EFFICIENT LARGE FLOW DETECTION OVER ARBITRARY WINDOWS: AN EXACT ALGORITHM OUTSIDE AN AMBIGUITY REGION

BY

HAO WU

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Adviser:

Associate Professor Yih-Chun Hu

ABSTRACT

Being able to exactly detect large network flows under an arbitrary time window model is expected in many current and future applications like Denial-of-Service (DoS) flow detection, bandwidth guarantee, etc. However, to the best of our knowledge, there is no existing work that can achieve exact large flow detection without per-flow status. Maintaining per-flow status requires a large amount of expensive line-speed storage, thus it is not practical in real systems. Therefore, we proposed a novel model of an arbitrary time window with exactness outside an ambiguity region, which trades the level of exactness for scalability. Although some existing work also uses some techniques like sampling, multistage filters, etc. to make the system scalable, most of them do not support the arbitrary time window model and they usually introduce a lot of false positives for legitimate flows. Inspired by a frequent item finding algorithm, we proposed Exact-outside-Ambiguity-Region Detector (EARDet), an arbitrary-window-based, efficient, simple, and no-per-flow-status large flow detector, which is exact outside an ambiguity window defined by a high-bandwidth threshold and a low-bandwidth threshold. EARDet is able to catch all large flows violating the high-bandwidth threshold; meanwhile it protects all legitimate flows complying with the low-bandwidth threshold. Because EARDet focuses on flow classification but not flow size estimation, it demonstrates amazing scalability such that we can fit the storage into on-chip Static Random-Access Memory (SRAM) to achieve line-speed detection. To evaluate EARDet, we not only theoretically proved properties of EARDet above, but also evaluated them with real traffic, and the result perfectly supports our analysis.

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 of this thesis. Also, I want to give special thanks to Professor Hsu-Chun Hsiao, who has worked together with me on this project for almost one year. She gave me many valuable suggestions on how to work like a real researcher. Finally, many thanks to my family who supported and helped me finish my Master's Degree.

TABLE OF CONTENTS

LIST OF ABBREVIATIONS

DoS	Denial of Service
EARDet	Exact-Outside-Ambiguity-Region Detector
MG	Misra-Gries
FMF	Fixed-window-based Multistage Filter
AMF	Arbitrary-window-based Multistage Filter
FN	False Negative for large flows
FP	False Positive for small flows
No-FN$_\ell$	No False Negative for large flows
No-FP$_s$	No False Positive for small flows

LIST OF SYMBOLS

f	Flow in network link
$\mathcal{F}$	A set of flows
t	Timestamp
ρ	Rate of link capacity
α	Maximum size of packet
TH_ℓ	Low-bandwidth threshold of ambiguity region
TH_h	High-bandwidth threshold of ambiguity region
γ_ℓ	Rate limit for low-bandwidth threshold (TH_ℓ)
β_ℓ	Burst limit for low-bandwidth threshold (TH_ℓ)
γ_h	Rate limit for high-bandwidth threshold (TH_h)
β_h	Burst limit for high-bandwidth threshold (TH_h)
n	Number of counters in EARDet
β_{TH}	Threshold of counter in EARDet ($> \beta_\ell$)
β_Δ	Equals $\beta_{TH} - \beta_\ell$
$R(t_1, t_2)$	Average flow rate over time interval $[t_1, t_2)$
t_{incb}	Incubation period of large flows
t_{upincb}	Upper bound of incubation period (t_{incb}) for all large flows
R_{NFN}	Lower bound of large flow rate for no false negative over large flows (No-FN$_\ell$)
R_{NFP}	Upper bound of small flow rate for no false positive over small flows (No-FP$_s$)

CHAPTER 1

INTRODUCTION

Accurately identifying large flows[1] which occupy high bandwidth or consume a large volume of link bandwidth over some short time window is significantly important for various applications in networking and security, such as Denial of Service (DoS) defense and bandwidth guarantee. However, no existing scalable approach both considers large flows over the *arbitrary window model* and is exact in classifying flows in one pass. The arbitrary window model considers large flows over every range of time which begins at each time point in the past and ends at the current time, so that it can even detect burst flows which could evade the detection by a system over fixed-time window model.

