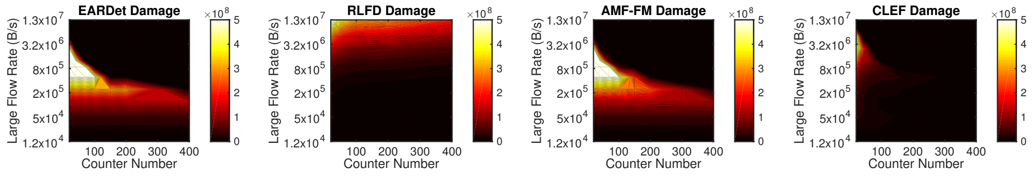
- **EARDet.** We set the low-bandwidth threshold to be the flow specification $\gamma t + \beta$, and compute the corresponding high-rate threshold for a given number of counters m as in [1]. The high-rate threshold rate is $\gamma_h = \frac{\rho}{m+1}$.
- **RLFD.** A RLFD has d levels and m counters. We set the period of a detection level as $T_\ell = \beta/\gamma = 0.242$ seconds.[5] $d = \lfloor 1.2 \times \log_m(n) \rfloor + 1$ 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.
- **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 Table B.1 (Appendix B.3).
- **AMF-FM.** We allocate $m/2$ counters to AMF and $m/2$ counters to FM. AMF has four stages (a typical setting from [2]), each of which contains $m/8$ counters. According to the flow specification $\gamma t + \beta$, all m counters are leaky buckets with a drain rate of γ and a bucket size β.

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

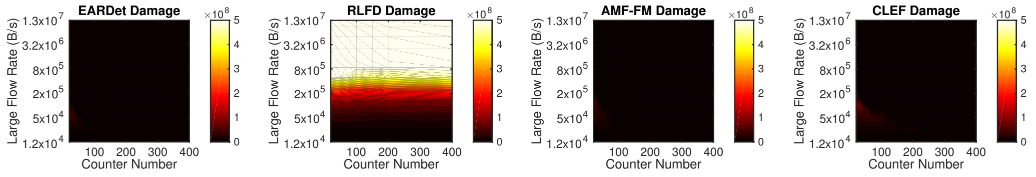
We further test CLEF with different memory allocation ratio $\phi = m_{\text{EARDET}} : m_{\textit{Twin-RLFD}}$, where $m_{\textit{Twin-RLFD}}$ is the number of counters in Twin-RLFD (each RLFD has $m_{\textit{Twin-RLFD}}/2$ counters) and m_{EARDET} is the number of counters in EARDET. In this test, we fix the total number of counters $m = m_{\textit{Twin-RLFD}} + m_{\text{EARDET}} = 200$.



(a) Flat large flows, $\theta = 1.0$



(b) Bursty large flows, $\theta = 0.50$



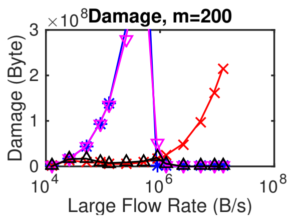
(c) Bursty large flows, $\theta = 0.02$

Figure 4.8: Damage (in Bytes) caused by 200-second large flows at different average flow rate R_{atk} (in Byte/s) and duty cycle θ 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 5×10^8. CLEF outperforms other schemes in detecting flat flows, and has competitive performance to AMF-FM and EARDET over bursty flows.

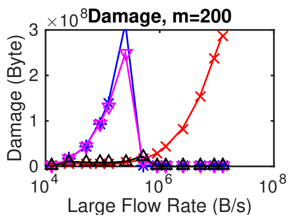
4.6.2 Experiment Results

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

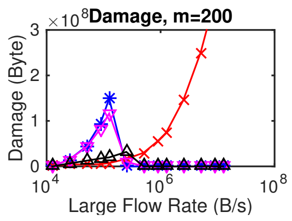
Figures 4.8(a) to 4.8(c) demonstrate the damage caused by large flows at different average rates, duty cycles, and number of detector counters during 200-second experiments; the lighter the color, the higher the damage. The damage $\geq 5 \times 10^8$ Byte is represented by the color white. Figures 4.9(a)



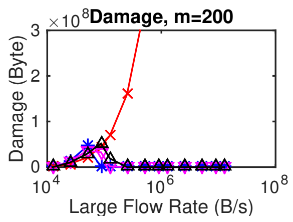
(a) Flat, $\theta = 1.0$



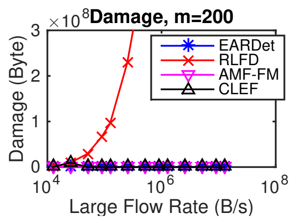
(b) Bursty, $\theta = 0.50$



(c) Bursty, $\theta = 0.25$

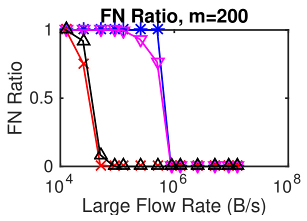


(d) Bursty, $\theta = 0.10$

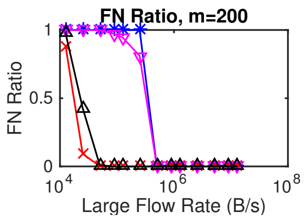


(e) Bursty, $\theta = 0.02$

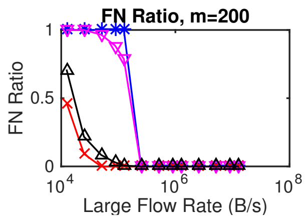
Figure 4.9: Damage (in Bytes) caused by 200-second large flows at different average rate R_{atk} (in Byte/s) and duty cycle θ. Each detection scheme uses 200 counters in total. The clear comparison among schemes suggests CLEF outperforms others with low damage against various large flows.



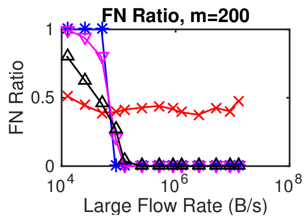
(a) Flat, $\theta = 1.0$



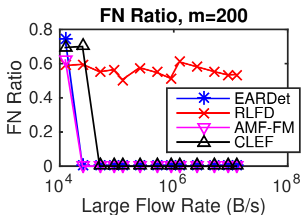
(b) Bursty, $\theta = 0.50$



(c) Bursty, $\theta = 0.25$



(d) Bursty, $\theta = 0.10$



(e) Bursty, $\theta = 0.02$

Figure 4.10: FN ratio in a 200-second detection for large flows at different average rate R_{atk} (in Byte/s) and duty cycle θ. 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.

to 4.9(e) compare damage in cases of different detector schemes with 200 counters. Figures 4.10(a) to 4.10(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 < 1.0, it is able to detect remaining large flows at some point in the future.

CLEF ensures low damage against flat flows. Figures 4.8(a), 4.9(a), and 4.10(a) support our theoretical analysis 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 4.8(a)) for CLEF shows that CLEF is effective for both high-rate and low-rate flat flows with different memory limits. Figure 4.9(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 4.10(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. This supports our theoretical analysis of AMF-FM and EARDet in Table 4.3: the infinite damage by low-rate flows against AMF-FM and EARDet.

CLEF ensures low damage against various bursty flows. Figures 4.8(b) and 4.8(c) demonstrate the damage caused by bursty flows with different duty cycle θ. The smaller the θ 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}/θ, which increases as θ 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 4.4.6, when the burst duration θT_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 4.9(b) to 4.9(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 θ 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.

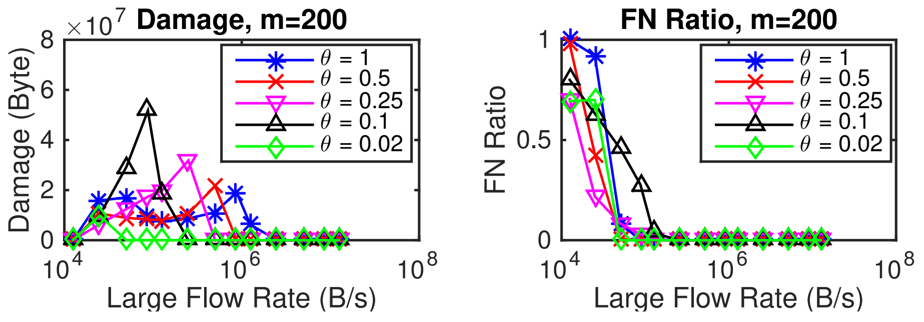
CLEF outperforms others in terms of FN and FP. To make our comparison more convincing, we examine schemes with classic metrics: FN and FP. Since we know all four schemes have no FP, we simply check the FN ratios in Figures 4.10(a) to 4.10(e). Generally, CLEF has a lower FN ratio than do AMF-FM and EARDet. CLEF can detect large flows at a much lower rate with a zero FN ratio, and is competitive with AMF-FM and EARDet against very bursty flows (e.g., Figures 4.10(b) and 4.10(e)).

CLEF is memory-efficient. Figures 4.8(a) to 4.8(c) show 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 large amounts 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 4.8(a), CLEF ensures low damage (shown in black) with tens of counters, while AMF-FM suffers from high damage (shown in light colors), even with 400 counters. This supports our theoretical results in Figures 4.7(a) and 4.7(b).

CLEF is effective against various types of bursty flows. Figures 4.11(a) and 4.11(b) demonstrate the changes of damage and FN ratio versus different duty cycles θ when CLEF is used to detect bursty flows. In the 200-second evaluation, as θ 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.

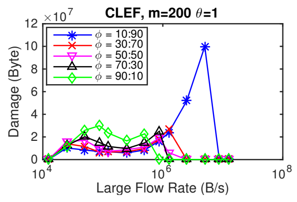
However, the maximum damage does not increase all the way as θ decreases, because when θ is getting smaller, EARDet is able to catch bursty flows with a lower average rate. This explains the lower damage from large flows in the 200-second timeframe. Figure 4.11(b) shows that the FN ratio curve changes within a small range as θ 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.



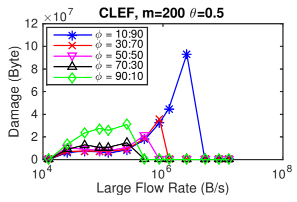
(a) Damage (b) FN Ratio

Figure 4.11: Damage and FN ratio for large flows at different average rate R_{atk} (in Byte/s) and duty cycle θ under detection of CLEF with $m = 200$ counters. CLEF is not sensitive to bursty flows across duty cycles: (1) the damages are around the same scale (not keep increasing as duty cycle decrease, because of EARDet), and (2) the FN ratios are stable and similar.

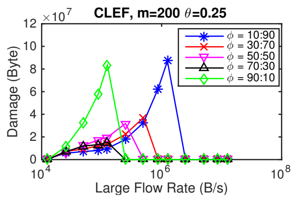
Tuning CLEF's performance by changing memory allocation strategy. Figures 4.12(a) to 4.12(e) compare damage caused by CLEFs with different memory allocation strategies between EARDet and Twin-RLFD, when we fix the total number of counters at 200. Figures 4.13(a) to 4.13(e) show the percentage of FNs produced by CLEFs with different memory allocation strategies. These figures demonstrate that the more counters allocated to Twin-RLFD, the lower damage caused by flat flows; on the contrary, the more counters allocated to EARDet, the lower damage caused by bursty flows. Thus, we may dynamically change the memory allocation strategy so that CLEF can detect both flat and bursty flows with strong performance.



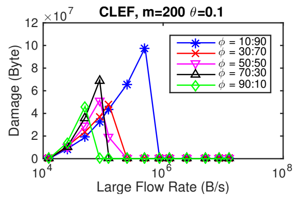
(a) Flat, $\theta = 1.0$



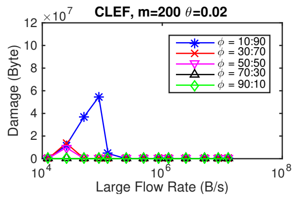
(b) Bursty, $\theta = 0.50$



(c) Bursty, $\theta = 0.25$

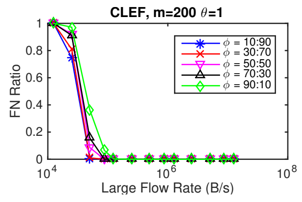


(d) Bursty, $\theta = 0.10$

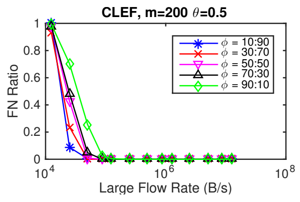


(e) Bursty, $\theta = 0.02$

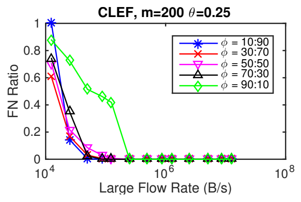
Figure 4.12: Damage (in Bytes) caused by 200-second large flows at different average rate R_{atk} (in Byte/s) and duty cycle θ. Each CLEF scheme allocate 200 counters to EARDet and Twin-RLFD according to the ratio $\phi = \frac{\#\ of\ \text{EARDET}\ counters}{\#\ of\ Twin\text{-}RLFD\ counters}$.



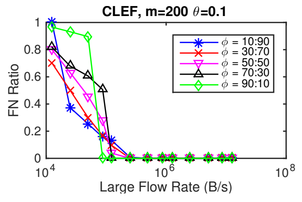
(a) Flat, $\theta = 1.0$



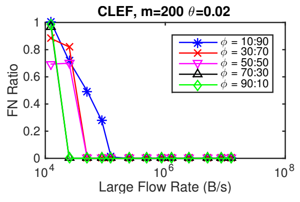
(b) Bursty, $\theta = 0.50$



(c) Bursty, $\theta = 0.25$



(d) Bursty, $\theta = 0.10$



(e) Bursty, $\theta = 0.02$

Figure 4.13: FN ratio in a 200-second detection for large flows at different average rate R_{atk} (in Byte/s) and duty cycle θ. Each CLEF scheme allocates 200 counters to EARDet and Twin-RLFD according to the ratio $\phi = \frac{\#\ of\ \text{EARDET}\ counters}{\#\ of\ Twin\text{-}RLFD\ counters}$.

Moreover, such dynamical memory allocation may help prevent attackers from gaining the maximum damage by choosing the worst-case attack traffic.

