UNIVERSITY OF CALIFORNIA
RIVERSIDE

Resource Efficient Frameworks for Network and Security Problems

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Azeem Aqil

December 2017

Dissertation Committee:

Srikanth V. Krishnamurthy, Chairperson
Zhiyun Qian
Evangelos Papalexakis
K. K. Ramakrishnan

The Dissertation of Azeem Aqil is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

There are a lot of people who deserve my gratitude. First, I am grateful to my advisor, Srikanth Krishnamurthy, for everything he has done. For his faith in me, and for keeping me focused, I will always be grateful.

Over the past 5 years, I have been privileged to work with some excellent collaborators and mentors and I want to thank them all.

I want to thank my committee members, Zhiyun Qian, K.K Ramakrishnan, and Evangelos Papalexakis, for agreeing to serve. I want to thank K.K and Vagelis for all the help and support with my last project. I especially want to thank Vagelis and Karim Khalil; I cannot imagine finishing *Jaal* without you.

I want to thank all my friends, who made the struggle easier. A PhD is, as much as anything can be, a synergistic effort.

Finally, for Maa, Abba and Rabea, gratitude is an understatement.

Published Works: Chapters 2, 3, 4 and 5 were publised at IEEE MILCOM 2015, IEEE CNS 2016, ACM CoNEXT 2017 and IEEE ICNP 2015 respectively. Chapter 6 was under submission at a conference when this thesis was submitted in Fall 2017

For Maa and Abba.

ABSTRACT OF THE DISSERTATION

Resource Efficient Frameworks for Network and Security Problems

by

Azeem Aqil

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, December 2017
Srikanth V. Krishnamurthy, Chairperson

In recent years we have witnessed an almost exponential growth in traffic that is transferred over the Internet. This traffic growth is a result of various innovations in network technology. However, this growth has also introduced various problems with regards to performance and security in modern networks. In this thesis we develop various frameworks that address some of those problems. The thesis is divided in two parts, the first deals with security issues and the second deals with performance issues.

In the first part, we analyze how modern day IDS systems are unable to efficiently cope with both the volume of traffic and the amount of information that can be collected from modern day systems. To deal with information overload, we develop and automated framework for feature selection that specifies the optimal set of features that a given system should monitor. This allows the IDS to focus only on what is important. To address the scalability problem in IDS systems, we develop Jaal, a framework that enables intrusion detection at ISP scales. The key idea in Jaal is to monitor traffic and construct in-network

packet summaries. The summaries are then processed centrally to detect attacks with high accuracy.

In the second part, we first highlight how battery technology is often the performance bottleneck in smartphones. Consequently we develop a mathematical framework that accurately predicts how much energy a given high definition video will consume on a mobile device. The predictive framework empowers user by pre-calculating energy consumption and letting the user decide whether a certain video download is worth the energy budget available. Finally, we develop NEST, a novel transport framework for delivering extractive summaries of a dataset distributed across multiple producers over NDN. The goal is to exploit diversity in network conditions between a consumer and different producers towards delivering the consumer-specified summary while minimizing latency.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet is arguably the backbone of the modern digital era. The volume of traffic that traverses the Internet has steadily been increasing . The traffic growth is an almost direct consequence of and a testament to many of the technological strides in networks in particular and the Internet in general.

This unprecedented traffic growth, unfortunately, has also resulted in several problems that only become visible at this scale. In this thesis, we examine some of those problems and propose efficient frameworks for dealing with them. This thesis tackles two main areas. Chapters 2 through 4 address scalability problems in modern network based Intrusion detection systems (IDS's) while chapter 5 and 6 address performance issues.

Security: The main focus of this thesis is directed towards addressing problems of scale in IDS's. Increasing network capacities, large traffic volumes, and increasingly complex systems are all moderns day phenomenons that have directly contributed to increasing the available attack surface area. Consequently, we have witnessed a rise in both the

complexity, and the sheer number of network based attacks. Modern intrusion detection systems, unfortunately, have not been able to adequately deal with this combination of complexity and volume

In chapter 2 we begin by demonstrating the ease with which new attacks can be constructed to target today's network. Denial of service (DoS) attacks are among the most crippling of network attacks because they are easy to orchestrate and usually cause an immediate shutdown of whatever resource is targeted. Today's intrusion detection systems check if specific single scalar features exceed a threshold to determine if a specific TCP-based DoS attack is underway. To defeat such systems we demonstrate that an attacker can simply launch a combination of attack threads, each of which on its own does not break a system down but together can be very potent. We demonstrate that such attacks cannot be detected by simple threshold based statistical anomaly detection techniques that are used in today's intrusion detection systems. We argue that an effective way to detect such attacks is by jointly considering multiple features that are affected by such attacks. We demonstrate that this approach is extremely effective in detecting stealthy DoS attacks; the true positive rate is close to 100

The detection approach however, while effective for the attack we constructed, does not scale because it requires the manual identification of features that must be monitored for detection to be successful. Specifying such a set of features for every possible attack is too onerous.

In chapter 3, we present an effective solution for the above stated problem. We note that one can traditionally gather a lot of data from different sources (packet contents,

application logs, OS behaviors, etc.) as evidence that could be used for intrusion detection. However, it is not easy to determine which of these evidence vectors or features are useful in facilitating highly accurate intrusion detection. In the second part of the proposal, we undertake an in-depth experimental study to determine whether appropriately trained search algorithms can help us find the right set of features for detecting a class of attacks (e.g., denial of service). The output of such algorithms yields a set of features that should potentially improve detection accuracy. Towards this we monitor 365 features across system layers and compare the detection performance of 3 popular feature selection algorithms to reduce the state space of the feature set for two classes of attacks. We find that the approach can yield significantly improved detection accuracy in comparison to statically chosen single features, sub or super sets of features of what the algorithms yield. Finally in chapter 4 we tackle the problem of Intrusion detection at scale.