Hence, we proposed a newly designed model with exactness outside a small *ambiguity region*, which is determined by two adjustable bandwidth thresholds. This novel model classifies small flows, median flows, and large flows in a network link. The small flow is a flow whose size is below the low-bandwidth threshold, the large flow is a flow whose size exceeds the high-bandwidth threshold, and the result of flows are defined as median flows. Our model with exactness outside the ambiguity region is able to guarantee that all large flows are going to be caught anyway and no small flows are going to be caught by mistake. This model makes sense in large flow detection, because we can limit the severe damage resulting from large flows and protect the user experience of legitimate users (i.e. small flows). Although existing work [2], [3] discussed some similar ideas, only probabilistic bounds outside

[1]This is also called as elephant flows in the literature.

a region are discussed. With the model of ambiguity region between two bandwidth threshold, we are able to trade the level of exactness for scalability, and thus we are able to run the large flow detector in on-chip memory and achieve link-speed flow check. With this ambiguity-region model and the arbitrary window model, we are able to improve a lot of applications, like DoS flow detection, bandwidth guarantees, etc.

As far as we known, there is no algorithm that is able to provide an arbitrary window check with exactness outside an ambiguity region. Those prior approaches which only monitor average network throughput are not able to detect bursty flows. For instance, by deliberately spreading burst across two neighbor time intervals, a large bursty flow is able to pass the periodic large flow detector which resets the state periodically. Moreover, our model is deterministic. Although adding randomness into large flow detection (e.g. the time interval is random) can to some extend solve the problem, such randomized algorithms are not able to provide strong deterministic guarantees and stable performance.

Therefore, we proposed a novel arbitrary-window-based detector, EARDet (**E**xact-outside-**A**mbiguity-**R**egion **Det**ector), which is exact outside an *ambiguity region*. Built on the frequent-item-finding algorithm proposed by Misra and Gries [4], EARDet is an efficient, simple, and no-per-flow-state one-pass streaming algorithm that only maintains a small array of counters. We increase or decrease the counters when a new packet comes into the detector, and once a flow's counter exceeds the counter threshold, we identify it as a large flow. It classifies the flows by a high-bandwidth threshold and a low-bandwidth threshold into large, medium and small flows, and the ambiguity region contains flows between these two thresholds (i.e. medium flows). While keeping no miss detection on large flows, EARDet can project every small flow from false detection with no assumption on the input traffic and attack pattern.

Moreover, after optimization, we surprisingly found EARDet is extremely scalable in that it only requires a very small amount of memory while keeping the strong guarantee over large flow detection. With the requirement of only such a small size of memory, we can fit it into some on-chip SRAM (Static Random-Access Memory) and let it process packets with line speed. Of course, the reason of such efficiency is that EARDet only classifies flows into small flow, median flow, and large flow, but does not estimate the size

of the flow.

To evaluate and demonstrate those strong properties of EARDet, in the rest of this thesis, we are going to not only present detailed theoretical analysis for EARDet, but also test EARDet with real traffic in our simulation platform.

CHAPTER 2

BACKGROUND KNOWLEDGE

2.1 System Model

Flow identifiers In general, the information in the packet header acts as the flow identifiers (e.g. IP address, port number, etc.) to group packets into different flows. In this thesis, we are focusing on research on a generic large-flow detection solution which can be applied in cases without assumption on flow identifiers. Similarly to prior traffic monitor work, we assume the flow identifiers (or flow IDs) are unforgeable, which can be achieved by multiple existing approaches, e.g. source authentication [5] and ingress filtering [6].

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 $\mathsf{vol}(f, t_1, t_2) \triangleq \sum_{x \in \mathcal{X}, \mathsf{fid}(x)=f, t_1 \leq \mathsf{time}(x) < t_2} \mathsf{size}(x)$.

2.2 Large and Small 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 $[t_2, t_1)$.

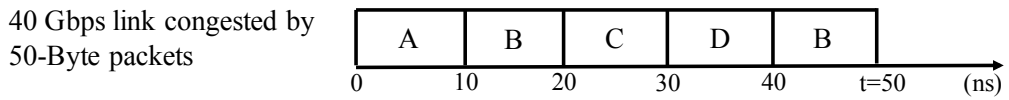
For a flow f, if there exists a time window $[t_1, t_2)$ over which the volume of flow $\mathsf{vol}(f, t_1, t_2)$ exceeds a threshold function $\mathsf{TH}(t_2 - t_1)$, then the flow f is classified as large flow; otherwise, the flow f is considered as small flow. Namely, (i) when $\mathsf{vol}(f, t_1,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

2.3 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 [3]. However achieving the arbitrary window model in practice is challenging, therefore, people usually use some approximate approaches to roughly identify large flows. Thus we



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].

have two more typical time window models: the landmark window model [2, 4, 7–12] and the sliding window model [13–15].