4.7 Timing-Randomized CLEF in Adversarial Environment

In this section, we further discuss the performance of CLEF with *timing-randomized RLFD* against adversarial attacks. In Section 4.4.6, we introduce that the timing-randomized RLFD randomly chooses the detection cycle T_c, so that it can prevent attackers from deterministically maximizing damage; here, we explore how good CLEF can achieve to limit the damage caused by an optimal attack.

4.7.1 Timing-Randomized RLFD

To randomize the detection cycle of a RLFD, we randomly pick up a detection cycle T_c at the beginning of each detection cycle according to a probability distribution $Pr(T_c)$. For simplicity in the analysis, we set each detection level period T_ℓ equally as T_c/d, where d is number of levels in one cycle.

By timing randomization, a single RLFD may act as a Twin-RLFD, because we can switch the detection cycle T_c of a single RLFD between the two detection cycles of two RLFDs in Twin-RLFD (i.e. $T_c^{(1)}$ and $T_c^{(2)}$). Thus, without of loss of generalizability, we only discuss the performance of a single timing-randomized RLFD in this section.

Although switching T_c in a single RLFD between $T_c^{(1)}$ and $T_c^{(2)}$ can mimic a Twin-RLFD, to prevent attackers from deterministically maximizing damage, T_c should also be set as some values in $[T_c^{(1)}, T_c^{(2)}]$. The minimum detection cycle is $T_c^{(1)}$ and the maximum detection cycle is $T_c^{(2)}$. To clearly formulate the problem, we denote $T_c^{(1)} = T_{c,min}$, $T_c^{(2)} = \sigma T_{c,min}$, and the minimum detection level period $T_{\ell,min} = T_{c,min}/d$.

4.7.2 Adversary Model

We assume attackers know the probability distribution of the detection cycle, but have no knowledge of secret seeds used to generate random variables. To learn the upper bound damage caused by the optimal attack, we further assume that the attacker can arbitrarily manipulate the timing of each packet in the large flow. Thus, for each timing-randomized RLFD, an attacker can create an optimal attack against the detection cycle probability distribution. These strong assumptions for attackers are very challenging to CLEF and impractical for real attackers, thus the performance of CLEF against attackers in the real world should be much better than the worst-case performance in such an extreme case.

4.7.3 Optimal Attack Traffic Problem

The optimal attack traffic is the large flow that can cause the maximum damage before it is detected by CLEF.

Since RLFD identifies large flows according to the total traffic amount during a detection level, thus the detection level period $T_{\ell,min}$ of the minimum detection cycle $T_{c,min}$ is the time granularity to the attacker. The attacker only consider how much traffic amount to send during each $T_{\ell,min}$ slot, but there is no need to manipulate the traffic inside a time slot $T_{\ell,min}$, because it does not change the detection results of RLFD. To avoid the detection by EARDet as much as possible, the attacker should always send flat traffic during each time slot $T_{\ell,min}$, because burstier traffic is easier for EARDet to detect.

As shown in Section 4.5.3, the single-level detection probability for a large flow is close to 1 for most of large flows, unless the large-flow rate is too close to the threshold rate γ of the flow specification. Therefore, if a large flow sends traffic during a time slot $T_{\ell,min}$, the best strategy is to send traffic at the rate close to EARDet's high-rate threshold γ_h to maximize the damage caused in this time slot $T_{\ell,min}$, while not being detected by EARDet (because it is below the γ_h). Because a successful RLFD detection requires that a large flow sends traffic in all detection levels in one detection cycle, a large flow can send traffic at rate close to γ_h in $d-1$ levels and keep silent in the one level left, so that the large flow can maximize the damage in one detection cycle. However, due to the timing randomization applied in RLFD, the attacker cannot deterministically learn its current state, such as which detection level is in the current detection cycle and how long the detection cycle and detection level are. At the best, the attacker can only learn a probability distribution for the current state, and make decision for the next time slot $T_{\ell,min}$ to maximize the expected damage caused by the large flow before being caught.

A large flow that aggressively sends traffic in all time slots will be caught shortly; while a large flow that conservatively sends traffic may cause relatively low damage. Thus a smart attacker should try to over-send traffic above the threshold rate as much as possible while not being detected; if the attacker cannot avoid being detected while over-sending traffic (when the threshold rate is larger than the maximum large-flow rate at which the large flow can be undetectable), it should try to maximize the over-sent traffic during the lifetime of the large flow.

Brute-force search for the optimal attack. Figure 4.14 describes how we formulate the problem of finding the optimal attack traffic into a brute-force search problem. First, we divide the time axis into time slots of $T_{\ell,min}$. As we discussed above, in a time slot, an optimal large flow either sends traffic at the rate close to γ_h or does not send any traffic. Thus, for each time slot, there are two possible states: sending traffic ("1"), and sending no traffic ("0"). For a chunk of large-flow traffic over N time slots, there are 2^N possible traffic patterns.

In our problem, it is unrealistic to traverse every traffic pattern, because some traffic patterns may result in undetectable large flows, which have an infinite lifetime. Thus, we have to assume that the traffic patterns are in periods, traffic pattern repeats in each period. Unfortunately we have to set the maximum period size P in the brute-force search, otherwise, the problem is unsolvable. We will search for traffic pattern for each period less than or equal to P, and find the traffic pattern that causes the most damage as the optimal traffic pattern. Although theoretically, the solution of this problem is not strictly proved as an optimal problem, it can still help us understand CLEF's performance against adversarial environments.

The value P is limited by the computation resource. If we set P as a very large value, the brute force is hard to terminate shortly with limited computation power. Intuitively, we set P as a value around $T_c^{(2)}/T_{\ell,min} = \sigma d$, because the attack traffic that can avoid detection should have the similar periodic behavior as that of the detection cycles, therefore setting P as the number of time slots in a maximum detection cycle $T_c^{(2)}$ is a good guess.

There are at least 2^P traffic patterns, and it is still hard to use a brute-force search for a real example of links requiring high σ (e.g. ≥ 1000). Thus, we further propose an efficient approximation by reinforcement learning [53] with neural networks [54].

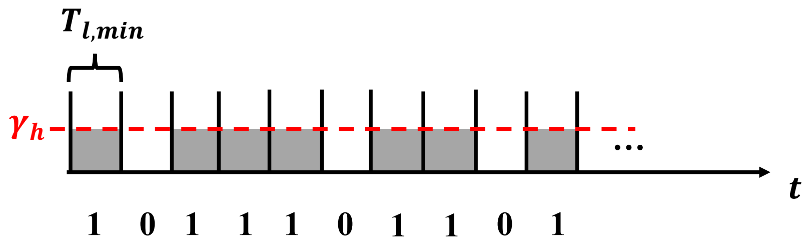


Figure 4.14: Brute-force search for the optimal attack.

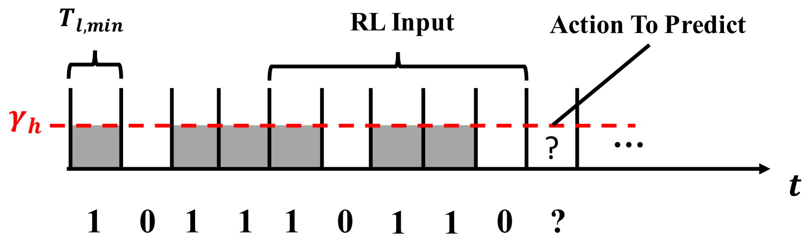


Figure 4.15: Reinforcement learning (RL) for the optimal attack.

Reinforcement learning for the optimal attack. We formulate the problem of finding the optimal attack as a reinforcement learning problem, in which the learner decides whether it should send traffic (at the rate close to γ_h) in the incoming time slot (i.e. decide to take action "0" or "1") based on the states of the most recent $W_{in} = T_c^{(2)}/T_{\ell,min} = \sigma d$ time slots in the past (as in Figure 4.15). Thus, we take the states of W_{in} time slots as the input vector of the reinforcement learner, and the output is a probability of sending traffic in the incoming time slot (i.e. taking action "1"). Although it is best to take all history states in the past as input, it is infeasible to feed such length-unfixed and very long input vectors to a neural network. Thus, we truncate the history states and only use the most recent W_{in} time slots, because $W_{in} = \sigma d$ must include all time slots in the most recent detection cycle, which is intuitively the most relevant information for the attacker to deduce how long the current cycle is and which level it is currently in.

Neural network for reinforcement learning. We currently use a fully connected neural network with three hidden layers, each of which has $2W_{in}$

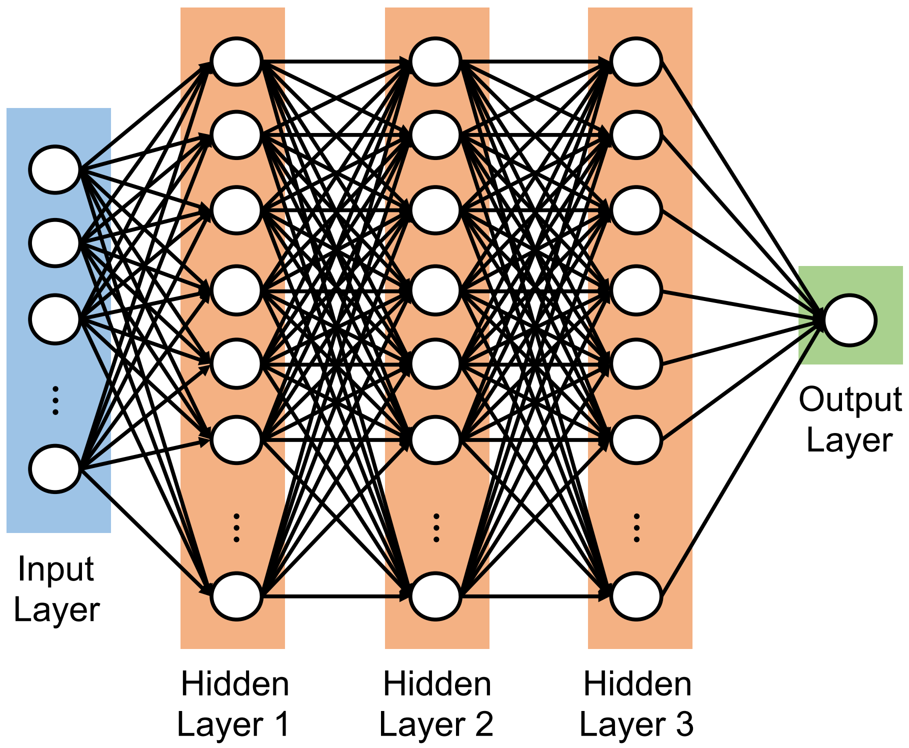


Figure 4.16: Reinforcement learning neural network (a fully connected network).

hidden nodes. The input layer has W_{in} nodes, which are the 0/1 states of the last W_{in} time slots; the output layer only has one node, which is the probability (from 0 to 1) of sending traffic in the next time slot. The activation function of hidden nodes is ReLU [55] for the fast model converging, and the activation function of the output node is Sigmoid function [56] limiting the output value from 0 to 1.

Because our target to maximize is the damage caused by the large flow after the action made by the output of the neural network, the reward of each action A_t at the time slot t is $r_t = 1$ when we send traffic, and $r_t = 0$ when we send nothing. For a input vector $[S_{t-W_{in}}, ..., S_{t-2}, S_{t-1}]$, the target to maximize is $\sum_{i \geq t} \lambda^{i-t} r_i$, where S_i denotes the state value of time slot i. The reward discount $\lambda = 0.99$ is a typical value used in reinforcement learning.

However, such an experiment with finite steps cannot represent a good attack in infinitely long experiments, because the attackers are more aggressive to take the risk of being detected when it is closer to the end of a finite experiment, while a good attack in the infinite experiment may not always be detected so that it can cause more damage in the future.

Thus, we modify the optimization target above in two folds. First, we add additional p punishment to the learner, if the large flow is caught. Second, for an action, we only consider the reward itself (i.e. at time slot t) and rewards of the next $\sigma d - 1$ steps in the future, because this action cannot make the large flow detected after σd time slots, which is out of the range of the largest detection cycle $T_c^{(2)}$. Therefore the $target(A_t)$ (for action at time slot t) to maximize for an input vector $[S_{t-W_{in}}, ..., S_{t-2}, S_{t-1}]$ is as follows:

$$target(A_t) = -p + \sum_{i=t}^{t+\sigma d-1} \lambda^{i-t} r_i \tag{4.5}$$

We set $p = \sigma d$, so that being detected is the worst case to an action, and the attacker will try to maximize the damage in the future σd steps, while avoiding the detection.

We further introduce a punishment probability $Pr(p)$ to specify how aggressive the trained attacker we want to reach. When $Pr(p) = 1$, we always apply the punishment p when the large flow is detected, and the trained attacker is the least aggressive; when $P(p) = 0$, we never apply the punishment p, and the trained attacker is the most aggressive. By tuning the $Pr(p)$ from 0 to 1, we can continuously change the aggressiveness of the trained attacker, so that we can explore the attack space. In some cases with a high threshold rate, conservative attackers may send traffic at rates even below the threshold rate and do not cause damage at all, while an aggressive attacker may cause more damage (even though it may be detected sooner). Therefore, it is necessary for us to explore the attack space by changing the aggressiveness of the attackers, and find the optimal attack that causes the most damage.

Because the neural network does gradient decent to converge the solution to the optimal one, it can avoid the brute-force computation complexity which increases exponentially by the maximum period $P = \sigma d$.

4.7.4 Experiment Settings

Experiment context. In preliminary experiments, we test our reinforcement learning method in a small-scale example with $d = 4$ levels and

$\sigma = T_c^{(2)}/T_c^{(1)} = 10$ (therefore punishment $p = 40$), and compare the reinforcement learning results with the ones from brute-force search with maximum period $P = 24$ (due to limited computation resource). The goal is to testify that the reinforcement learning results is a good approximation for the one from brute-force search.

At the beginning of each detection cycle, the RLFD randomly pick up T_c from $[1, 10] \times T_c^{(1)}$ according to the probability distribution $Pr(T_c = i \times T_c^{(1)}) = \frac{1/i}{\sum_{j=1}^{j=10} 1/j}$, where $i = 1, 2, 3, ..., 10$. Thus, RLFD can run detection cycles with different T_c with equal duty time in average.