We note that recently we have seen an increasing number of attacks that are distributed, and span an entire wide area network (WAN). Today, typically, intrusion detection systems (IDSs) are deployed at enterprise scale and cannot handle attacks that cover a WAN. Moreover, such IDSs are implemented at a single entity that expects to look at all packets to determine an intrusion. Transferring copies of raw packets to centralized engines for analysis in a WAN can significantly impact both network performance and detection accuracy. In chapter 6, we propose *Jaal*, a framework for achieving accurate network intrusion detection at scale. The key idea in *Jaal* is to monitor traffic and construct in-network *packet summaries*. The summaries are then processed centrally to detect attacks with high accuracy. The main challenges that we address are (a) creating summaries that are concise,

but sufficient to draw highly accurate inferences and (b) transforming traditional IDS rules to handle summaries instead of raw packets. We implement *Jaal* on a large scale SDN testbed. We show that on average *Jaal* yields a detection accuracy of about 98%, which is the highest reported for ISP scale network intrusion detection. At the same time, the overhead associated with transferring summaries to the central inference engine is only about 35% of what is consumed if raw packets are transferred.

Performance: The second part of the thesis focuses on performance issues that arise due to the expectations placed on modern networks. In chapter 5 we argue that streaming video content over cellular connectivity impacts the battery consumption of a client (e.g., a smartphone). The problem is exacerbated when the channel quality is poor because of a large number of retransmissions; moreover, streaming high quality video in such cases can negatively impact user experience (e.g., due to stalling). In chapter 5, we develop an analytical framework which can provide the user with an estimate of "how much" energy she can save by choosing to view a lower quality stream of the video she wishes to view. The framework takes as input the network conditions (in terms of packet error rate or PER) and a coarse characterization of the video to be viewed (slow versus fast motion, resolution), and yields as output the energy savings with different resolutions of the video to be viewed. Thus empowered, the user can then make a quick, educated decision on the version of the video to view. We validate that our framework is extremely accurate in estimating the energy consumption via both simulations, and experiments on smartphones (within $\approx$ 5% of real measurements). We find that switching to a lower resolution video can potentially lead to $\approx$ 418 mW (23.2%) decrease in the consumed power for slow motion

video, and $\approx$ 480 mW (26%) for fast motion video in bad channel conditions. This translates to an energy savings of 376.2 J and 432 J respectively, for video clips that are 15 minutes long. Finally, in chapter 6 we tackle the problem of efficiently transferring content over the internet. In many emerging data retrieval applications, consumers are interested in getting a summarized version of content quickly rather than retrieving all available data, in response to a query. Recently, Named Data Networks (NDN) have been considered for efficient transfer of summarized information, but the research is still in its infancy. In chapter 6, we propose *NEST*, a novel transport framework for delivering extractive summaries of a dataset distributed across multiple producers over NDN. The goal is to exploit diversity in network conditions between a consumer and different producers towards delivering the consumer-specified summary while minimizing latency. *NEST* first creates a unified hierarchical representation of the available distributed content using state-of-the-art distributed clustering. Then, using this representation of the dataset, the protocol creates interest messages that enable consumers to opportunistically retrieve data objects from the best producer according to the dynamic network conditions, capitalizing on the flexibility offered by the NDN infrastructure. We implement *NEST* on the Mini-NDN network emulator and evaluate it's performance using datasets collected from Twitter. Our experimental results show that *NEST* takes advantage of producer diversity achieving large latency reduction gains of up to 100% compared to baseline protocols.

Chapter 2

Detection of Stealthy TCP-based DoS Attacks

Denial of service (DoS) attacks are among the most common of all network attacks [38]. Despite being well studied and several detection/preventive measures in place, DoS attacks continue to be prevalent today. The inherent ease with which DoS attacks can be initiated makes them attractive. Most attacks require minimal resources but can induce potentially crippling effects on the target.

Traditional detection engines for such attacks are of two types. Anomaly detection systems try to flag statistically significant deviations from normal behavior ([158] [111]) but such systems are limited by their choice of features and by the definition of normal behavior. Signature based systems essentially look at single scalar features for the purposes of detecting anomalies. For example, a high number of open but relatively unused ports

would suggest that a TCP SYN flood attack is possibly occurring. Signature based schemes see much greater practical deployment[18]

In this chapter, we argue that an attacker can simply undermine the efficiency of such practically deployed systems by combining a plurality of TCP-based DoS attacks. Essentially, he would use multiple different attack threads, each of which by itself would not overwhelm the system; however, jointly the threads would have the potency of a powerful DoS attack. More importantly, such an attack strategy defeats traditional intrusion detection systems that look for a single scalar threshold to be exceeded to issue an alert with regards to a TCP-based DoS attack; since the aggressiveness of each thread is moderate (controlled), these scalar thresholds are never exceeded causing the detection to fail. Thus, it is essential for a detection system to identify the effects caused by each attack individually and carefully assess the joint occurrence of such effects. If the evidence suggests that this to be the case, the detection engine can issue an alert.

We argue in this chapter that an effective way to detect such attacks is by jointly considering a plurality of features (as opposed to a single scalar feature). We identify a basis set of such features and show that these are all affected in different ways by different TCP-based DoS attacks. Our experiments also help us design and implement a detection approach that jointly considers whether each of these features is (a) above a high threshold or (b) below a low threshold. This examination facilitates the identification of stealthy combinations of TCP-based DoS attacks while maintaining a relatively low false positive rate. In brief, we make the following contributions in this chapter.