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 windows (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 timescales 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 in Figure 2.1.[1]

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

CHAPTER 3

RELATED WORK

In this chapter, we review prior works by the techniques they use in the algorithm as presented in the survey by Cormode and Hadjieleftheriou [16]: counter-based technique, sketch-based technique, and sampling-based technique. We pick and summarize some typical algorithms in each category and discuss their drawbacks.

3.1 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 [4] takes a stream of items as input and find the set of frequent items exactly. The MG algorithm extends the majority algorithm [17, 18], which only considers finding the majority vote.

3.2 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[1] (called FMF in this paper). The FMF has multiple

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

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[2] spanning two measurement intervals. To address this, Estan [3] 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.

3.3 Sampling-Based Algorithm

By sampling the packets in the link, the overhead of algorithm can be reduced effectively. The Sampled NetFlow [19] 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.

3.4 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 small 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

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

process. The FMF and AMF are the multistage filters algorithms, whose counter could be shared by both large flows and small flows. With some probability, the hash function could map a large flow and small flow to exactly the same counter in each stage. This problem is the nature of multistage filters and cannot be avoided.

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.

Since the MG algorithm, sample and hold algorithm, and FMF are based on the landmark window model, they cannot catch bursty flows as Section 2.3 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 small 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 4

PROBLEM DEFINITION

To make progress in large flow detection, we aim 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.

4.1 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 $\mathsf{TH_h}(t_2 - t_1)$; 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: $\mathsf{TH_h}(t) = \gamma_h t + \beta_h$ 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]

[1]The Definition 1 is derived from Wu et al's paper [1].

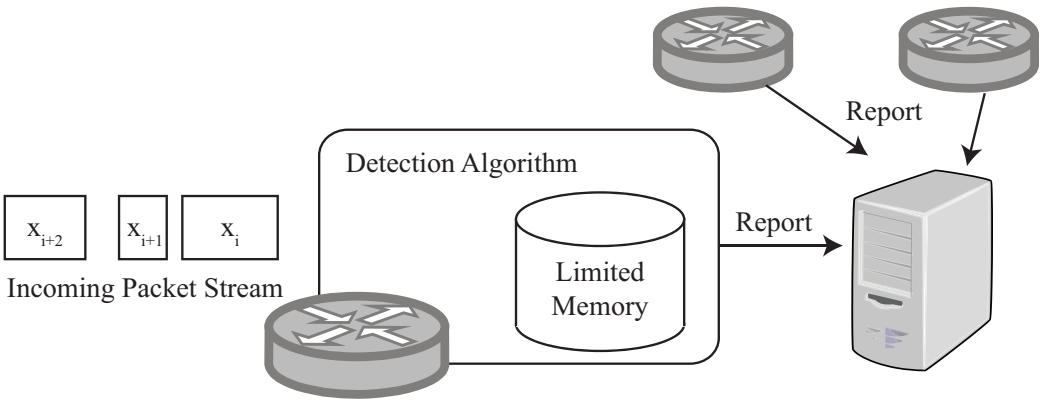


Figure 4.1: A general framework for a large-flow-detection algorithm. The detection algorithm processes incoming flows and keeps limited states in memory. Results may be reported to a remote server for further analysis [1].

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 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 [19], 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 4.1.[2] Therefore such large flow detection algorithm have to correctly make response without knowledge from the flow set $\mathcal{F}$.

4.2 Design Goal

Exactness outside an ambiguity region We want to design a deterministic large flow detector which can accurately identify every large flow

[2]Figure 4.1 is taken from the paper written by Wu et al. [1] ©2014 Association for Computing Machinery, Inc. by permission.

(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).

Scalability In front of a high rate link, the large flow detector should maintain 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.

CHAPTER 5

ALGORITHM

According to the design goals described in Section 4.2, 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 [4], EARDET takes the no-FN$_\ell$ 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.

5.1 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.

5.2 Algorithm Description

We show how EARDet works in Algorithm 1.[1] 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 5.1[2] 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.