Reinforcement learning setting. The batch size is 10 episodes, which means we update our network model parameters every 10 episodes. The beginning of a episode is when we reset the RLFD and pass a new large flow through the RLFD; the end of an episode is either when the large flow is caught by the RLFD, or the number of time slots that has been run reaches our maximum number which is set as 1000 in our experiment.

Every 1000 episodes for training, we test the model by running it for 100 episodes and average the damage caused by the large flow, so that we can learn whether the model converges at some point in the training procedure.

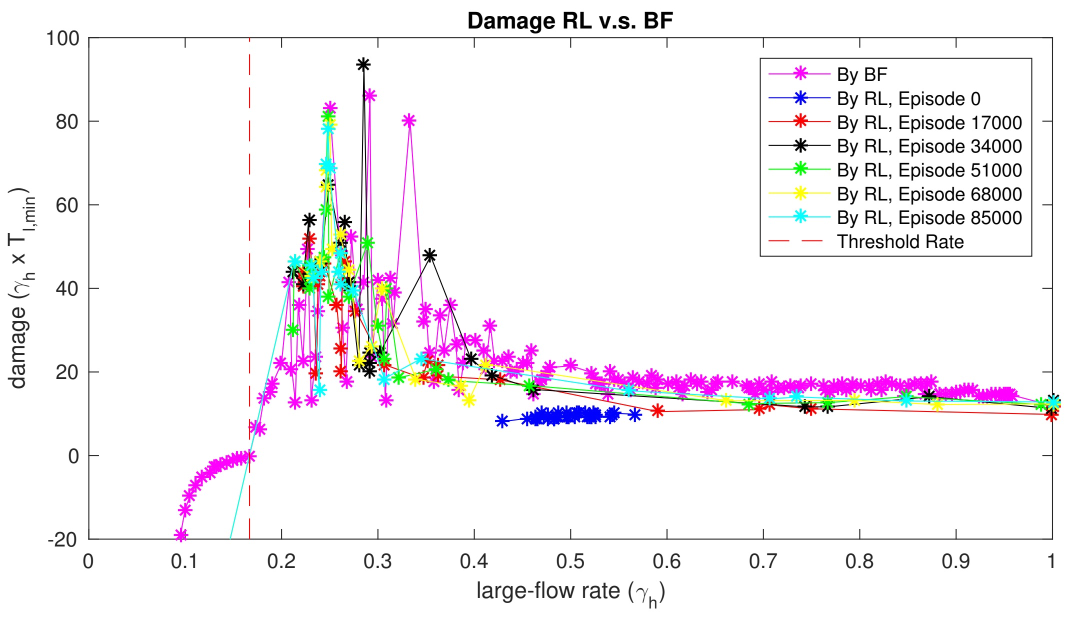
Neural network setting. The neural network has 40 input nodes and 80 hidden nodes in each hidden layer. We use the Adam optimizer [57] with learning rate of 0.001.

4.7.5 Experiment Results

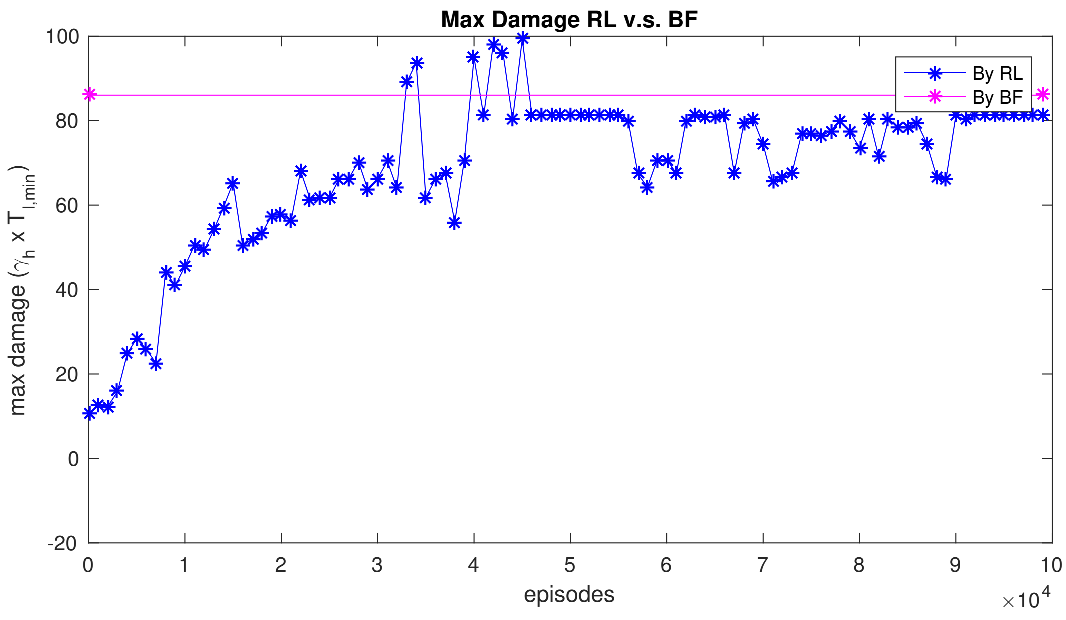
Figures 4.17(a) and 4.17(b) present the damages caused by large flows produced by reinforcement learning and brute-force search, when the threshold rate is 0.167 times the EARDET high-rate threshold.

Reinforcement learning can converge to optimal attacks. Figure 4.17(a) shows the damage caused by large flows at different average rates. First, the damage caused by flows from brute-force search shows the maximum damage that a large flow at a specific rate can achieve.[6] As we expected, the most aggressive attacker sends the traffic at the rate of the EARDET high-rate threshold γ_h can only cause a relatively low damage

[6]With the assumption that the maximum period of the traffic pattern is $P = 24$.



(a) Damage at different rates, threshold rate $\gamma = 0.167 \times \gamma_h$.



(b) Maximum damage by episode, $\gamma = 0.167 \times \gamma_h$.

Figure 4.17: Damage caused by large flows produced by reinforcement learning (RL) and brute-force search (BF), when the threshold rate is $\gamma = 0.167 \times \gamma_h$. The γ_h is the high-rate threshold of EARDET in CLEF.

(less than $20\gamma_h T_{\ell,min}$) in this case, because such large flows are caught in a short time. However, a large flow at a rate of $0.29\gamma_h$ can produce damage as high as $85\gamma_h T_{\ell,min}$, because it stays hard to detect while sending traffic at a non-trivial rate.

Then, the damages caused by large flows produced by reinforcement learning are gradually close to the brute-force-search damages as the the number of episodes increases. The reinforcement-learning damage is even higher than the brute-force-search damage at some points, for example, the damage at the rate of $0.29 \times \gamma_h$. A reasonable explanation is that our brute-force search assumes the maximum period of the traffic pattern is $P = 24$; however, the reinforcement learning results do not have such an assumption, and may find attacks even better than the brute-force-search attacks.

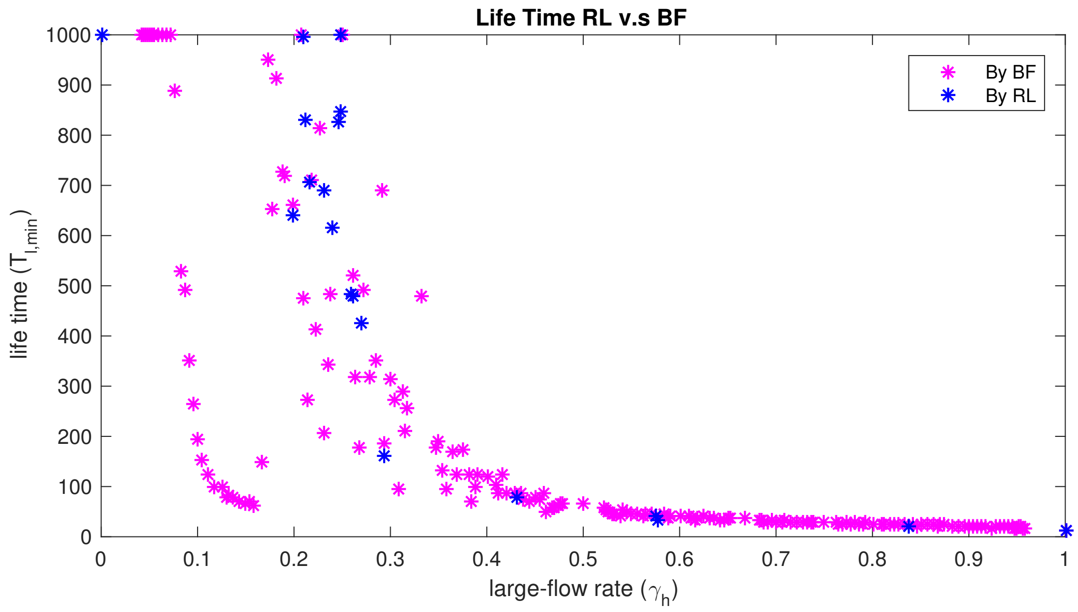
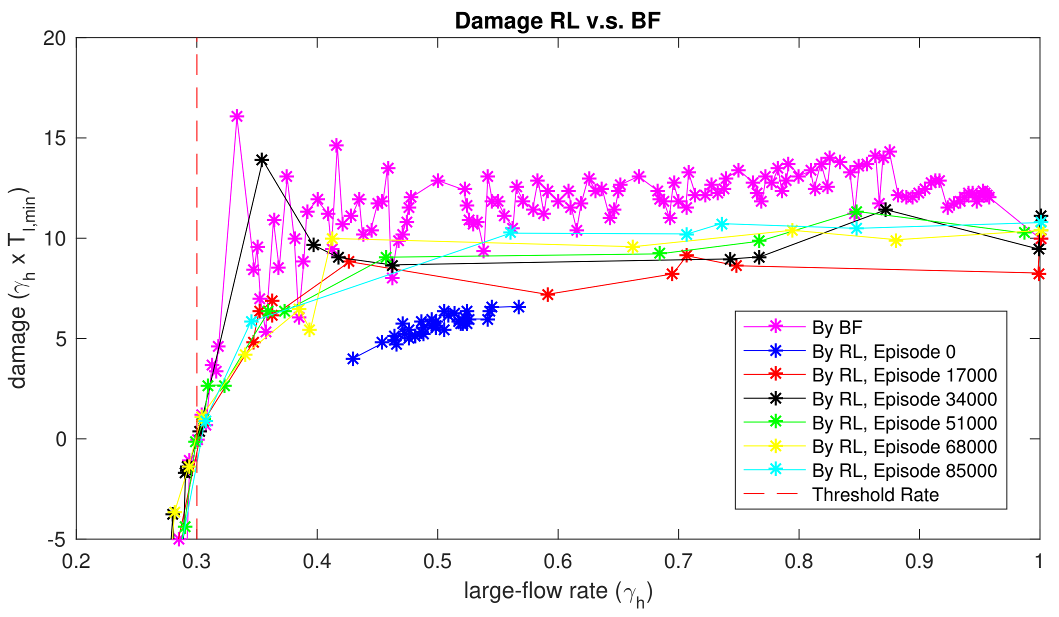
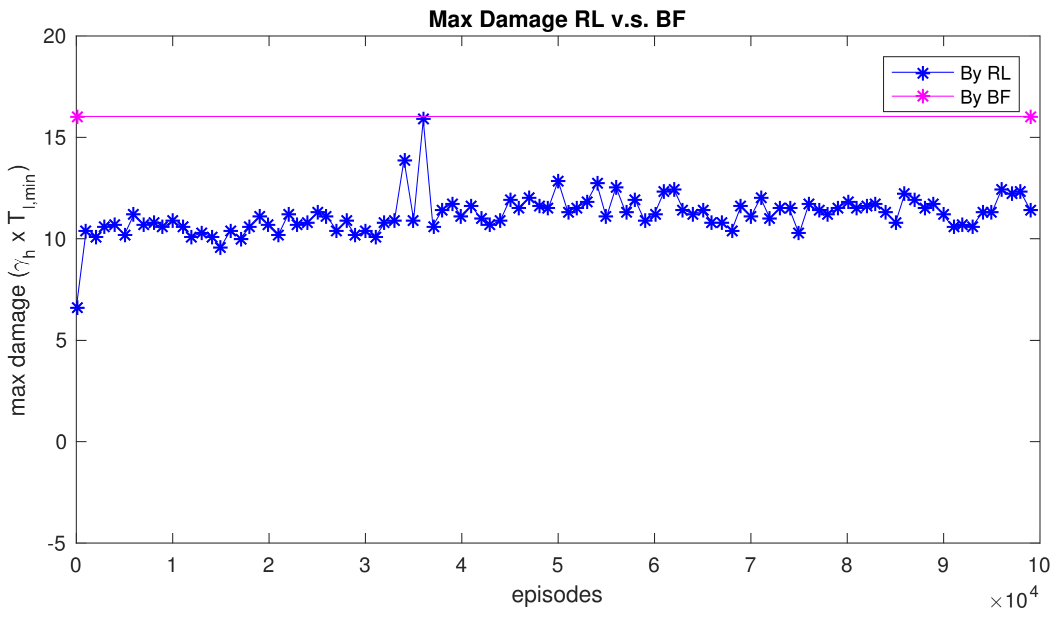


Figure 4.18: Lifetime of large flows at different rates.

Reinforcement learning is computational efficient. Figure 4.17(b) demonstrates the maximum damage caused by all large flows at various rates at each episode of the reinforcement learning. We can see the maximum damage is approaching the maximum brute-force-search damage as the number of episodes increases, which suggests that our model is converging to the optimal attack. This process only needs around $100 \times 3 \times 10^4 = 3 \times 10^6$ episodes (we try 100 different punishment probability $Pr(p)$, and each trial takes 3×10^4 episodes to converge), while brute-force search requires more than $2^{24} \times 100 = 1.68 \times 10^9$ episodes (because we need to calculate an average damage across 100 repeated episodes for each traffic pattern out of at least 2^{24}). This supports that the reinforcement learning is much more efficient



(a) Damage at different rates, threshold rate $\gamma = 0.3 \times \gamma_h$.