- Via extensive experiments we demonstrate the potency of a stealthy DoS attack and its ability to defeat traditional threshold based intrusion detection systems.
- Using an experimental approach, we identify features that are affected by each type of considered DoS attack at multiple layers. A combined examination of these features can yield a better assessment of whether or not the system is under attack.
- We propose an approach to combine the considered features in an effective way. Via experiments, we show that our approach is extremely effective (True positive $\approx 98\%$ and false positive $\approx 20\%$) in detecting both the stealthy DoS attack and traditional DoS attacks.

Scope: While our approach is generic and can account for stealthy attacks that combine a large number of different TCP-based DoS attacks, for clarity and tractability, we only consider a stealthy attack that combines two popular TCP-based DoS attacks viz., the TCP SYN flood and the Slowloris attacks. To account for other possible attacks, an offline study to understand the effects of such attacks, and features that may be effective in identifying them is needed. However, we believe that the generic approach that we use to jointly consider the TCP SYN flood and Slowloris attacks can be applied in such cases.

Roadmap: The rest of the chapter is organized as follows. In Section 6.2, we present brief overviews of the TCP SYN flood and Slowloris attacks. In Section 3.7 we provide a summary of related work. In Section 2.3, we showcase the stealthy DoS attack, and identify features that can be used as a basis set for detecting such attacks. In Section 2.4, we discuss the key insights drawn from our experiments ; these lead to guidelines for design of detection approaches; experimental evaluation of such a design is then presented.

2.1 Background

We provide a brief background on the DoS attacks we consider in our study. Specifically, to demonstrate the effectiveness of a stealthy DoS attack, we combine a SYN Flood attack[107] and a Slowloris attack [30].

2.1.1 The TCP SYN Flood Attack

The TCP SYN flood attack is a TCP-based DoS attack and has been known to the community for a long time. The attack takes advantage of the traditional TCP three way handshake mechanism. Most implementations of TCP establish some system state when a TCP connection is initiated through a SYN packet. Since there are practical limits on how much state can be maintained, attackers send a high volume of TCP SYN packets until the combined effect of all the half open connections saturates some system resource. This attack can be launched with minimal resources by an attacker since he is not required to maintain any state. In fact, most SYN flood attacks are launched using spoofed IP addresses. There are numerous approaches to defending against SYN floods but the most widely used approach is Syncookies[107].

2.1.2 The Slowloris Attack

Slowloris is a TCP-based DoS attack that exploits HTTP. The attack establishes multiple connections to a HTTP server and keeps them alive by regularly sending incomplete HTTP headers. The attack is maintained by sending partial, incomplete HTTP requests to the server and this continues to hog the connection as the headers are received regularly by

the server. The idea is to cause a single server machine to maintain multiple connections until it runs out of all allocatable sockets and is thus, subject to DoS.

Slowloris is a different DoS attack as compared to the TCP SYN flood attack; it has different attacker and victim semantics. The attack requires much more attacker resources because it requires a single (powerful) machine that maintains multiple connections. However, the resource requirements are not too limiting because typically one could instrument the machine such that the attack program only wakes up periodically to send HTTP headers and then goes back to sleep. Application layer attacks like Slowloris are considered stealthy attacks because it's very hard to discriminate between legitimate traffic and attacks like Slowloris.

The attack semantics on the victim are also different. Instead of a traditional DoS attack like SYN floods where the victim machine is flooded with traffic, Slowloris attacks are characterized by bursts of traffic at regular intervals. This, potentially, makes this attack much more stealthy than high volume DoS attacks. The Slowloris attack can be further optimized if the server's connection timeout is known. This allows the Slowloris to minimize the number of times it has to send incomplete HTTP requests. There are various techniques, like load balancing and reverse proxies, that can be used to diminish the effect of Slowloris, but some versions of Apache, one of the most widely deployed servers, is still susceptible.

2.2 Related Work

In this section, we discuss existing detection schemes for DoS attacks. We also describe work relevant to mixed DoS and stealthy DOS attacks.

There are several efforts that target the detection of TCP SYN flood attacks. In [201], the authors use sequential change point detection to flag SYN floods. They analyze TCP behavior by looking at the number SYN and RST packets. However their approach is applicable only at leaf routers. The most prevalent detection and defense mechanism against SYN flood is the use of SYN cookies [107]. This approach detects attacks if the SYN buffer fills up. This is essentially a single threshold based technique and fails when a low intensity SYN flood is being employed with other attacks. Approaches such as the above, or popular signature based detectors such as those employed in Snort [31] and Bro [7] do not work against mixed DoS attacks because they do not collect and use adequate evidence information. In [202] the authors present a statistical approach to compute the correlation between requests and acknowledgments to detect anomalous behaviors; such an approach is reliant on some definition of normal traffic which can be tricky to characterize.There are other anomaly based detectors (such as [158] [111]) but they all face the problem of trying to accurately model or define normal behavior.

Slowloris is already considered a stealthy DoS attack. This is because it is very hard to differentiate between Slowloris traffic and normal traffic. It is also a relatively new attack.

There is little work on emerging application layer, TCP-based DoS attacks. In [105] the authors explore several application layer attacks but fail to provide any substantial

defense mechanisms. In [209], an approach to detect application layer flooding attacks is presented. The approach however, only focuses on the application layer and will miss any mixed attacks that target the network layer. In [112], the authors model HTTP traffic with the aim of flagging anomalous behavior but it too suffers from being limited to the application layer.

There have been various Internet reports of attackers using multiple kinds of DoS attacks to achieve their goals ([14], [33]) but to the best of our knowledge, our work is the first to analyze the affects of a joint network and application layer TCP-based DoS attack.