5.3 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

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

[2] Figure 5.1 is taken from the paper written by Wu et al. [1]

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 ($c_f > \beta_{TH}$)

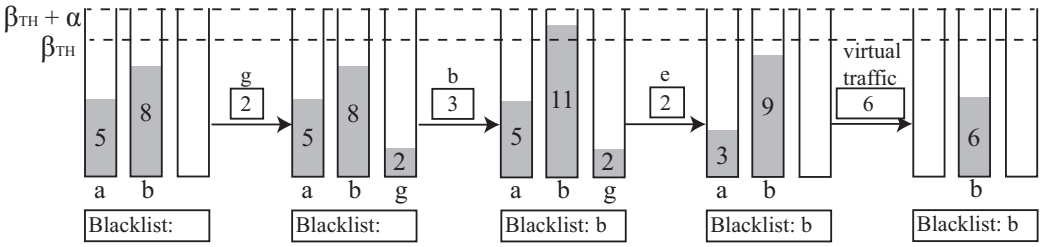


Figure 5.1: Example of updating EARDET status. $\beta_{TH} = 10$, $\alpha = 3$, and $n = 3$ [1].

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 retrieving 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-size 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 $\mathsf{TH}_\mathsf{h}(t) = \gamma_h t + (\beta_h + 1)$ 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_i's 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 $\mathsf{TH}_\mathsf{h}(t) = \gamma_h t + (\beta_h + 1)$ 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.

CHAPTER 6

ALGORITHM ANALYSIS

In this chapter, we analyzed the unique properties in EARDet, and theoretically proved them. Moreover, we analyzed the trade-off 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.

6.1 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.3.

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

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.

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 $\beta_h \geq \alpha + 2\beta_{TH}$.*[2]

Proof sketch According to Lemma 2, in time interval $[t_1, t_2]$, the decrement of a counter will not exceed $\frac{\rho}{n+1} \cdot (t_2 - t_1) + \alpha + \beta_{TH}$. And 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:

$$\int_{t_1}^{t_2} R(t)\,\mathrm{d}t \geq TH_h(t_2 - t_1) \geq \frac{\rho}{n+1}(t_2 - t_1) + \alpha + 2\beta_{TH} \tag{6.1}$$

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

$$Remains = \int_{t_1}^{t_2} R(t)\,\mathrm{d}t - dec_f > \beta_{TH} \tag{6.2}$$

[1]Lemma 2 is taken from Wu et al.'s paper [1].
[2]Theorem 3 is taken from Wu et al.'s paper [1].

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.

6.2 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.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 $t_{\beta_\ell} = \frac{(n-1)\alpha+(n+1)\beta_\ell}{[1-(n+1)\gamma_\ell/\rho]\rho}$.*[3]

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 $0 < \beta_\ell < \beta_{TH}$, $\gamma_\ell < R_{NFP}$, where $R_{NFP} = \frac{\beta_\Delta}{(n-1)\alpha+(n+1)\beta_\ell+(n+1)\beta_\Delta} \cdot \rho$.*[4]

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

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

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, $\frac{(n-1)\alpha+(n+1)\beta_\ell}{[1-(n+1)\gamma_\ell/\rho]\rho} < \frac{\beta_{TH}-\beta_\ell}{\gamma_\ell}$,

$$\Leftrightarrow \gamma_\ell < \frac{\beta_\Delta}{(n-1)\alpha + (n+1)\beta_\ell + (n+1)\beta_\Delta} \cdot \rho \tag{6.3}$$

The theorem is proved. ■

6.3 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 $[t_1, t_2)$.

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 $t_{incb} < \frac{\alpha+2\beta_{TH}}{R_{atk}-\frac{\rho}{n+1}}$.*[5]

Proof sketch[6] 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

$$\begin{aligned} &R_{atk}(t'_2 - t'_1) = \frac{\rho}{n+1}(t'_2 - t'_1) + \alpha + 2\beta_{TH} \\ &\Rightarrow t_{incb} < t'_{incb} = t'_a - t'_1 \leq t'_2 - t'_1 = \frac{\alpha + 2\beta_{TH}}{R_{atk} - \frac{\rho}{n+1}} \end{aligned} \tag{6.4}$$

Thus, the theorem is proved. ■

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

6.4 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.

CHAPTER 7

EVALUATION

In this chapter, 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) [3], to evaluate performance of EARDet. In terms of the exactness outside the ambiguity region, the evaluation shows that EARDet outperforms the prior work in both large rate flow detection and burst flow detection.

7.1 Theoretical Evaluation

As introduced in Section 3.2, 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 7.1.