(b) Maximum damage by episode, threshold rate $\gamma = 0.3 \times \gamma_h$.

Figure 4.19: Damage caused by large flows produced by reinforcement learning (RL) and brute-force search (BF), when the threshold rate is $\gamma = 0.3 \times \gamma_h$. The γ_h is the high-rate threshold of EARDET in CLEF.

than brute-force learning in this task. The computation requirement gap between the reinforcement learning and the brute-force search is much larger in the large-scale examples in the real core routers.

RLFD's guarantee in large-flow detection. Figure 4.18 presents the lifetime of large flows at different rates. The large flows are produced by both brute-force search and reinforcement learning at the final episode. It shows that the large flows with high damage in Figure 4.17(a) have a long lifetime, and some of them are not detected at all (i.e. lifetime is the maximum time limit $1000T_{\ell,min}$ in our experiments). That means RLFD in this setting can guarantee detection for large flows at rates higher than $0.25\gamma_h$. In a large-scale example with $T_c^{(2)}/T_c^{(1)} = \sigma > 100$, RLFD may guarantee detection of large flows at lower rates, but with a higher damage (because the average T_c is longer).

Figures 4.19(a) and 4.19(b) show the results in a different threshold rate $\gamma = 0.3 \times \gamma_h$. With higher threshold rate, RLFD can guarantee a lower damage limit, because large flows at rates higher than this threshold rate are hard to escape from the detection.

4.7.6 Future Work

The experiment results in the small-scale example suggest the efficiency and effectiveness of using reinforcement learning to find optimal attack patterns. In the future, we should try a more efficient reinforcement learning model and test it on a large-scale example which is closer to the practice. For example, a one-dimensional convolutional neural network (CNN) [58] would be a good try for large-scale examples. The CNN comes with fewer parameters than a fully connected network for the same input and output scale, thus the model is easier to converge and uses less computation power to train.

4.8 Chapter Summary

This chapter describes two new 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 propose 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 smaller large flows. We proved that CLEF is able to guarantee low-damage large-flow detection against various attack flows with extremely 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. We believe that CLEF makes deployment of large-flow detection finally practical even for core routers.

We further explore CLEF's performance when we randomize its timing in RLFD. We propose a reinforcement-learning-based method to find optimal attacks to such timing-randomized CLEF, and the preliminary experiment results show our method is much more computational efficient than a brute-force search. The results demonstrate the timing-randomized CLEF guarantees a good detection performance in such adversarial environments.

CHAPTER 5

DISSERTATION SUMMARY

Many malicious behaviors threatening service providers are anomalies different from legitimate operations. In this dissertation, we focus on mitigating the damage caused by large Internet flows, which are collections of related traffic consuming more than the resource allocated to them. This dissertation reviews some basic knowledge and typical existing approaches in the large-flow detection problem, and identifies the shortcomings of current work.

First, we propose EARDet, a novel arbitrary-window-based algorithm, which is exact outside an ambiguity window. EARDet not only inherits the property of no false negative over a high-bandwidth threshold (from MG algorithm), but also has no false positive on small flows complying a low-bandwidth threshold. EARDet guarantees deterministic detection that does not make any assumption on the input traffic or attack traffic pattern. Our theoretical and experimental evaluation demonstrate these properties in EARDet, and show it outperform existing algorithms in detecting both flat flows and bursty flows.

Although EARDet guarantees exact detection over large flows above high-bandwidth with limited memory, the detection for large flows falling in the ambiguity region is still not guaranteed. Then, we propose a randomized Recursive Large-Flow Detection (RLFD) scheme, which uses very little memory yet provides eventual detection of persistently large flows in the ambiguity region of EARDet. Unlike existing detectors, RLFD efficiently distinguishes large flows from legitimate flows by evaluating one set of flows at a time, and recursively shrinking the set of suspected large flows. Because of the pigeonhole principle, RLFD cannot guarantee immediate exact detection with limited memory. Thus, larger flows are detected with higher probability in RLFD, so the expected detection time decreases in the level of overuse, resulting in limited damage across a wide range of flow overuse amounts.

Furthermore, we develop, CLEF, a hybrid scheme that combines EARDet and RLFD in parallel, which is super memory-efficient and able to scale to Internet core routers. CLEF is resilient against worst-case traffic, while none of the existing approaches can achieve same level of resilience with the same memory limitations. To compare attack resilience among various detectors, we propose a damage metric that summarizes the impact of attack traffic on legitimate traffic. With a deterministic EARDet, CLEF can guarantee the rapid detection of very large flows; with two RLFDs, CLEF can guarantee detection of smaller large flows that are in the ambiguity region of EARDet. Our theoretical analysis and experimental evaluation both support that CLEF has the lowest worst-case damage among all detectors and consistently low damage over a wide range of attack flows.

Future work. There is some future work for CLEF and its components (RLFD and EARDet). (1) CLEF is quite simple and easy to apply in industry, therefore, we want to build a real system with the CLEF algorithm and test it in the real network to see the performance in practice. (2) In the experiment, EARDet's detection probability curves under different input traffic (congested and non-congested traffic) are highly matched, even in the ambiguity region. Thus, it should be interesting to research the performance in the ambiguity region in future research. (3) For RLFD with randomized timing, we preliminarily use reinforcement learning to explore RLFD's resilience against adversarial environments, and the results in small-scale examples show that RLFD can guarantee detection and the limit damage caused by adversarial attack traffics. In the future, we should further test the timing-randomized RLFD in a realistic and large-scale example, which may require more sophisticated neural network models.

APPENDIX A

EXACT-OUTSIDE-AMBIGUITY-REGION DETECTOR

A.1 Proof Sketches for Lemmas

Note that Appendices A.1.1 and A.1.4 are presented in the technical report by Wu et al. [48].

A.1.1 Lemma 13 and Proof Sketch

Lemma 13 *In any time interval* $[t_1, t_2]$, *we assume there are k attack flows occupy k counters from the beginning time* t_1 *to the ending time* t_2. *If all the normal counters (counters except the ones occupied by attack flows) are empty at beginning time* t_1 *and ending time* t_2, *then, the decrement of all the counters is* $\frac{(t_2-t_1)-t_{lrg}}{n+1-k}\rho$, *where* t_{lrg} *is the sum of time that k attack flows are sending packets.*

Proof sketch: In $[t_1, t_2]$, because the attack flows occupied the link for t_{lrg}, the time length of $t_2 - t_1 - t_{lrg}$ is occupied by some real flows F or virtual flows (there is no assumption for the flows in F, but such flows should fulfill that normal counters are empty at beginning time t_1 and ending time t_2). In the time of $t_2 - t_1 - t_{lrg}$, sometimes the counters are increased by flows in F, and sometimes the counters are decreased by flows in F or virtual flows. Therefore, we can assume that the sum of all the decrement dec consists of many small decrements dec_i, which happen in time interval $t_{i,dec}$, and the number of counters occupied by flows in F is x_i during $t_{i,dec}$. Because the normal counters are empty at the beginning and the ending, when there is a decrement dec_i for each counter, then there must be x_i increment inc_i that happened on x_i non-empty normal counters. Therefore all the decrements dec_i in these normal counters have a counterpart of x_i increment inc_i, which takes $t_{i,inc}$ time length. Maybe dec_i and x_i values of inc_i are not neighbors in time domain, but for a decrement dec_i there must be x_i values of inc_i, such that

$$inc_i = dec_i \tag{A.1}$$

In $t_{i,dec}$, according to the three ways of decreasing and increasing the counter, when the number of empty counters is $l = n - k - x_i$, the dec_i and inc_i are as follows:

$$dec_i = \frac{\rho}{n+1-k-x_i} \cdot t_{i,dec} \tag{A.2}$$

$$inc_i = \rho \cdot t_{i,inc} \tag{A.3}$$

Then, according to (A.1), (A.2), and (A.3)

$$\Rightarrow \begin{cases} t_{i,dec} = \dfrac{(n+1-k-x_i) \cdot dec_i}{\rho} \\ t_{i,inc} = \dfrac{inc_i}{\rho} = \dfrac{dec_i}{\rho} \end{cases} \tag{A.4}$$

At any time point in $t_2 - t_1 - t_{lrg}$, counters either increase or decrease, thus

$$t_2 - t_1 - t_{lrg} = \sum_i (x_i \cdot t_{i,inc} + t_{i,dec}) \tag{A.5}$$

Then, according to (A.4) and (A.5), we can get

$$t_2 - t_1 - t_{lrg} = \sum_i (x_i \cdot \frac{dec_i}{\rho} + \frac{(n+1-k-x_i) \cdot dec_i}{\rho}) \tag{A.6}$$

$$= \sum_i \frac{(n+1-k) \cdot dec_i}{\rho} \tag{A.7}$$

$$= \frac{dec(n+1-k)}{\rho} \tag{A.8}$$

Then,

$$\Rightarrow dec = \frac{(t_2 - t_1) - t_{lrg}}{n+1-k} \rho \tag{A.9}$$

Therefore, during $[t_1, t_2]$ the decrement of all the counters is $\frac{(t_2-t_1)-t_{lrg}}{n+1-k}\rho$, and this lemma is proved. ■

A.1.2 Lemma 14 and Proof Sketch

Lemma 14 *In any time interval* $[t_1, t_2]$*, if all the counters are empty at the beginning time* t_1 *and the ending time* t_2*, then, the decrement of all the counters is* $\frac{\rho}{n+1} \cdot (t_2 - t_1)$*.*

Proof sketch: Considering the scenario of Lemma 13 , when all the counters are normal counters (namely $k = 0$), there is no assumption on all the incoming flows except the condition that all the counters are empty at the beginning time t_1 and the ending time t_2. This scenario is exactly the same to what is described in Lemma 14. Therefore, to prove Lemma 14, we just need to consider the scenario of $k = 0$ in Lemma 13. According to Lemma 13, when $k = 0$, the t_{lrg} must be 0, and therefore the decrement is

$$dec = \frac{(t_2 - t_1) - t_{lrg}}{n + 1 - k}\rho = \frac{(t_2 - t_1)}{n + 1}\rho \tag{A.10}$$

Thus, the decrement of the scenario described in Lemma 14 is $\frac{\rho}{n+1} \cdot (t_2 - t_1)$, and this lemma is proved. ■

A.1.3 Proof Sketch of Lemma 2

Proof sketch: To get the upper bound of decrement of all the counters, we just need consider the maximum decrement for a counter in time interval $[t_1, t_2]$. According to Lemma 14, we can know the decrement of each counter is $\frac{\rho}{n+1} \cdot (t_2 - t_1)$ when all the counters are empty at the beginning time t_1 and the ending time t_2. However, intuitively, the greater the values of counters are at the beginning, the greater the decrement is, because each counter saves some time to increase these counters and they have more time to decrease; also, the less the values of counters are at the ending, the more the total decrement of all counters is, because if there are remaining values in the counters, the counters must waste some time to increase the counters instead of decrease them. Because the maximum value of a counter is $\alpha + \beta_{TH}$ and the minimum value of a counter is 0, the scenario of maximum decrement is: (1) all the counters' value are $\alpha + \beta_{TH}$ at the beginning time t_1 and (2) all the counters are empty at the ending time t_2. Denote the scenario described in Lemma 14 and Lemma 2 as *CASE1* and *CASE2*. The difference between

CASE1 and *CASE2* is that counters in *CASE2* have a value of $\alpha+\beta_{TH}$ at the beginning, therefore there is an extra decrement of $\alpha+\beta_{TH}$ in *CASE2.* To have the extra decrement in *CASE2*, counters need to take some extra $t_{i,dec}$ to decrease the extra decrement, and then the decrement of *CASE2* except the extra decrement $\alpha+\beta_{TH}$ is lower than the $\frac{\rho}{n+1}\cdot(t_2-t_1)$, which is the decrement of *CASE1.* Therefore, the total decrement of *CASE2* is lower than $\frac{\rho}{n+1}\cdot(t_2-t_1)+\alpha+\beta_{TH}$, namely:

$$dec < \frac{\rho}{n+1}\cdot(t_2-t_1)+\alpha+\beta_{TH} \tag{A.11}$$

Therefore, this lemma is proved. ■

A.1.4 Proof Sketch of Lemma 4

Proof sketch: WLOG, we assume flow f is associated with a counter at $t_1=0$, and in $[0,t_{ocp}]$, flow f always occupies this counter. Then, intuitively, in $[0,t_{ocp}]$, the case to have minimum decrement dec_{min} on this counter is that: (1) at time 0 all the counters are empty; and (2) at time t_{ocp}, except the counter of flow f, all other counters have the maximum value $\alpha+\beta_{TH}$. Because the remaining values in the counter will cost extra time t_{inc} for increasing these counters, then according to Lemma 13, the t_2-t_1 in Lemma 13 is smaller and the decrement is smaller. Therefore, in the case mentioned above, the decrement is minimized. According to Lemma 13, in this case $t_2-t_1=t_{ocp}-t_{inc}$, $k=1$, then the minimum decrement is

$$dec_{min}=\frac{t_{ocp}-t_{inc}-t_{lrg}}{n}\rho \tag{A.12}$$

where $t_{inc}=\frac{(n-1)(\beta_{TH}+\alpha)}{\rho}$.