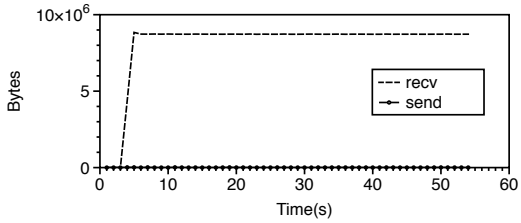
2.3 Stealthy DoS attacks: Impact and Key Characteristics

In this section, we conduct an experimental study to demonstrate the potency of a mixed or stealthy DoS attack and how such attacks can easily slip under the radar with respect to today's DoS detectors. We also identify key features that should be jointly considered for detecting such attacks.

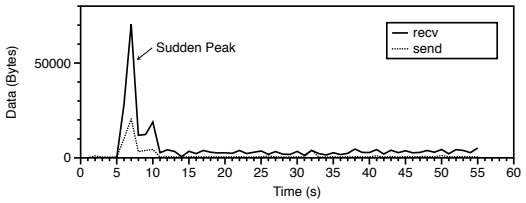
Experimental Setup: All our experiments are conducted on the DeterLab test bed [13], a state of the art scientific computing facility for cyber security research. Our network topology consists of one victim machine, and two attacker machines that are connected to the victim machine. The links between the machines can be tuned to incorporate varying delays and different degrees of reliability (packet success). We also have a legitimate machine that issues requests and measures response times to the victim. The victim machine is running Ubuntu server version 14.04 and apache web server[6] version 2.2. The attacker machines are both running Kali Linux[19], a Linux based penetration testing distribution.

Designing a stealthy attack: As discussed in Sec 6.2, traditional DoS attacks can be detected by simple statistical scalar measures. SYN floods are typically detected by examining if the rate at which SYN packets are received exceeds a certain threshold [107]. A Slowloris alert is issued if the number of incomplete HTTP headers is higher than a threshold. Our goal here is to experimentally determine the optimum detection threshold for our server by launching full scale Slowloris and SYN flood attacks and measuring the change in response as measured by the observer machine. The stealthy attack would then combine a mix of the two attacks, but each individually below the threshold determined as above. Full scale SYN flood is the maximum number of packets our SYN Flood program is able to generate (100 SYN Packets/Second). Full Scale Slowloris, is the max number of malicious connections our program can maintain (500 simultaneous connections).

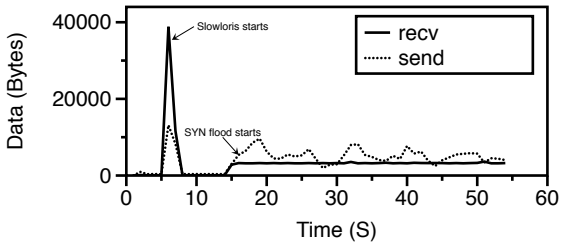
To determine detection thresholds we launched each individual attack, starting from the highest possible frequency, and reducing this gradually, while measuring response time (with respect to a benign scenario) from the observer machine. We set the detection threshold to the point where the attacks were completely imperceptible from the point of view of the observer machine. We set our stealthy attack rates to be below this threshold; this results in our mixed attack evading the single threshold detection mechanisms. The detection thresholds are presented in table 2.1. **Determining Key Features:** What are the features that facilitate the effective detection of both full scale and stealthy mixed attacks? This is the question we seek to answer here. Given how these attacks function, we chose a set of 6 performance metrics as features which can potentially characterize these



(a) SYN flood



(b) Slowloris

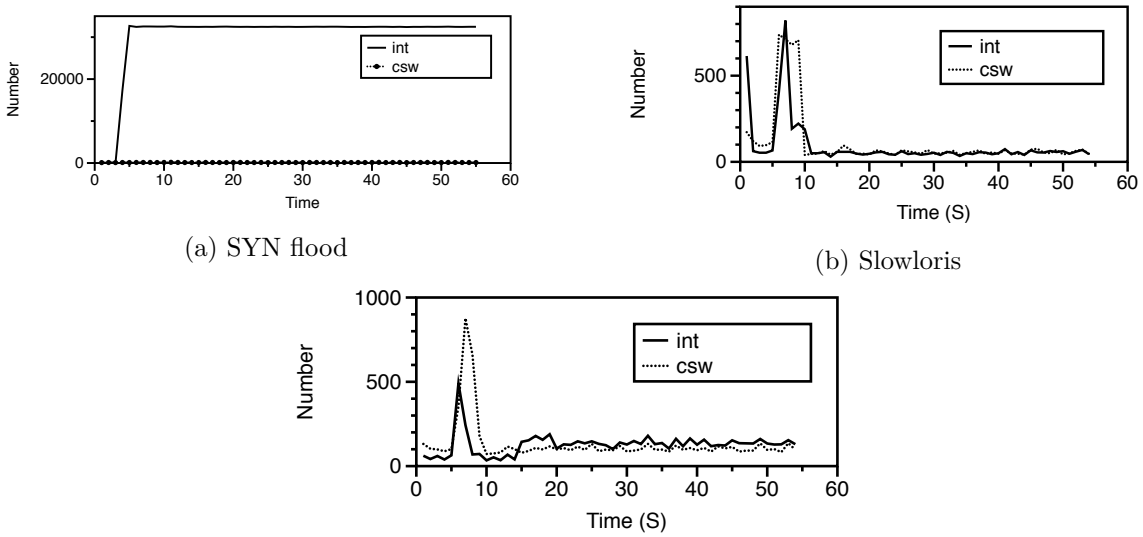


(c) Mixed attack

Figure 2.1: Data received and sent data during (a) a full-scale SYN flood; (b) a full scale Slowloris attack (c) a mixed attack

attacks. The first two features are (i) the volume of data sent and (ii) the volume of data received. These two metrics are applicable for the following reasons. Attacks like SYN floods are characterized by a disproportionate amount of traffic received versus traffic sent. For high volume SYN flood DoS attacks, this single feature is often enough to identify attacks. We do not expect stealthy DoS attacks like Slowloris to display the same disproportionate traffic levels, they do however cause the volume of traffic sent to almost zero out. This feature by itself is not enough to identify an attack but does warrant suspicion.