Table 7.1: Numerical Example for FMF, AMF, and EARDet.

Detector	Number of Counters (n)	Rate of FP_s	Rate of FN_ℓ
EARDet	101	0	0
FMF	101	no guarantee	0*
FMF	1000	≤ 0.04*	0*
AMF	101	no guarantee	0
AMF	2000	≤ 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 7.1 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 7.2. 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 7.2: Comparison Summary for FMF, AMF, and EARDet.

Detector	Storage Cost	No-FP_s	No-FN_ℓ	Deterministic
EARDet	low	guarantee	no guarantee	yes
FMF	high	no guarantee	no guarantee	no
AMF	high	no guarantee	guarantee	no

7.2 Experimental Evaluation Environment

Traffic datasets To make the experiment more convincing, we use real network traffic datasets Federico II [20–22] and CAIDA [23], 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 [24,25], 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 7.3. 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. [26].

Table 7.3: Parameters of EARDet.

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

Table 7.4: 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 7.5: 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 7.4 and Table 7.5.

7.3 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 7.1, Figure 7.2, Figure 7.3(a) to 7.3(h), and Figure 7.4 are taken from paper written by Wu et al. [1]

Figure 7.1 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 7.2 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 7.3(a) to 7.3(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 7.3(a) and Figure 7.3(b) indicate that in the congested link, FMF

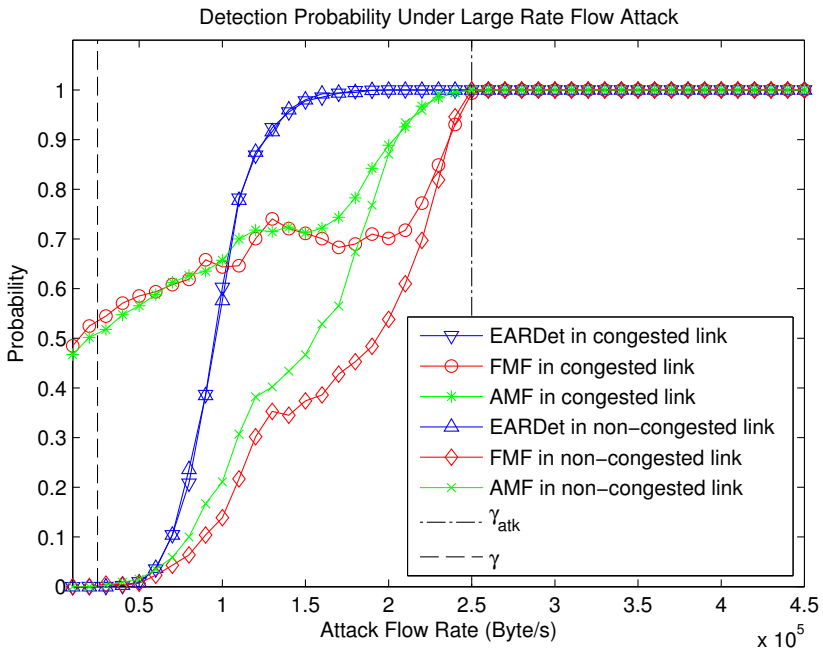


Figure 7.1: Detection probability under flooding DoS. FMF and AMF use 55*2 counters [1].