Since f complies with $TH_\ell(t)$, $t_{lrg}<\gamma_\ell/\rho\cdot t_{ocp}+\frac{\beta_\ell}{\rho}$.

$$\Rightarrow dec_{min}>\frac{t_{ocp}(1-\gamma_\ell/\rho)}{n}\rho-\frac{(\beta_{TH}+\alpha)(n-1)+\beta_\ell}{n} \tag{A.13}$$

$$\Leftrightarrow dec_{min}>\gamma_\ell\, t_{ocp}+\frac{t_{ocp}(1-(n+1)\frac{\gamma_\ell}{\rho})}{n}\rho-\frac{(\beta_{TH}+\alpha)(n-1)+\beta_\ell}{n} \tag{A.14}$$

When $t_{ocp}>t_{\beta_\ell}=\frac{(n-1)\alpha+(n+1)\beta_\ell}{[1-(n+1)\gamma_\ell/\rho]\rho}$,

$$\Rightarrow dec_{min} > \gamma_\ell \, t_{ocp} + \frac{(n-1)\alpha + (n+1)\beta_\ell}{n}\rho - \frac{(\beta_{TH}+\alpha)(n-1)+\beta_\ell}{n} \tag{A.15}$$

$$\Rightarrow \gamma_\ell \, t_{ocp} + \beta_\ell - dec_{min} < \beta_{TH} \tag{A.16}$$

Because flow f complies with $TH_\ell(t)$, its counter value is smaller than $t_{ocp} + \beta_\ell - dec_{min}$. Therefore, the counter is smaller than β_{TH} after t_{β_ℓ}. ■

APPENDIX B

IN-CORE LIMITING OF EGREGIOUS FLOWS

B.1 Additional Analysis

B.1.1 Flow Memory Analysis

We analyze the Flow Memory (FM) with random flow eviction mechanism, which is applied with multistage filters in [2]. For each incoming packet whose flow is not tracked, such FM randomly picks a flow from the tracked flows and the new flow to evict. Thus, for each packet of the flow not tracked, the existing tracked flow has a probability $P_e = \frac{1}{m+1}$ to be evicted, where m is the number of counters in the FM.

Theorem 15 *In a link with total traffic rate of R ($\leq \rho$), the packet size of S_{pkt}, and the large-flow threshold $\mathsf{TH}(t) = \gamma t + \beta$, a flow memory with m counters is able to detect large flows at rate around or higher than $\frac{\beta}{S_{pkt}} \frac{R}{m}$ with high probability.*

Proof sketch: We assume number of packets arriving at the FM per second is at the packet rate of R_{pkt}, thus the time gap between two incoming packets is $T_{pkt} = \frac{1}{R_{pkt}} = \frac{S_{pkt}}{R}$. For a newly tracked flow f at timestamp 0, the kth eviction happens at $k \cdot T_{pkt}$, and $P_e = \frac{1}{m+1}$ is the probability that flow f is evicted at the kth eviction. Evictions are not triggered by packets of flows being tracked, however the number of flows untracked is far larger than the number of flows being tracked, thus we can approximate treat the time gap between evictions as T_{pkt}. Thus, the expected time length for the flow f to be tracked is

$$\begin{aligned}E(T_{track}) &= \sum_{k=1}^{+\infty} P_e(1-P_e)^{k-1} k T_{pkt} \\ &= \lim_{k\to+\infty}(1-P_e)\left(\frac{1-(1-P_e)^k}{P_e} - (k+1)(1-P_e)^k\right)T_{pkt} \\ &= \frac{1-P_e}{P_e}T_{pkt} = m\cdot T_{pkt}\end{aligned} \tag{B.1}$$

As the FM uses leaky bucket counters to enforce the large-flow threshold $\frac{TH}{t} = \gamma t + \beta$ (defined in Section 2.1), the counter threshold is the burst threshold β. Thus, to detect a large flow at traffic rate of R_{atk}, the FM requires the large flow being tracked at least for a time of β/R_{atk}, otherwise the counter value cannot reach the threshold. Therefore,

$$R_{atk} > \frac{\beta}{E(T_{track})} = \frac{\beta}{m\cdot T_{pkt}} = \frac{\beta}{S_{pkt}}\frac{R}{m} \tag{B.2}$$

Thus for the large flows at rates far smaller than the $\frac{\beta}{S_{pkt}}\frac{\rho}{m}$ are likely to be evicted before violating the threshold β.■

In the practice, the packet size is not fixed, but we treat it with fixed size for analyzing the least R_{atk} changes along with the m. Because the real packet size is also limited in 1514 Bytes, the $\frac{\beta}{S_{pkt}}$ is a bounded factor. As the β is usually larger than the maximum packet size, the $\frac{\beta}{S_{pkt}} > 1$ for sure.

We can see the scale of the large flow rate can be detected by FM is similar to that can be detected by EARDet (i.e., $\frac{\rho}{m+1}$, where ρ is the link capacity). They both increase as $\frac{1}{m}$ increases. In the worst case of the FM, when the traffic rate is at link capacity ($R = \rho$), the least detectable average rates R_{atk} of the FM and the EARDet are at the same scale. One difference between them is that the EARDet can guarantee deterministic detection, while the Flow Memory detects flows probabilistically. Our simulations in Section 4.6 support the analysis above.

B.1.2 Multistage Filter Analysis

According to the theoretical analysis in [2], an m-counter multistage filter with d stages each of which has m/d counters, the probability for a flow hashed into a counter in each stage without collision (C_{free}) to other flows is as follows. We let $m' = m/d$, and assume there are n flows in total, then

$$\begin{aligned} Pr(C_{free}) &= 1 - (1 - (1 - \frac{1}{m'})^{n-1})^d \\ &= 1 - (1 - (1 - \frac{1}{m'})^{m' \frac{n-1}{m'}})^d \\ &\approx 1 - (1 - e^{-\frac{n-1}{m'}})^d \\ &\to 0, \ when \ n \to +\infty \end{aligned} \tag{B.3}$$

where we assume the $m' \gg 1$ and $n/m' \gg 1$. The assumptions are reasonable: (1) the number of counters m is usually around hundreds, and the d is typically chosen as 4 in [2], therefore $m' \gg 1$; and (2) we aim to use very limited counters to detect large flows from a large number of legitimate flows, thus $n/m' \gg 1$.

In the case that every legitimate flow is higher than the half of the threshold rate $\gamma/2$, the false positive rate is almost 100%, because the $Pr(C_{free})$ is close to 100%. Any collision in a counter results in that the counter value violates the counter threshold and thus a falsely positive on legitimate flows.

B.1.3 RLFD Worst-Case Background Traffic

General case: Weighted balls-into-bins problem. In the well-known balls-into-bins problem, we have m bins and n balls. For each ball, we randomly throw it into one of m bins.

We treat the flows in the network as the balls, and the counter array as the bins. Hashing flows into counters is just like randomly throwing balls into bins, where each flow is a weighted ball with weight of its traffic volume sent during a period T_ℓ of each level L_k ($1 \leq k \leq d$).

Worst case: Single-weight balls-into-bins problem. We assume the rate threshold γ of our flow specification, $\mathsf{TH}(t) = \gamma t + \beta$, is $\gamma = \frac{\rho}{N}$, where the ρ is the outbound link capacity. In the general case, the legitimate flows are at average rates less than or equal to the threshold rate γ, however we show that the worst-case background traffic for RLFD to detecting a large flow is that all legitimate flows are sending traffic at the rate of the threshold rate γ (Theorem 7). As the inbound link capacity can be larger than the outbound one, there still could be attack flows in this case. We prove the

Theorem 7 by the Theorem 16 from Berenbrink et al. [59] which is for weighted balls-into-bins games.

Theorem 16 ***Berenbrink et al. Theorem 3.1*** *For two weighted balls-into-bins games $B(w, n, m)$ and $B'(w', n, m)$ of n balls and m bins, the vectors $w = (w_1, ..., w_n)$ and $w' = (w'_1, ..., w'_n)$ represent the weight of each ball in two B and B', respectively. If $W = \sum_{i=1}^{n} w_i = \sum_{i=1}^{n} w'_i$ and $\sum_{i=1}^{k} w_i \geq \sum_{i=1}^{k} w'_i$ for all $1 \leq k \leq n$, then $E[S_i(w)] \geq E[S_i(w')]$ for all $1 \leq i \leq m$, where the $S_i(w)$ is the total load of the i highest bins, and the $E(S_i(w))$ is the expected $S_i(w)$ across all m^n possible balls-into-bins combinations.*

Lemma 17 and proof sketch.

Lemma 17 *The RLFD has the lowest probability to select the right counter of a large flow f_{atk} to the next level, when the legitimate flows use up all legitimate bandwidth.*

We assume $\mathcal{C}_1$ and $\mathcal{C}_2$ are two different counter states after adding the attack traffic and the traffic of some legitimate flows, and there are V more volume of traffic allowed to send by the other legitimate flows before the total volume of legitimate flows reaches the outbound link capacity. Let V_{atk} be the value of the counter assigned to f_{atk}, and the V_{max} be the maximum value of other counters. In the $\mathcal{C}_1$, we let $V_{atk} > V_{max} + V$; in the $\mathcal{C}_2$, we let $V_{atk} \leq V_{max} + V$. Hence, $\mathcal{C}_1$ and $\mathcal{C}_2$ cover all possible counter states. As there are still up to V volume of legitimate flows can be added into counters. We use V'_{atk} and V'_{max} to represent the final value of V_{atk} and V_{max}. Thus, the probability to select the counter of f_{atk} is

$$Pr(V'_{atk} > V'_{max}) = Pr(V'_{atk} > V'_{max}|\mathcal{C}_1)Pr(\mathcal{C}_1) + Pr(V'_{atk} > V'_{max}|\mathcal{C}_2)Pr(\mathcal{C}_2) \quad \text{(B.4)}$$

Because $V_{atk} > V_{max} + V$ in $\mathcal{C}_1$, and the V'_{max} cannot exceed $V_{max} + V$, thus always $V'_{atk} > V'_{max}$. Then,

$$Pr(V'_{atk} > V'_{max}) = Pr(\mathcal{C}_1) + Pr(V'_{atk} > V'_{max}|\mathcal{C}_2)Pr(\mathcal{C}_2) \quad \text{(B.5)}$$

Let x be the amount of legitimate traffic added into counters after $\mathcal{C}_2$, where $0 \leq x \leq V$. If the $x = V$, then there is a chance to have all V added on the V_{max}, and thus $V'_{max} = V_{max} + V = V_{atk} = V'_{atk}$, so that $Pr(V'_{atk} > V'_{max}|\mathcal{C}_2)$ is lower than that when $x < V$. Therefore, $Pr(V'_{atk} > V'_{max})$ is lower in the case that legitimate flows fully use the link capacity than other cases. Thus, the Lemma 17 is proved. ■

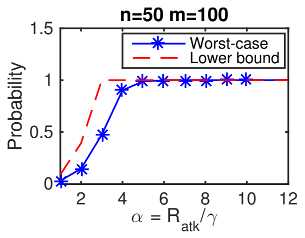
Proof sketch of Theorem 7. We first just consider the legitimate traffic but not the attack flow. As Lemma 17 illustrated, the more traffic sent from legitimate flows, the harder for RLFD to correctly select the counter with the attack flow f_{atk}, thus to have the worst RLFD detection probability, legitimate flow should use all outbound link capacity, and it requires the flow number $n \geq \rho/\gamma$.

Given $n \geq \rho/\gamma$ and m, we first construct a legitimate flow configuration $B(w, n, m)$, $w_i = \gamma$ for $1 \leq i \leq \rho/\gamma$ and $w_i = 0$ for $i > \rho/\gamma$ which is the worst-case legitimate configuration we want to prove, because there are actually only the first ρ/γ flows with non-zero rate. For any legitimate flow configuration $B'(w', n, m)$ with constraint of $\sum_{i=1}^{k} w'_i = \rho$. It is easy to find $\sum_{i=1}^{k} w_i \geq \sum_{i=1}^{k} w'_i$.

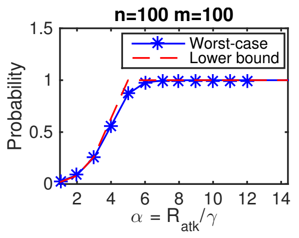
Thus, according to Theorem 16 [59] the $E[S_i(w)] \geq E[S_i(w')]$ for all $1 \leq i \leq m$, where $E[S_i(w)]$ is the expected total counter value of the i highest counters in the case of $B(w, n, m)$ and $E[S_i(w')]$ is the one in the case of any other legitimate flow configuration $B'(w', n, m)$.

It is not hard to find that the $E[S_i(w)] \geq E[S_i(w')]$ for all $1 \leq i \leq m$ suggests that the variation of expected counter values across all counters of the $B(w, n, m)$ is larger than that of the $B'(w', n, m)$. Let V_{max} be the maximum counter value, and V_{atk} be the value of the counter randomly assigned to the attack flow f_{atk} (V_{atk} does not count the traffic of f_{atk}). The higher the variation, the larger the expected $V_{max} - V_i$, thus the harder for RLFD to correctly select the counter of f_{atk} for the next level.

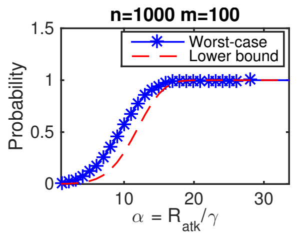
Therefore, the $B(w, n, m)$ is the worst legitimate flow configuration for RLFD to detect large flows. ■



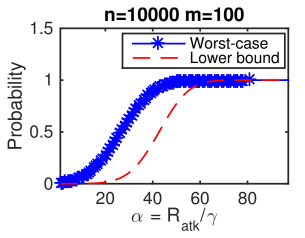
(a) $n = 50$, $m = 100$



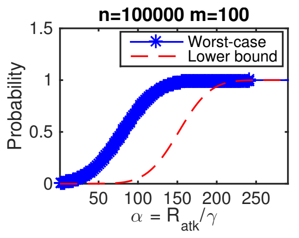
(b) $n = 100$, $m = 100$



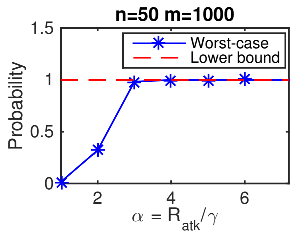
(c) $n = 10^3$, $m = 100$



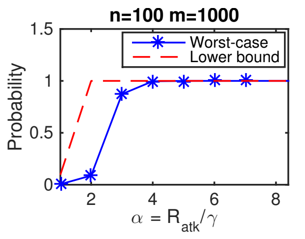
(d) $n = 10^4$, $m = 100$



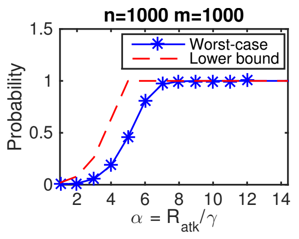
(e) $n = 10^5$, $m = 100$



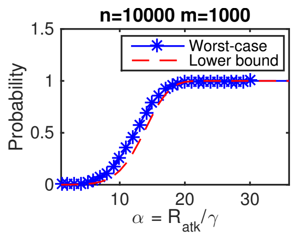
(f) $n = 50$, $m = 1000$



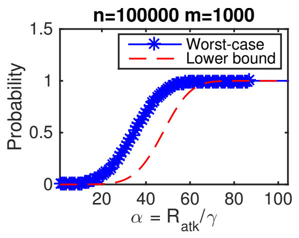
(g) $n = 100$, $m = 1000$



(h) $n = 10^3$, $m = 1000$



(i) $n = 10^4$, $m = 1000$



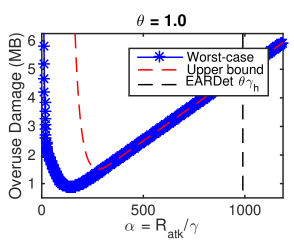
(j) $n = 10^5$, $m = 1000$

Figure B.1: The probability $\mathsf{P}_{worst}(m, n, \alpha)$ when n full-use legitimate flows (at rate of γ), m counters, and a large flow at the rate of $R_{atk} = \alpha \cdot \gamma$.