The next two features in our set are system interrupts and context switches. Interrupts and context switches exhibit similar behaviors but are semantically different. A context switch occurs when the OS switches from the currently running execution thread. The kernel saves the state of the running execution thread and loads in a new one. An interrupt, however, occurs when the executing process receives an asynchronous signal requiring



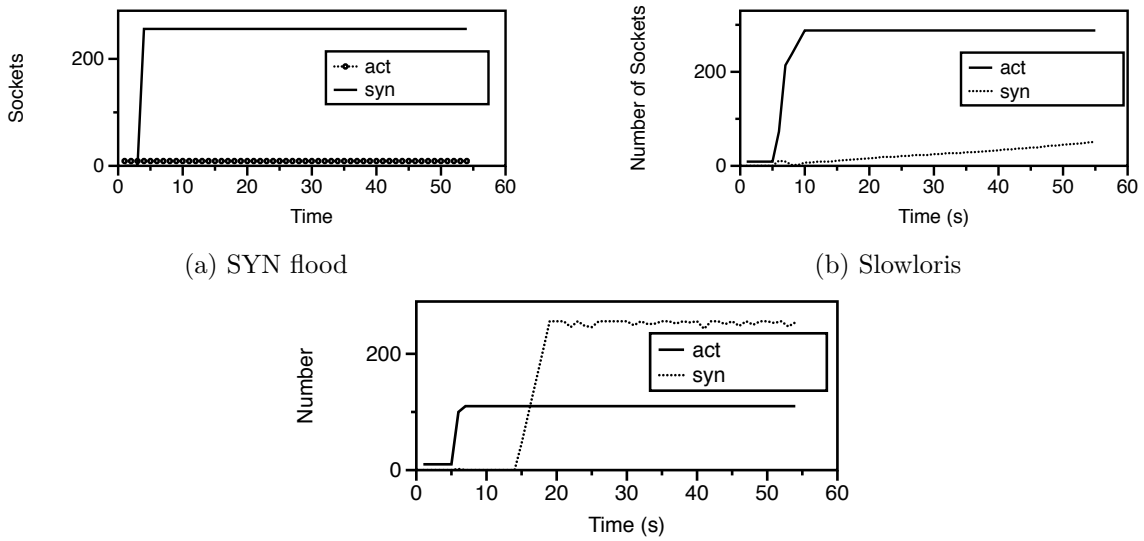
(a) SYN flood

(b) Slowloris

(c) Mixed attack

Figure 2.2: Interrupts and context switches during (a) a full-scale SYN flood (b) a full scale Slowloris attack (c) a mixed attack

attention. Interrupt routines do not change context; they return to the same execution thread after the interrupt is handled. Examining context switches indicate how much, or how little, work is being done by the server. Interrupts are important indicators of unexpected events. Every new TCP connection request (SYN packet) will trigger an interrupt. Too many interrupts, coupled with too few context switches (a normal functional server has to serve multiple requests which results in an inevitably larger number of context switches) can be a possible indication that the server is not performing useful work. In other words, a high number of interrupts, coupled with few context switches and a high data reception rate can effectively identify a SYN flood. Slow loris is also likely to exhibit few context switches. The number is likely to be higher than that with SYN floods because actual connections are established here; however, it will still be low because these connections just wait rather than performing useful work.



(a) SYN flood

(b) Slowloris

(c) Mixed attack

Figure 2.3: TCP Sockets during (a) a full-scale SYN flood; (b) a full scale Slowloris attack (c) a mixed attack

The final two features that we consider are the number of TCP sockets that are in the SYN state and in the established state (ACT sockets), respectively. Keeping track of sockets in the SYN state is useful because SYN flood attacks try and exhaust the SYN buffer. The number of sockets in the established state will include those connections that are induced by a Slowloris attacker. A slowloris attack will result in many sockets in being in the established state. High socket occupancy coupled with low data volumes and low system interrupt rates could allow us to identify a Slowloris attack.

From everything we have learned, we expect mixed attacks to exhibit the following characteristics (i) A large combined number of SYN and ACT sockets (ii) a relatively high interrupt rate (depending on the intensity of SYN floods), (iii) low volumes of sent data sent and (iv) few context switches.

Discussion: The above features are readily obtained by reading certain hardware registers. However, we point out that these features are by no means exhaustive. By including additional features, the accuracy of detection of stealthy attacks can possibly increase significantly. Our objective here is to showcase the potential of the approach, rather than find an exhaustive set of such features.

Experimental validation of our feature set: Next, we seek to demonstrate experimentally, that our chosen feature set can lead to an effective detection approach.

We launched full scale SYN Flood and Slowloris attacks (characterized in table 2.1) against our server to analyze the effectiveness and behaviors of our chosen features. Panels a,b of Figs 2.1 to 2.2 depict the results of those experiments. By combining the previously described DoS attacks in stealthy modes (note that each component by itself in such mixed attacks do not cause a performance degradation), we launched our mixed (stealthy) attack. The effects of the stealthy attack on the considered features, are shown in Figures 2.1c, 2.2c and 2.3c.

We analyze the behavior of these features, and this is the lynchpin of our detection approach.

Figs. 2.1a and 2.1b depict the volume of traffic during full fledged (or full scale) attacks while the same feature is presented in Fig. 2.1c for the mixed attack. As expected, SYN floods are characterized by a very high volume of traffic received and disproportionately low volume that is sent. Slowloris (fig 2.1b) is characterized by a spike in traffic volume (when the connections are established) but the volume of data tapers off. The key observation is that in all three cases (SYN Flood, Slowloris and the mixed attack) the data

	SYN Flood (SYN Packets/S)	Slowloris (Connections)
Full Scale Attack	100	500
Stealthy Attack	7	100
Detection Threshold	10	120

Table 2.1: Experimental Parameters

sent is always below 800 bytes. For full scale SYN flood, our experiments show that the data received is always above 70,000 bytes. However, for the mixed attack the volume of data is much lower.