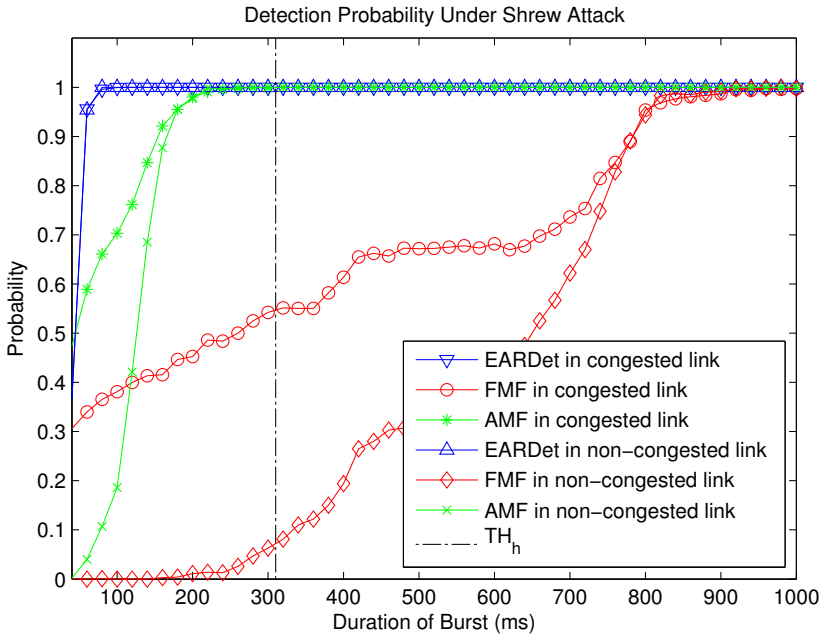


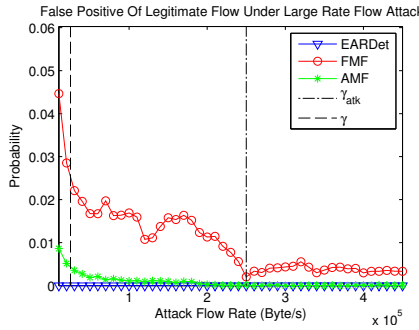
Figure 7.2: Detection probability under shrew DoS. FMF and AMF use 55*2 counters [1].

suffers more 1% and 4% false positives under shrew DoS attack and flooding DoS attack respectively. Increasing the number of counters can reduce the

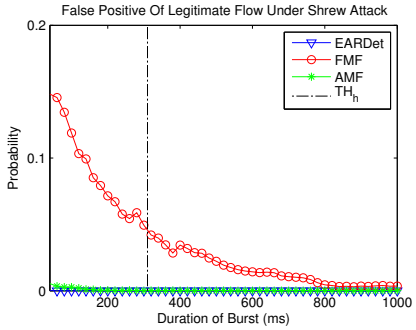
false positives for FMF and AMF, but it is impossible to guarantee no false positive.

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.

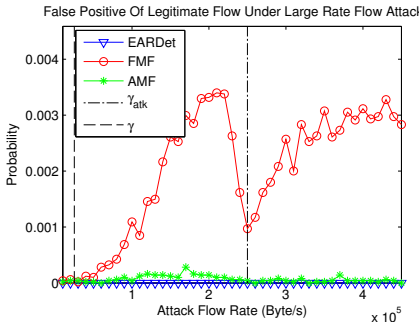
Figure 7.4 perfectly supports Theorem 6. The figure shows that the maximum incubation period of attack flows is always below the theoretical upper bound no matter what the attack flow rate is. Moreover, we observed that usually the maximum incubation period is much smaller than the theoretical upper bound and the average incubation period is even much smaller.