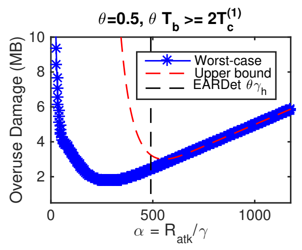
B.1.4 Numeric Analysis for RLFD Detection Probability

Numeric analysis for single-level detection. For each theoretical result, we show numeric examples in the scenario of $n_\gamma = 10^5$ and $m = 100$, a even more memory-limited setting than the one in the complexity analysis (Section 4.4.4).

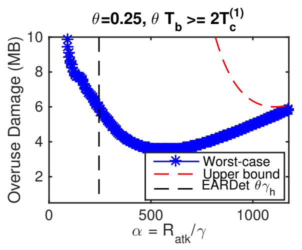
Figures B.1(a) to B.1(j) comprehensively shows the simulated worst-case detection probability $\mathsf{P}_{worst}(m, n, \alpha)$ in a level and its lower bound for various number of full-use legitimate flows $n \leq n_\gamma$ (50 to 10^5). We also give the



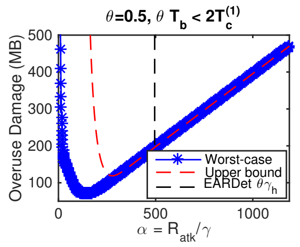
(a) $\theta = 1.0$ (Flat)



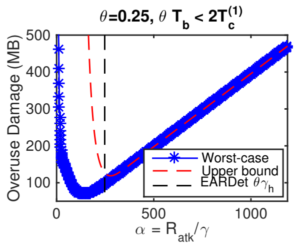
(b) $\theta = 0.5$, $\theta T_b \geq 2T_c^{(1)}$



(c) $\theta = .25$, $\theta T_b \geq 2T_c^{(1)}$



(d) $\theta = 0.5$, $\theta T_b < 2T_c^{(1)}$



(e) $\theta = .25$, $\theta T_b < 2T_c^{(1)}$

Figure B.2: Twin-RLFD worst-case expected overuse damage $E(D_{over})$ (in MBytes) and its upper bound for flat/bursty flows in various duty cycles θ, burst periods T_b, and average rates $R_{atk} = \alpha\gamma$, in the 40 Gbps link with threshold rate $\gamma = 400$ Kbps ($n_\gamma = 10^5$ full-use legitimate flows at most). The Twin-RLFD has a limited memory of $m = 100$ counters (50 counters for each RLFD), a typical number of levels $d = 4$, and detection cycle $T_c^{(1)} = 0.1$ sec, $T_c^{(2)} = 7.92$ sec for two RLFDs respectively. Flows at the EARDet detectable rate $R_{atk} \geq \theta\gamma_h = \theta\frac{n_\gamma}{m+1}\gamma$ are detected by the EARDet with $m = 100$ counters in nearly zero damage.

numeric results with different m (100 and 1000) for comparison. When $n = 10^5, m = 100$, we can see $\alpha_{0.5} = 152$ and $\alpha_{1.0} = 303$, which are far smaller than EARDet's lowest detectable $\alpha = \frac{\rho}{\gamma(m+1)} = \frac{n_\gamma}{m+1} = 991$. For RLFD, the α with actual worst-case detection probability of 0.5 and 1.0 are around 75 and 150, respectively, which are much lower than the $\alpha_{0.5}$ and $\alpha_{1.0}$. Thus, it suggests RLFD's ability of detecting low-rate large flows. The figures also show that the probability of detecting low-rate flows increases as the number of flows (n) decreases or the number of counters (m) increases. The figures show that the $\alpha_{0.5}$, $\alpha_{1.0}$, and the lower bound holds for $n > m$, because we derive it with the assumption of $n \gg m \log m$. When $n \leq m$, RLFD has 100% detection probability as explained in Section 4.4.

Since $\alpha_{1.0}$ decreases rapidly when n decreases, we approximate the total detection probability by the detection probability of the first few levels.

Numeric analysis for total detection probability. In a tough scenario with $n_\gamma = 10^5$, $m = 100$, and $n = 10^7$ legitimate flows in a link during one detection cycle (around a second), RLFD has at least 0.25 and 1.0 probability to detect a flat large flow with $\alpha = 152$ and 303, respectively; and the simulation results suggest that RLFD can detect a large flow with $\alpha = 75$ and 150 with probability around 0.25 and 1.0, respectively. Again, EARDET can only guarantee to detect $\alpha \geq \frac{\rho}{\gamma(m+1)} = \frac{n_\gamma}{m+1} = 991$. That is, RLFD outperforms the exact detection algorithm on low-rate large flows.

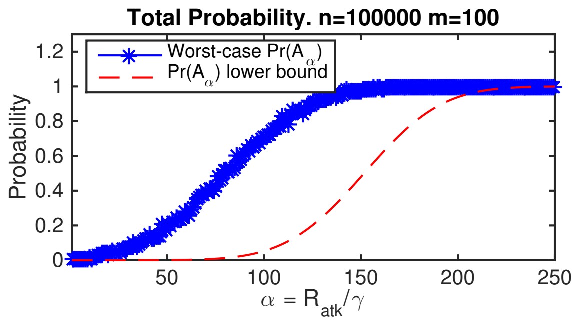


Figure B.3: RLFD total detection probability; $n_\gamma = n = 10^5$, $m = 100$.

Figure B.3 shows an example of simulated worst-case total detection probability $Pr(A_\alpha)$ and its theoretical lower bound (Theorem 11), when $n = n_\gamma = 10^5$ and $m = 100$. The lower bound holds for the most of α, except some very small ones whose $Pr(A_\alpha)$ is close to 0.

B.1.5 Numeric Analysis for Twin-RLFD Theoretical Overuse Damage

Figures B.2(a) to B.2(e) show the expected overuse damage $E(D_{over})$ calculated in the worst case and its upper bound from Theorem 12, in a 40 Gbps link with threshold rate $\gamma = 400$ Kbps ($n_\gamma = 10^5$ full-use legitimate flows at most). The Twin-RLFD has $m = 50$ counters for each RLFD, $d = 4$ levels, and detection cycle $T_c^{(1)} = 0.1$ sec and $T_c^{(2)} = 7.92$ sec for two RLFDs, respectively. Damages by large flows with various duty cycles θ and burst periods T_b are shown. Flows with an average rate R_{atk} higher than $\theta\gamma_h$ (black dash line) will be detected instantly by EARDET (with 100 counters) with nearly zero damage.

The Twin-RLFD has $d = 4$ so that the number of virtual counters in the RLFD bottom level ($m^d = 50^4 = 6.25 \times 10^6$) is larger than the number of flows. Therefore, we the flows selected to the bottom level is fewer than the counters, and RLFD can track each flow individually in the bottom level.

We set $T_c^{(1)} = T_c = 0.1$ sec around $\frac{\beta}{\gamma} = \frac{2\times1514}{400Kbps} = 0.06$ sec (β is usually a few times of maximum packet size 1514 Bytes, so that bursty flows are easier to catch). If $T_c \ll \beta/\gamma$, it is hard for a large flow to reach burst threshold; if $T_c \gg \beta/\gamma$, the detection delay is too long, resulting in excessive damage.

For Twin-RLFD's second RLFD, $T_c^{(2)} = \frac{2d\gamma_h}{\alpha\gamma}T_c^{(1)} = 7.92$ sec (according to Theorem 12). Therefore, Twin-RLFD can guarantee detection for the worst-case bursty flows ($\theta T_b < 2T_c^{(1)}$) at rate $R_{atk} \geq \alpha\gamma = 100\gamma$. We can guarantee detection of lower rate flows with worst-case burstiness by increasing $T_c^{(2)}$; however, increased $T_c^{(2)}$ increases the damage caused by worst-case bursty flows. We say bursty flows with $\theta T_b < 2T_c^{(1)}$ are the worst-case bursty flows, because such flows are unlikely showing up in every level of the RLFD with cycle ($T_c^{(1)}$), so that we have to use the RLFD with longer detection cycle ($T_c^{(2)}$) to catch those flows, which requires longer delay, thus higher damage. Furthermore, such worst-case flows can inflict more damage by increasing θ (thus the average rate), but remain undetectable by EARDet. As discussed in Section 4.4.6, we can use choose different $T_c^{(2)}$ randomly in different cycles to prevent attackers from deterministically maximizing damage.

A hybrid scheme consisting of EARDet and Twin-RLFD can limit the worst-case damage caused by flat flows ($\theta = 1$) and bursty flows ($\theta < 1$). Specifically, for flows with an average rate larger than 30γ (i.e. 12 Mbps), the damage is as low as tens of MBytes (less than ten MBytes for flat flows). We admit that Twin-RLFD cannot limit the damage for flows at extremely low rate ($\ll 30\gamma$) as effectively as for other flows, however other existing schemes cannot neither, because of the limited memory. For flows at high rates, although the Twin-RLFD detects them with almost 100% probability in one detect cycle, it requires at least one cycle to finish detection, hence the damage increases linearly with the flow rate.

Twin-RLFD and EARDet complement each other. Twin-RLFD can detect flows with an average rate lower than $\theta\gamma_h$ but it incompetent at detecting high-rate flows, whereas EARDet is the opposite.

B.2 Proof Sketches

B.2.1 Proof Sketch for RLFD Single-level Detection Probability

Proof sketch of Theorem 8. In the analysis, we treat hashing flows into counters as uniformly assigning n legitimate flows into counters and pick a counter for the large flow f_{atk} at random. We denote the random variable of the maximum number of legitimate flows assigned to a counter as Y and the random variable of the number of legitimate flows in the counter of the large flow f_{atk} as X.

Because the RLFD pick the counter with the largest value for the next level, thus as long as the value of the large-flow counter $(R_{atk} + X \cdot \gamma)T$ is higher than the value of the maximum-value legitimate counter $Y \cdot \gamma T$, the large-flow counter will be picked, where T is the time length of the level. Then we get

$$\begin{aligned}\mathsf{P}_{worst}(m,n,\alpha) =& Pr(R_{atk} + X \cdot \gamma - Y \cdot \gamma > 0)\\ =& Pr(Y - X < \alpha)\\ =& \sum_{y} Pr(y - X < \alpha | Y = y) \cdot Pr(Y = y)\end{aligned} \tag{B.6}$$

As we discussed in Section B.1.3, the distributions of X and Y are the same as those of the X_b and Y_b in a single-weight balls-into-bins game with n balls and m bins, where the X_b is the random variable of the number of balls in a randomly picked bin, and the Y_b is the random variables of the maximum number of balls in a counter. Thus, we can apply Theorem 18 (by Raab and Steger [60]) to calculate the y_{max}, the upper bound of Y at high probability.

Theorem 18 ***Raab and Steger's Theorem 1.*** *Let Y be the random variable that counts the maximum number of balls in any bin, if we throw n balls independently and uniformly at random into m bins. Then $Pr(Y > y_{max}) = o(1)$, if $y_{max} = \frac{n}{m} + \lambda\sqrt{2\frac{n}{m}\log n}$, $\lambda > 1$, $m \log m \ll n \leq m \cdot ploylog(m)$, and n is very large. When $n \to \infty$, $o(1) \to 0$.*

We think it is a good approximation to our large-flow problem. Because the number of legitimate flows n in a backbone link is more than a million, while the number of counters m is quite limited (e.g. one thousand counters in L1 cache, according to Section 4.4.2), thus we say $m \log m \ll n \leq m \cdot ploylog(m)$[1] and n is very large. We derive the approximate lower bound of $\mathsf{P}_{worst}(m, n, \alpha)$ as follows:

$$\begin{aligned}
\mathsf{P}_{worst}(m, n, \alpha) &= \sum_{y} Pr(y - X < \alpha | Y = y) \cdot Pr(Y = y) \\
&= \sum_{y \leq y_{max}} Pr(X > y - \alpha, Y = y) + \sum_{y > y_{max}} Pr(X > y - \alpha, Y = y) \\
&\geq \sum_{y \leq y_{max}} Pr(X > y_{max} - \alpha, Y = y) + \sum_{y > y_{max}} Pr(X > y - \alpha, Y = y) \\
&= Pr(X > y_{max} - \alpha) \cdot Pr(Y \leq y_{max}) + \sum_{y > y_{max}} Pr(X > y - \alpha, Y = y)
\end{aligned} \tag{B.7}$$

We prove that the second part is $o(1)$ as follows:

$$\begin{aligned}
\sum_{y > y_{max}} Pr(X > y - \alpha, Y = y) &\leq \sum_{y > y_{max}} Pr(X > y_{max} - \alpha, Y = y) \\
&= Pr(X > y_{max - \alpha}) \cdot Pr(Y > y_{max - \alpha}) \\
&= Pr(X > y_{max - \alpha}) \cdot o(1) = o(1)
\end{aligned} \tag{B.8}$$

According to Equation B.7 and B.8, we get

$$\begin{aligned}
\mathsf{P}_{worst}(m, n, \alpha) &\geq Pr(X > y_{max} - \alpha) \cdot Pr(Y \leq y_{max}) + o(1) \\
&= Pr(X > y_{max} - \alpha) \cdot [1 - Pr(Y > y_{max})] + o(1) \\
&= Pr(X > y_{max} - \alpha) \cdot (1 - o(1)) + o(1) \\
&= Pr(X > y_{max} - \alpha) - o(1) \\
&\approx Pr(X > y_{max} - \alpha)
\end{aligned} \tag{B.9}$$

Therefore, when n is large, we approximately have $\mathsf{P}_{worst}(m, n, \alpha) \geq Pr(X > y_{max} - \alpha)$.