Figs. 2.2a and 2.2b demonstrate the behaviors in terms of interrupts and context switches with full scale attacks. For SYN floods we see that interrupts consistently dominate context switches (every new connection request triggers an interrupt) while for Slowloris we only see a spike that correlates with network data and then, few context switches and interrupts. For the mixed attack (Fig 2.2c), we observe relatively high values for Interrupts (corresponding to SYN Flood) and few context switches. For both full scale SYN flood and mixed attacks, the number of Interrupts (in a short span of time) is greater than 200 and the number of context switches is less than 100.

Finally, Figs. 2.3a and 2.3b capture the behavior of TCP Sockets for full scale attacks while Fig. 2.3c does the same for mixed attacks. As expected, we see that a full scale SYN flood results in a large number of SYN sockets and Slowloris results in many established sockets. Mixed attacks on the other hand, yield a high number when the two are combined.

Feature	High Threshold	Low Threshold
Context switches	600	100
Interrupts	200	N/A
Data Received	70,000	450
Data Sent	N/A	800
TCP SYN sockets	200	9
TCP ACT sockets	95	9

Table 2.2: Algorithm Variable Description

Data: Set of features S aggregated over 3 seconds
Result: Attack Classification
if *Any feature in S above its high threshold* **then**
  θ = All feautures above their high thresholds;
  $\Theta = S - \theta$;
  if *All features in* Θ *below their low thresholds* **then**
    OUTPUT "attack";
  end
  else
    OUTPUT "no attack" ;
  end
end

Algorithm 1: Attack Classification

The experiments demonstrate that the behaviors of the key features that we choose are as expected. We leverage these behaviors to design an effective detection approach as will be described in the following section.

2.4 A framework for detecting stealthy DoS attacks: design and validation

In this section we leverage the take aways from the previous section to design our detection framework and experimentally demonstrate its effectiveness.

	True Positives	False Positives
SYN Flood-Full	100	N/A
Slowloris-Full	95	NA
Mix - 1	100	N/A
Mix - 2	100	N/A
Mix - 3	100	N/A
Normal Traffic	N/A	20

Table 2.3: Results with our detection approach

	True Positives	False Positives
SYN Flood-Full	100	N/A
Slowloris-Full	100	NA
Mix - 1	0	N/A
Mix - 2	0	N/A
Mix - 3	60	N/A
Normal Traffic	N/A	60

Table 2.4: Results with scalar threshold detection

2.4.1 Design of our approach

Key Insights: As discussed in section 2.3, in order to detect both full scale attacks and mixed (stealthy) attacks we propose to use a combination of features. When under attack, some of these features exhibit high values while others tend have low values. In essence, since DoS targets the consumption of specific resources, those will be over utilized while certain other resources will be left under utilized. Legitimate web requests on the other hand, usually utilize (and thereby affect) a majority of the system, each having a part to play in servicing requests. This observation is the cornerstone of our detection framework.

Based on the understanding we gained with respect to our features in Section 2.3 we set two thresholds for each feature viz., a high threshold and a low threshold. Whenever any of the features crosses it's respective high thresholds (a sign of DoS), our detection

approach examines the other features to check if they are below their low thresholds. It is important to realize that this is fundamentally different from single threshold based approaches; the use of multiple features and two thresholds (low and high) help's reduce both false positives and false negatives.

Determining Optimal Thresholds: To determine it's high and low thresholds, we recall results from Section 2.3. As determined earlier, Slowloris (both full scale and stealthy) attacks result in (a) a high number of sockets in the ACT state, (b) few interrupts, (c) few context switches, (d) few SYN sockets and (e) low volume of data transfered. SYN floods have (i) high interrupt rates, (ii) high volume of data received (SYNs) and (iii) high number of sockets in SYN sate. From the results of our mixed attack we observe that together, they result in high number of interrupts (true for both SYN flood and Slowloris), and a high total number of sockets that are in ACT and SYN states. Thus, we must look for the crossing of high thresholds for these features.

We get our high threshold value for interrupts from the mixed attack empirically from Fig 2.2c (a more sophisticated machine learning approach can be potentially used but we leave this for future work). We obtain our threshold for high received data volume from Fig. 2.1a for full scale SYN floods since, high data volume is only a feature that is manifested with full scale SYN floods. High thresholds for sockets together in SYN and ACT state are both extracted from the results in Fig. 2.3c.

Low thresholds for ACT and SYN sockets (important in identifying full scale Slowloris and SYN Flood) are empirically obtained from Figs. 2.3b and 2.3a. The low threshold for data sent (characteristic of mixed attacks, full scale SYN flood as well as a full

scale Slowloris) is obtained from the results in Fig 2.1c. Table 2.2 summarizes the values of these thresholds.

Detection Algorithm: Our detection algorithm (Algo 1) is executed once every 3 seconds. It checks if any of the considered features are above their high thresholds; if there are such features, it checks to see if all other features are below their low thresholds. If this is true then, the algorithm flags an attack. It is easy to verify that when any of the DoS attacks is in progress (full scale or stealthy), there are certain resources that are heavily used (e.g., high received data volume), but inevitably, there are other parameters that indicate low usage of certain other resources (e.g., number of context switches). In other words, there is a serious imbalance in the way in which resources are utilized in a system. This in essence, is the lynchpin of the algorithm. By using two thresholds, the algorithm is effective in all cases (again, because any attack cannot saturate all resources).