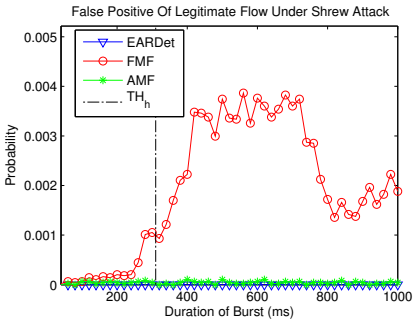
(a) 55*2 counters - Congested Link



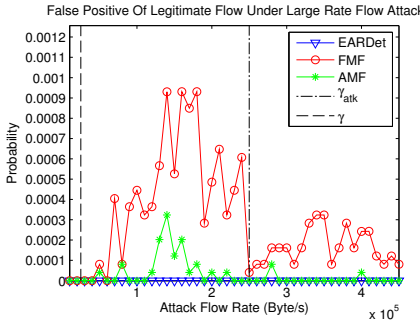
(b) 55*2 counters - Congested Link



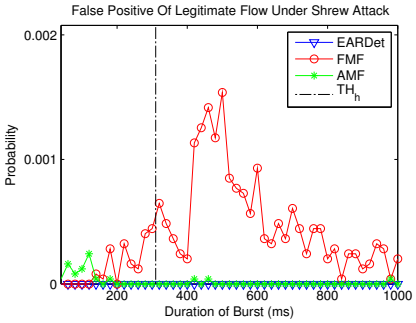
(c) 55*2 counters - Non-congested Link



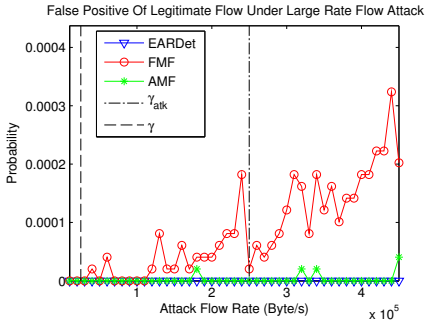
(d) 55*2 counters - Non-congested Link



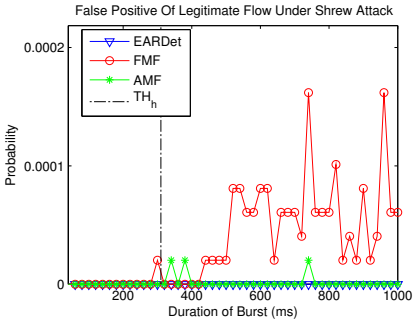
(e) 250*2 counters - Congested Link



(f) 250*2 counters - Congested Link



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



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

Figure 7.3: False positive for small flows [1].

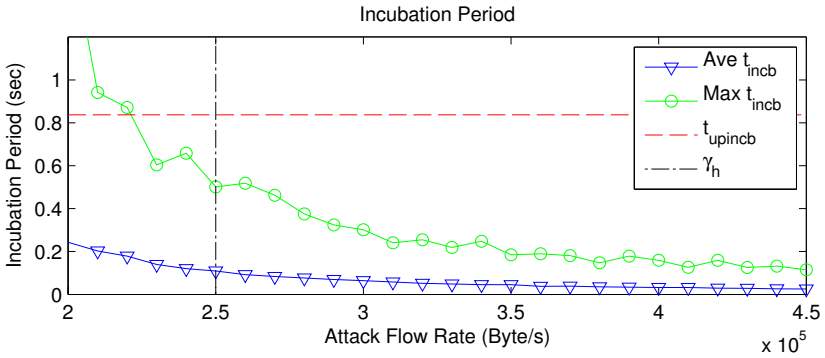


Figure 7.4: Incubation period for large flows [1].

CHAPTER 8

CONCLUSION

This thesis reviews some basic knowledge and typical existing approaches in the large flow detection problem, and identifies the shortcomings of current work. Then, we proposed a novel arbitrary-window-based algorithm, EARDet, 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.

There is some future work for EARDet. (1) EARDet is quite simple and easy to apply in industry, therefore, we want to build a real system with the EARDet 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.

APPENDIX A

PROOF SKETCH FOR LEMMAS

Note that Appendices A.1 and A.4 are presented in the technical report by Wu et, al. [26].

A.1 Lemma 7 and Proof Sketch

Lemma 7 *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.2 Lemma 8 and Proof Sketch

Lemma 8 *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 7 , 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 8. Therefore, to prove Lemma 8, we just need to consider the scenario of $k = 0$ in Lemma 7. According to Lemma 7, 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 8 is $\frac{\rho}{n+1} \cdot (t_2 - t_1)$, and this lemma is proved. ■

A.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 8, 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 8 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.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 7, the t_2-t_1 in Lemma 7 is smaller and the decrement is smaller. Therefore, in the case mentioned above, the decrement is minimized. According to Lemma 7, 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 $t_{ocp}>t_{\beta_\ell}=\frac{(n-1)\alpha+(n+1)\beta_\ell}{[1-(n+1)\gamma_\ell/\rho]\rho}$,

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

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

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}. ■

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 *IMC '14 Proceedings of the 2014 Conference on Internet Measurement Conference*. New York, NY, USA: 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] C. Estan, "Internet traffic measurement: What's going on in my network?" Ph.D. dissertation, University of California, San Diego, 2003.

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

[5] 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/

[6] P. Ferguson and D. Senie, "Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing," RFC 2827 (Best Current Practice), May 2000, updated by RFC 3704. [Online]. Available: http://www.ietf.org/rfc/rfc2827.txt

[7] 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

[8] 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

[9] 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

[10] 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

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

[12] 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

[13] 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

[14] 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

[15] 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

[16] 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

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

[18] 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.

[19] "Random sampled NetFlow." [Online]. Available: http://www.cisco.com/en/US/docs/ios/12_0s/feature/guide/nfstatsa.html

[20] 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.

[21] 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, pp. 945–968, 2009.

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

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

[24] 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

[25] 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

[26] H. Wu, H.-C. Hsiao, and Y.-C. Yu, "Efficient large flow detection over arbitrary windows: An algorithm exact outside an ambiguity region," CMU-CyLab-14-006, CyLab, Carnegie Mellon University, Tech. Rep., 2014.