We let $\eta = \lceil y_{max} - \alpha \rceil$, and use random variable M_k to denote the number of bins exactly contain k balls. We calculate $Pr(X > y_{max} - \alpha)$ as follows:

[1] Raab et al. also provides a similar y_{max} for $m(\log m)^3 \ll n$, but it is enough to only discuss one of them for an approximate result.

$$
\begin{aligned}
Pr(X > y_{max} - \alpha) = Pr(X > \eta) &= \sum_{k\geq\eta} Pr(X = k) \\
&= \sum_{k\geq\eta} \sum_{0\leq m_k\leq m} Pr(X = k|M_k = m_k) \cdot Pr(M_k = m_k) \\
&= \sum_{k\geq\eta} \sum_{0\leq m_k\leq m} \frac{m_k}{m} Pr(M_k = m_k) \\
&= \sum_{k\geq\eta} \frac{1}{m} \sum_{0\leq m_k\leq m} m_k Pr(M_k = m_k) = \sum_{k\geq\eta} \frac{E(M_k)}{m} \\
&= \sum_{\eta\leq k\leq n} \frac{1}{m} \cdot m \frac{\binom{n}{k}(m-1)^{n-k}}{m^n} \\
&= \sum_{\eta\leq k\leq n} \binom{n}{k} \left(1 - \frac{1}{m}\right)^{n-k} \left(\frac{1}{m}\right)^k
\end{aligned}
\tag{B.10}
$$

The above result requires to calculate the sum of the last $m - \eta + 1$ items from the binomial distribution $B(n, \frac{1}{m})$. As we know, there are no simple, closed forms for Equation B.10. According to the law of rare events [61], binomial distribution $B(n,p)$ is approximate to Poisson distribution $Pois(np)$, when n is large and p is small. According to Equation B.15, the detection probability $Pr(A_\alpha)$ is mainly related to non-bottom levels in which the number of flows $n$is large $(n > m)$, and $p = \frac{1}{m}$ is small because m is around hundreds to thousands, we approximately treat $B(n, \frac{1}{m})$ as the Poisson distribution $Pois(\frac{n}{m})$, then we have

$$
\binom{n}{k} \left(1 - \frac{1}{m}\right)^{n-k} \left(\frac{1}{m}\right)^k \approx \frac{e^{-\frac{n}{m}} (\frac{n}{m})^k}{k!}
\tag{B.11}
$$

which is the probability of the item happens k times in the Poisson distribution. Then, the Equation B.10 turns to

$$
\begin{aligned}
Pr(X > y_{max} - \alpha) &= \sum_{\eta\leq k\leq n} \binom{n}{k} \left(1 - \frac{1}{m}\right)^{n-k} \left(\frac{1}{m}\right)^k \\
&= \sum_{\eta\leq k\leq n} \frac{e^{-\frac{n}{m}} (\frac{n}{m})^k}{k!} = 1 - Q(\eta - 1, \frac{n}{m})
\end{aligned}
\tag{B.12}
$$

where $Q(K, \frac{n}{m})$ is the cumulative distribution function (CDF) of the Poisson distribution $Pois(\frac{n}{m})$, i.e. sum of probabilities for $0 \leq k \leq K$. As the Theorem 18 holds when $\lambda > 1$, thus we choose $\lambda \to 1^+$, thus $y_{max} = \frac{n}{m} + \sqrt{2\frac{n}{m}\log n}$. Because we focus on how does the probability lower bound change along with the m and n, the λ does not matter much here. Therefore, we proved that the $1 - Q(K, \frac{n}{m})$ is an approximate lower bound for $\mathsf{P}_{worst}(m, n, \alpha)$, where $K = \eta - 1 = \left\lfloor \frac{n}{m} + \sqrt{2\frac{n}{m}\log n} - \alpha \right\rfloor$. ■

Proof sketch of Corollary 9. According to Theorem 8, $\mathsf{P}_{worst}(m, n, \alpha_{0.5}) > 1 - Q(K, \frac{n}{m})$ approximately, where $K = \left\lfloor \frac{n}{m} + \sqrt{2\frac{n}{m}\log n} - \alpha_{0.5} \right\rfloor$. As the median[2] ν of the Poisson distribution $Pois(\frac{n}{m})$ is bounded by $\frac{n}{m} - \log 2 \le \nu < \frac{n}{m} + \frac{1}{3}$ [62]. Thus, $\nu \approx \frac{n}{m}$, then

$$K \approx \frac{n}{m} \Rightarrow \alpha_{0.5} \approx \sqrt{2\frac{n}{m}\log n} \tag{B.13}$$

Therefore the Corollary 9 is proved.■

Proof sketch of Corollary 10. According to Pearson's skewness coefficients [63], the symmetry of a distribution is measured by its skewness. The probability distribution is approximately symmetrical to its mean when the skewness is small. According to [64], the skewness of Poisson distribution $Pois(\frac{n}{m})$ is $\left(\frac{n}{m}\right)^{-0.5}$. Thus when $n \gg m \log m$ the $Pois(\frac{n}{m})$ is approximately symmetrical to its mean $\frac{n}{m}$.

Because when $\alpha = 1$ the actual $\mathsf{P}_{worst}(m, n, \alpha)$ should be $\frac{1}{m} \approx 0$ (because the large flow rate is the same as the legitimate flow rate, thus the detection equals to randomly picking one from m counters), thus the approximate lower bound $1 - Q(K_{\alpha=1}, \frac{n}{m}) \approx 0$, where $K_{\alpha=1} = \left\lfloor \frac{n}{m} + \sqrt{2\frac{n}{m}\log n} - 1 \right\rfloor$. As $Pois(\frac{n}{m})$ is symmetrical to $K_s = \frac{n}{m}$, when $K = K_s + (K_s - K_{\alpha=1}) \approx \frac{n}{m} - \sqrt{2\frac{n}{m}\log n}$, the $1 - Q(K, \frac{n}{m}) \approx 1$, in which $\alpha \approx 2\alpha_{0.5}$ (according to Corollary 9). Thus, $\alpha_{1.0} = 2\alpha_{0.5}$ has been proved.■

B.2.2 Proof Sketch for RLFD Total Detection Probability

Proof sketch of Theorem 11. For the detection level k, we use $A_{k,\alpha}$ to denote the event that the counter containing the large flow f_{atk} with average rate of $R_{atk} = \alpha\gamma$ in the level k is selected for the next level, where γ is the threshold rate, $\alpha > 1$. Then the total probability for RLFD to catch the large flow f_{atk} in one detection cycle is

[2]The K such that the CDF $Q(K, \frac{n}{m}) = 0.5$

$$
\begin{aligned}
Pr(A_\alpha) =& Pr(A_{1,\alpha}, A_{2,\alpha}, A_{3,\alpha}, ..., A_{d,\alpha}) \\
=& Pr(A_{1,\alpha}) \cdot Pr(A_{2,\alpha}|A_{1,\alpha}) \cdot Pr(A_{3,\alpha}|A_{2,\alpha}, A_{1,\alpha}) \cdot \\
& ...Pr(A_{d,\alpha}|A_{d-1,\alpha}, ..., A_{1,\alpha}) \\
=& Pr(A_{1,\alpha}) \cdot Pr(A_{2,\alpha}|A_{1,\alpha}) \cdot Pr(A_{3,\alpha}|A_{2,\alpha}) \cdot \\
& ...Pr(A_{d,\alpha}|A_{d-1,\alpha})
\end{aligned}
\tag{B.14}
$$

As we described in Section 4.4.2, we use the Cuckoo hashing in the bottom level d to randomly assign flows into counters. Because we set enough levels to make the input flows in the bottom level less than the counters, the $Pr(A_{d,\alpha}|A_{d-1,\alpha}) \approx 1$. For the levels $k < d$ with $n^{(k)}$ legitimate flows, according to Theorem 8 the $Pr(A_{k,\alpha}|A_{k-1,\alpha}) \geq \mathsf{P}_{worst}(m, n^{(k)}, \alpha)$. Considering the maximum number of full-use legitimate flows in a link is $n_\gamma = \rho/\gamma$,

- When $n < n_\gamma$, $Pr(A_{k,\alpha}|A_{k-1,\alpha}) \geq \mathsf{P}_{worst}(m, n^{(k)}, \alpha)$
- When $n \geq n_\gamma$, $Pr(A_{k,\alpha}|A_{k-1,\alpha}) \geq \mathsf{P}_{worst}(m, n_\gamma, \alpha)$

Therefore,

$$
Pr(A_\alpha) \geq \prod_{k=1}^{d-1} \mathsf{P}_{worst}(m, \mathsf{min}(n_\gamma, n^{(k)}), \alpha) \tag{B.15}
$$

where we approximately let $n^{(k)} = n/m^{k-1}$, which is the average value of $n^{(k)}$ over repeated detection. n is the number of legitimate flows in the link.

According to Equation B.15 and the fact that $\alpha_{1.0}$ decreases fast as the $n^{(k)}$ decreases by the factor of m, $\mathsf{P}_{worst}(m, n^{(k)}, \alpha)$ for $n^{(k)} < n_\gamma$ does not affect the product much for the most of α values. Therefore, we can approximate $Pr_{worst}(A_\alpha)$ as follows:

$$
Pr(A_\alpha) \geq \begin{cases} \displaystyle\prod_{\{k|n^{(k)} \geq n_\gamma\}} \mathsf{P}_{worst}(m, n_\gamma, \alpha), & when\ n \geq n_\gamma \\ \mathsf{P}_{worst}(m, n, \alpha) & , when\ n < n_\gamma \end{cases} \tag{B.16}
$$

where size of $\{k|n^{(k)} \geq n_\gamma\}$ is $\lfloor \log_m(n/n_\gamma) \rfloor + 1$, because $n^{(k)} = n^{(k-1)}/m$.

According to Theorem 8, approximately $\mathsf{P}_{worst}(m, n, \alpha) \geq 1 - Q(K, \frac{n}{m})$ where $K = \lfloor \frac{n}{m} + \sqrt{2\frac{n}{m}\log n} - \alpha \rfloor$. Thus we can derive Theorem 11 from Equation B.16.■

B.2.3 Proof Sketch for Twin-RLFD Theoretical Overuse Damage

The upper bound of the expected overuse damage can be derived from the average rate of a flat large flow and the expected detection delay: $E(D_{over}) \leq E(T_{delay}) \cdot R_{atk}$, because attack flows cannot cause more overuse damage than the amount of traffic over-sent $E(T_{delay}) \cdot R_{atk}$. For a bursty flow with duty cycle θ and burst period T_b, a RLFD can also treat it as a flat flow at the time of each burst interval θT_b. Thus, we can still use the detection probability for flat flows to calculate the damage for bursty flows.

Lemma 19 and proof sketch.

Lemma 19 *A RLFD with detection cycle T_c can detect bursty flows with $\theta T_b \geq 2T_c$ with the expected overuse damage:*

$$E(D_{over}) \leq \begin{cases} T_c\gamma\alpha/\theta\Big(1 - Q(K_\gamma, \frac{n_\gamma}{m})\Big)^{\lfloor \log_m(n/n_\gamma)\rfloor+1} & \text{, when } n \geq n_\gamma \\ T_c\gamma\alpha/\theta\Big(1 - Q(K, \frac{n}{m})\Big) & \text{, when } n < n_\gamma \end{cases} \tag{B.17}$$

where $K_\gamma = \left\lfloor \frac{n_\gamma}{m} + \sqrt{2\frac{n_\gamma}{m}\log n_\gamma} - \frac{\alpha}{\theta} \right\rfloor$, $K = \left\lfloor \frac{n}{m} + \sqrt{2\frac{n}{m}\log n} - \frac{\alpha}{\theta} \right\rfloor$, *and* $Q(x, \lambda)$ *is the CDF of the Poisson distribution* $Pois(\lambda)$.

Proof sketch: Because $\theta T_b \geq 2T_c$, thus for each burst period T_b there are must be at least $\left\lfloor \frac{\theta T_b}{T_c} - 1 \right\rfloor$ detection cycles, in which RLFD can see the attack traffic in all levels. When the RLFD observes the bursty flow, the only difference from the detection over flat flow is that, the traffic rate at that moment is $\frac{\alpha}{\theta}\gamma$, instead of $\alpha\gamma$ in the case of flat flows. Thus, the probability $Pr(A_\alpha)$ to detect such bursty flow in one detection cycle is calculated as the one for flat flow detection in Theorem 11, by replacing the α with the $\frac{\alpha}{\theta}$. The expected detection delay $E(T_{delay})$ is derived as follows:

$$E(T_{delay}) \leq \frac{1}{Pr(A_\alpha)} \frac{T_b}{\left\lfloor \frac{\theta T_b}{T_c} - 1 \right\rfloor} \approx \frac{T_c}{\theta Pr(A_\alpha)} \tag{B.18}$$

Then we get the over-sent attack traffic in the input link is $E(T_{delay}) \cdot R_{atk}$, and the overused bandwidth by attack traffic is less than or equal to that, because the some attack packets may also be dropped during congestion.