2.4.2 Evaluation of our approach

Next, we conduct experiments on the Deter testbed to showcase the effectiveness of our detection framework. In order to evaluate how well our approach works in a real world setting, we need to generate realistic "normal" web traffic. To do so we used the extensive set of traffic traces detailed in [158] and available at [11]. The trace set contains network traffic collected at the edge router at a major university. We translated each incoming request (TCP packets with destination port set to 80) to a HTTP request for our server. We used a set of 10 traces. We consider multiple attack scenarios (full scale TCP, full scale Slowloris and 3 different mixed attacks). Our metrics of interest are the false positive and false negative rates. The 3 different kinds of mixed attacks represent different intensities of

the individual attack. Mix-1 is the initial mixed attack described in Section 2.3 from Table 2.1. Mix-2 incorporates a higher intensity Slowloris attack and a lower intensity SYN flood attack (Slowloris:110 connections, SYN:7 Packets/sec). Mix-3 includes a higher intensity SYN flood attack but a lower intensity Slowloris attack (Slowloris:100 connections, SYN:8.5 Packets/sec).

We compare our detection framework with traditional detection approaches that use a single scalar feature to determine if an attack is under way. Tables 2.1 and 2.2 describe the set up. Tables III and IV present our results. As evident from the results, our detection approach outperforms traditional approaches with respect to detecting mixed, stealthy attacks. The stealthy attacks Mix-1 and Mix-2 are completely undetected by the traditional schemes employing single thresholds. Mix-3 (which has a higher intensity of SYN Floods) was detected 60% of the time. This is because the combined SYN messages from the flood and those from the Slowloris component sometimes crossed the scalar detection threshold. Traditional approaches also do poorly in classifying normal traffic. In our experiments, traditional approaches erroneously issued alerts 60% of the time with normal traffic; these false alerts typically occurred during periods of high activity (i.e., sudden spikes of high volume but normal traffic). In contrast, the false alerts with our approach only occurred 20% of the time.

Chapter 3

Automated Cross Layer Feature Selection for Effective Intrusion Detection in Networked Systems

3.1 Introduction

There has been a recent increase in both the frequency and impact of cyber threats [5]. With network based attacks expected to rise [10], it is critical that highly efficient evidence collection and intrusion detection techniques be designed and deployed. Anomaly detection and signature based detection are the two most popular detection approaches (e.g., [71] [89]). The effectiveness of these approaches however, is tightly dependent on the underlying features (feature set of evidence) that are chosen.

There are multiple distinct sources of data that one can use to collect evidence. Features can be selected from the network, operating system, hardware or network layers. However, selecting the optimal subset of features to enable highly accurate intrusion detection is not easy. The quality of the eventual feature set is dependent both on the actual features, and the number of features. A set that is too small lacks the information to correctly reason about mutated or unknown attacks, while a set that is too large contains frivolous features that introduce noise and increase misclassification. Features are typically selected by studying the behavior of known attacks. This further complicates the problem for unknown or unseen attacks. Most detection approaches use features that have been carefully selected by domain experts. Such approaches, by definition, require precise knowledge about network threat semantics while also being prone to human judgment errors. Modern detection approaches also generally only use features from the network layer. Recent work has demonstrated the utility in considering features across different layers [68].

In this chapter, we seek to design a *unified* systematic framework to collect meaningful cross-layer features that are applicable to multiple classes of network attacks. Our framework must automate and sequentialize the process of feature selection and thereby enable the effective handling of high-volume data for highly accurate intrusion detection. Futher, we want our evidence collection to be general, in that the approach does not require deep knowledge about network behavior.

A high level depiction of our framework is shown in Figure 3.1. First, the system consists of an offline phase wherein it is trained with attack and normal behaviors. A large volume of evidence is collected offline from multiple layers and for each case (different

attacks, normal), an appropriate feature selection algorithm is then used to downselect the number of features. During runtime, only these downselected set of features are actively monitored. These are then fed to an inference engine which then provides an assessment of whether or not the networked system is under attack.

Challenges: There are a number of challenges that we need to address while building our framework. First, while some features are readily available, the networked system must be instrumented to collect other forms of evidence that are not exposed (e.g., create hooks in the OS). Second, one has to choose the right feature selection algorithm to achieve the right trade-off between accuracy and complexity. Towards understanding this trade-off we compare and contrast three algorithms, namely Linear Forward Selection (LFS), Sequential Backward Selection (SBS) and Simple Genetic Algorithm (SGA) on a large feature set that is collected. While one can expect the genetic algorithm to yield the highest accuracy it also is complex and takes a long time to run even on powerful servers; the others could potentially yield lower accuracy but run faster and on desktop computers. We seek to understand the tradeoffs between accuracy, complexity, and computation time for these three classes of algorithms.

To evaluate the effectiveness of automated cross-layer feature selection, we use the features selected with two detection (inference) engines viz., Dempster-Shafer Theory based inference and K-Means classification. The former outputs measures of likelihood (referred to as belief and plausibility) with respect to normal and different attack behaviors and is thus able to better differentiate between attacks and properly classify mutated attacks. The latter is arguably the most popular classification approach and allows us to reason about

the presence or absence of an attack, in more traditional terms. We point out here that the automated feature selection process is independent of the inference engine (i.e., any inference engine can be used to classify behaviors).