Thus we get the expected overuse damage $E(D_{over})$:

$$E(D_{over}) \leq E(T_{delay}) \cdot R_{atk} \leq T_c\gamma\alpha/\theta Pr(A_\alpha) \tag{B.19}$$

Thus, according to Theorem 11, we get the upper bound of the overuse damage in the Lemma 19. The proof also holds when $\theta = 1$, which is for the case of flat flows. ■

Proof sketch of Theorem 12. The overuse damage in the case of $\theta T_b \geq 2T_c^{(1)}$ are from Lemma 19. When $\theta T_b < 2T_c^{(1)}$ and $R_{atk} < \theta\gamma_h$, we prove the damage as follows:

$$T_b < \frac{2T_c^{(1)}}{\theta} < \frac{2T_c^{(1)}}{R_{atk}}\gamma_h = \frac{2T_c^{(1)}}{\alpha\gamma}\gamma_h = \frac{T_c^{(2)}}{d} \tag{B.20}$$

Thus the T_b is less than a detection level period of the $EFD^{(2)}$, which means the bursty flow is like a flat flow to $EFD^{(2)}$. Therefore, we use the overuse damage upper bound in Lemma 19, when $\theta = 1$, $T_c = T_c^{(2)}$, and we get

$$E(D_{over}) \leq \begin{cases} T_c^{(2)}\gamma\alpha/\left(1 - Q(K_\gamma, \frac{n_\gamma}{m})\right)^{\lfloor \log_m(n/n_\gamma)\rfloor+1} & , \textit{when } n \geq n_\gamma \\ T_c^{(2)}\gamma\alpha/\left(1 - Q(K, \frac{n}{m}))\right) & , \textit{when } n < n_\gamma \end{cases} \tag{B.21}$$

where $K_\gamma = \left\lfloor \frac{n_\gamma}{m} + \sqrt{2\frac{n_\gamma}{m}\log n_\gamma} - \alpha \right\rfloor$, $K = \left\lfloor \frac{n}{m} + \sqrt{2\frac{n}{m}\log n} - \alpha \right\rfloor$. By replacing $T_c^{(2)}$ with $\frac{2\gamma_h}{\alpha\gamma}T_c^{(1)}$, we proved the Theorem 4.3.■

B.3 Additional Table

Table B.1: Settings of RLFD and CLEF.

m	20	40	70	100	150	200	400
T_ℓ^*	.242	.242	.242	.242	.242	.242	.242
Single RLFD							
d	4	3	3	3	3	3	2
T_c^*	.968	.726	.726	.726	.726	.726	.484
Twin-RLFD (in CLEF)							
d	7	5	4	4	4	3	3
$T_c^{(1)*}$	1.69	1.21	.968	.968	.968	.726	.726
$T_c^{(2)*}$	168.6	63.75	31.92	26.56	21.96	10.68	7.59

* Time unit is second.

REFERENCES

[1] H. Wu, H.-C. Hsiao, and Y.-C. Hu, “Efficient large flow detection over arbitrary windows: An algorithm exact outside an ambiguity region,” in *Proceedings of the 2014 Conference on Internet Measurement Conference.* ACM, 2014, pp. 209–222.

[2] 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)*, vol. 21, no. 3, pp. 270–313, 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=859719

[3] S. Shenker, C. Partridge, and R. Guerin, “Specification of guaranteed quality of service,” RFC 2212 (Proposed Standard), Sep. 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2212.txt

[4] R. Braden, D. Clark, and S. Shenker, “Integrated services in the internet architecture: An overview,” RFC 1633 (Informational), June 1994. [Online]. Available: http://www.ietf.org/rfc/rfc1633.txt

[5] C. Basescu, R. M. Reischuk, P. Szalachowski, A. Perrig, Y. Zhang, H.-C. Hsiao, A. Kubota, and J. Urakawa, “SIBRA: Scalable internet bandwidth reservation architecture,” in *Proceedings of Network and Distributed System Security Symposium (NDSS)*, Feb. 2016.

[6] Z. Liu, H. Jin, Y.-C. Hu, and M. Bailey, “MiddlePolice: Toward enforcing destination-defined policies in the middle of the internet,” in *Proceedings of ACM CCS*, Oct. 2016.

[7] S. B. Lee, M. S. Kang, and V. D. Gligor, “CoDef: Collaborative defense against large-scale link-flooding attacks,” in *Proceedings of CoNext*, 2013.

[8] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker, “Accountable internet protocol (AIP),” in *Proceedings of ACM SIGCOMM*, 2008. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1402958.1402997

[9] J. Naous, M. Walfish, A. Nicolosi, D. Mazières, M. Miller, and A. Seehra, "Verifying and enforcing network paths with ICING," in *Proceedings of ACM CoNEXT*, 2011.

[10] A. Li, X. Liu, and X. Yang, "Bootstrapping accountability in the Internet we have," in *Proceedings of USENIX/ACM NSDI*, Mar. 2011.

[11] T. H.-J. Kim, C. Basescu, L. Jia, S. B. Lee, Y.-C. Hu, and A. Perrig, "Lightweight source authentication and path validation," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 271–282.

[12] X. Liu, A. Li, X. Yang, and D. Wetherall, "Passport: Secure and adoptable source authentication," in *Proceedings of USENIX/ACM NSDI*, 2008. [Online]. Available: http://www.usenix.org/event/nsdi08/tech/full_papers/liu_xin/liu_xin_html/

[13] "Resource Public Key Infrastructure (RPKI)," 2015. [Online]. Available: https://www.arin.net/resources/rpki/

[14] T. Anderson, K. Birman, R. Broberg, M. Caesar, D. Comer, C. Cotton, M. J. Freedman, A. Haeberlen, Z. G. Ives, A. Krishnamurthy et al., "The nebula future internet architecture," in *The Future Internet Assembly.* Springer, 2013, pp. 16–26.

[15] X. Zhang, H.-C. Hsiao, G. Hasker, H. Chan, A. Perrig, and D. G. Andersen, "SCION: Scalability, control, and isolation on next-generation networks," in *IEEE Symposium on Security and Privacy*, 2011, pp. 212–227.

[16] D. Han, A. Anand, F. Dogar, B. Li, H. Lim, M. Machado, A. Mukundan, W. Wu, A. Akella, D. G. Andersen, J. W. Byers, S. Seshan, and P. Steenkiste, "XIA: Efficient support for evolvable internetworking," in *Proc. 9th USENIX NSDI*, San Jose, CA, Apr. 2012.

[17] C. Estan, "Internet traffic measurement: What's going on in my network?" Ph.D. dissertation, University of California, San Diego, 2003.

[18] J. Misra and D. Gries, "Finding repeated elements," *Science of Computer Programming*, vol. 2, no. 2, pp. 143–152, 1982.

[19] E. D. Demaine, A. López-Ortiz, and J. I. Munro, "Frequency estimation of internet packet streams with limited space," in *Proceedings of ESA*, 2002. [Online]. Available: http://www.springerlink.com/index/0MJ1EXMY9L9MCQAD.pdf

[20] 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*, vol. 28, no. 1, pp. 51–55, 2003. [Online]. Available: http://portal.acm.org/citation.cfm?doid=762471.762473

[21] G. Manku and R. Motwani, "Approximate frequency counts over data streams," in *Proceedings of VLDB*, 2002. [Online]. Available: http://dl.acm.org/citation.cfm?id=1287400

[22] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *Proceedings of ICDT*, 2005. [Online]. Available: http://www.springerlink.com/index/TP581QC7AX7EQGT3.pdf

[23] M. Fang and N. Shivakumar, "Computing iceberg queries efficiently," in *Proceedings of VLDB*, 1999. [Online]. Available: http://ilpubs.stanford.edu:8090/423/

[24] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *Journal of Algorithms*, vol. 55, no. 1, pp. 58–75, 2005. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0196677403001913

[25] 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. [Online]. Available: http://dl.acm.org/citation.cfm?id=948227

[26] A. Arasu and G. S. Manku, "Approximate counts and quantiles over sliding windows," in *Proceedings of ACM PODS*, 2004. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1055558.1055598

[27] 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. [Online]. Available: http://dl.acm.org/citation.cfm?id=1142393

[28] G. Cormode and M. Hadjieleftheriou, "Finding frequent items in data streams," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1530–1541, 2008. [Online]. Available: http://www.springerlink.com/index/T17NHD9HWWRY909P.pdf

[29] B. Boyer and J. Moore, "A fast majority vote algorithm," Technical Report ICSCA-CMP-32, Institute for Computer Science, University of Texas, Tech. Rep., 1981.

[30] M. Fischer and S. Salzberg, "Finding a majority among n votes: Solution to problem 81-5," *Journal of Algorithms - JAL*, vol. 3, no. 4, pp. 362–380, 1982.

[31] B. Claise, "Cisco systems NetFlow services export version 9," RFC 3954 (Informational), Oct. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3954.txt

[32] A. Metwally, D. Agrawal, and A. El Abbadi, "Efficient computation of frequent and top-k elements in data streams," in *International Conference on Database Theory.* Springer, 2005, pp. 398–412.

[33] V. Sivaraman, S. Narayana, O. Rottenstreich, S. Muthukrishnan, and J. Rexford, "Heavy-hitter detection entirely in the data plane," in *Proceedings of the Symposium on SDN Research.* ACM, 2017, pp. 164–176.

[34] D. Tong and V. Prasanna, "High throughput sketch based online heavy hitter detection on FPGA," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 4, pp. 70–75, 2016.

[35] M. Chen, S. Chen, and Z. Cai, "Counter tree: A scalable counter architecture for per-flow traffic measurement," *IEEE/ACM Transactions on Networking*, 2016.

[36] Q. Xiao, S. Chen, M. Chen, and Y. Ling, "Hyper-compact virtual estimators for big network data based on register sharing," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 43, no. 1. ACM, 2015, pp. 417–428.

[37] Z. Liu, A. Manousis, G. Vorsanger, V. Sekar, and V. Braverman, "One sketch to rule them all: Rethinking network flow monitoring with UnivMon," in *ACM SIGCOMM*, 2016.

[38] J. Turner, "New directions in communications (or which way to the information age?)," *IEEE Communications Magazine*, vol. 24, pp. 8–15, 1986.

[39] 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*, vol. 11, no. 1, pp. 33–46, Feb. 2003. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1180544

[40] S. Machiraju, M. Seshadri, and I. Stoica, "A scalable and robust solution for bandwidth allocation," in *Tenth IEEE International Workshop on Quality of Service*, 2002, pp. 148–157.

[41] A. Kuzmanovic and E. Knightly, "Low-rate TCP-targeted denial of service attacks and counter strategies," *IEEE/ACM Transactions on Networking*, vol. 14, no. 4, pp. 683–696, 2006. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1677591

[42] S. A. Crosby and D. S. Wallach, "Denial of service via algorithmic complexity attacks," in *Proceedings of USENIX Security*, 2003. [Online]. Available: http://static.usenix.org/event/sec03/tech/full_papers/home/staff/alex/export/twycross/crosby/crosby_html/

[43] M. Guirguis, A. Bestavros, and I. Matta, "Exploiting the transients of adaptation for RoQ attacks on internet resources," in *Proceedings of IEEE ICNP*, 2004. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1348109

[44] A. Dainotti, A. Pescapè, P. Salvo Rossi, F. Palmieri, and G. Ventre, "Internet traffic modeling by means of hidden Markov models," *Computer Networks (Elsevier)*, vol. 52, pp. 2645–2662, 2008.

[45] A. Dainotti, A. Pescapè, and G. Ventre, "A cascade architecture for DoS attacks detection based on the wavelet transform," *Journal of Computer Security*, vol. 17, no. 6/2009, pp. 945–968, 2009.

[46] "Traces 1 of TCP port 80 traffic traces from Federico II." [Online]. Available: http://traffic.comics.unina.it/Traces/ttraces.php

[47] "The CAIDA UCSD anonymized internet traces 2012 - 1220." [Online]. Available: http://www.caida.org/data/passive/passive_2012_dataset.xml

[48] 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, Tech. Rep., 2014.

[49] A. Kuzmanovic and E. Knightly, "Low-rate TCP-targeted denial of service attacks: The shrew vs. the mice and elephants," in *Proceedings of ACM SIGCOMM*, 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=863966 pp. 75–86.

[50] E. C. Milner and R. Rado, "The pigeon-hole principle for ordinal numbers," *Proceedings of the London Mathematical Society*, vol. 3, no. 1, pp. 750–768, 1965.

[51] R. Pagh and F. F. Rodler, "Cuckoo hashing," in *European Symposium on Algorithms*. Springer, 2001, pp. 121–133.

[52] M. Mitzenmacher, "Some open questions related to cuckoo hashing," in *European Symposium on Algorithms.* Springer, 2009, pp. 1–10.

[53] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* MIT Press Cambridge, 1998, vol. 1, no. 1.

[54] H. B. Demuth, M. H. Beale, O. De Jess, and M. T. Hagan, *Neural Network Design.* Martin Hagan, 2014.

[55] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks." in *Aistats*, vol. 15, no. 106, 2011, p. 275.

[56] P. F. Verhulst, "Recherches mathématiques sur la loi d'accroissement de la population." *Nouveaux mémoires de l'académie royale des sciences et belles-lettres de Bruxelles*, vol. 18, pp. 14–54, 1845.

[57] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[58] P. Y. Simard, D. Steinkraus, J. C. Platt et al., "Best practices for convolutional neural networks applied to visual document analysis." in *ICDAR*, vol. 3. Citeseer, 2003, pp. 958–962.

[59] P. Berenbrink, T. Friedetzky, Z. Hu, and R. Martin, "On weighted balls-into-bins games," *Theoretical Computer Science*, vol. 409, no. 3, pp. 511–520, 2008.

[60] M. Raab and A. Steger, "'Balls into bins' - A simple and tight analysis," in *International Workshop on Randomization and Approximation Techniques in Computer Science.* Springer, 1998, pp. 159–170.

[61] A. C. Cameron and P. K. Trivedi, *Regression Analysis of Count Data.* Cambridge University Press, 2013, vol. 53.

[62] K. P. Choi, "On the medians of gamma distributions and an equation of Ramanujan," *Proceedings of the American Mathematical Society*, vol. 121, no. 1, pp. 245–251, 1994.

[63] E. W. Weisstein, "Pearson's skewness coefficients." From MathWorld–A Wolfram Web Resource, 2017. [Online]. Available: http://mathworld.wolfram.com/PearsonsSkewnessCoefficients.html

[64] E. W. Weisstein, "Poisson distribution." From MathWorld–A Wolfram Web Resource, 2017. [Online]. Available: http://mathworld.wolfram.com/PoissonDistribution.html