Our work in perspective: There have not been many attempts at addressing the problem we seek to solve. Attempts at applying feature selection algorithms to detect network based threats ([138], [164], [195] [128]) have considered only relatively small feature sets (less than 50) which are often chosen manually themselves. They are also specific to one particular kind of network attack. We consider feature sets that are much larger (> 300), and could reveal features that are better suited for specific attacks while covering large classes of attacks. We also seek to automate the entire feature selection process with little or no need for domain expertise about network threats. Virtually all attempts at detecting network threats, along with any attempts at feature selection, only consider network layer features. This is a problem because as new threats emerge, experts are forced to come up with increasingly novel ways to use network based features. We argue that contemporary approaches fail to capitalize on information that is present in features captured at other layers (OS, hardware, application). Further, prior approaches do not address the inherent uncertainty and noise that any selected features are bound to have. We point out that our work seeks to determine "what features to consider" while making an inference with regards to whether or not the networked system is under attack. It is agnostic to how those features are used in a detection engine. In our evaluations we show the effectiveness of our features with two different types of inference engines.

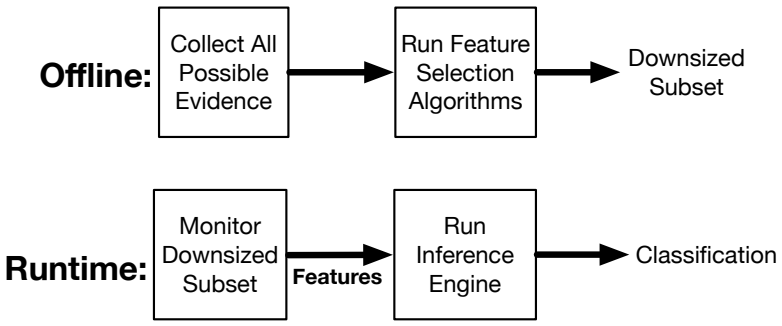


Figure 3.1: Overview of our approach

Contributions: To the best of our knowledge, this work is the first attempt at a unified approach (from collecting the initial set of features to downselecting to the eventual subset) at extensive cross layer feature selection for intrusion detection.

Specifically, we make the following contributions:

- We design a framework that sequentializes and automates the process of cross layer feature collection and downselection and eventual use of the downselected features for intrusion detection.

- Our framework incorporates a novel, evidence collection module that captures a relevant set of initial features by placing numerous monitors that are spread across the hardware, network, OS and application layers. We demonstrate this via a set of monitors that collect extensive cross-layer features for 3 DoS attacks, 2 SQL injection attacks, and normal behavior.

- Via extensive experiments, we test the feature selection capabilities of three popular search algorithms (LFS, SBS, SGA) and provide an analysis of their respective accuracy versus cost trade-offs. We find that LFS and SGA have similar results, in terms

of detection accuracy, for DoS attacks. This is very surprising since SGA performs a much more exhaustive search; LFS finishes in a fraction of the time it takes SGA to complete. SGA produces slightly better results (detection accuracy) for SQL injection and SBS selection lags behind the other two for both kinds of attacks. Given the significantly higher runtime associated with SGA, one has to carefully assess if the slightly increased accuracy (< 10 %) warrants the significant increase in complexity

- We build a complete system and show that feature selection algorithms lead to good detection accuracy not only in detecting known attacks but also a previously unseen attack that we construct. Our system decouples the feature selection process from the inference engine of an intrusion detection system, and is thus readily deployable.

- We show that selecting too few or too many features can actually hurt detection accuracy.

Scope: Our evidence collection approach is generic and we expect it to be applicable to other network based threats where features manifest at different layers. For tractability, we only test two different kinds of attacks. While our initial set of features is large (365 features), it is by no means an exhaustive set. However, other sources of evidence, when available, can be added to the initial set and our approach would still be applicable.

3.2 Evidence Collection

The first step in our framework is to collect a very large set of of features that is potentially relevant for intrusion detection. Evidence collection is generally a hard problem.

System Layer	Source of Evidence
Hardware	Hardware Counters, Perf Events
Network	Raw Stream Data, Packet Headers, Socket Statistics
Operating System	/proc Filesystem
Application	Log Files

Table 3.1: Source of Evidence

Modern systems are so complex that they present a seemingly limitless supply of features. This is one of the primary reasons why traditional anomaly detection systems rely so heavily on features selected by domain experts. However, not only is it hard to do this for all types of attacks, this may result in some key evidence features being left out from consideration. Our approach is to place monitors or sensors at multiple system layers. This is because the effects of attacks on networked systems are not felt in isolation at a single layer; their effects manifest themselves at different system layers. Some system based diagnostics are readily available (e.g., application logs), while for others we place hooks (e.g., in the OS) to collect the evidence. The rest of this section describes our monitoring of four system layers.

Hardware Layer: Typical Unix based systems export hardware statistics via hardware counters. These counters are intended for diagnostics purposes since they are indicators of the overall system health. They are extremely optimized with little sampling overhead. The values in these counters can be easily accessed via system API calls. We use all available hardware counters (on our test system) as features that can be sampled as a function of time. Examples of such features include CPU operating frequency, CPU utilization, memory consumption, cache hits, cache misses, core temperature, etc. The actual set of hardware level features that can be used is ultimately a function of the system under consideration.

OS Layer: The OS layer is the source of a wealth of information, not all of which, unlike hardware counters, can be easily accessible. However, the OS can be instrumented to provide information. We use the linux */proc* file system as our primary source of OS level features in this work. The */proc* is a virtual file system that exports information about OS state in the form of parsable text files (the files usually take the form *Variable: Value*). It is important to realize however, that the */proc* file system contains thousands of files (each process has its own set of files cataloging behaviors) the active monitoring of which, together can typically overwhelm a single server. For now, we only consider files in the top level directory and even this results in a large volume of evidence; in heavy duty custom systems, the sub-directories can be parsed also. We developed a file parser that periodically pings all top level files and records the value of each variable. We use a sampling frequency of one lookup per second so as to not overwhelm the server. Examples of features that can be obtained using this methodology include the number of system calls, the kernel load, the number of filesystem lookups and the number of system interrupts.

Application Layer: Primary sources of application layer information are log files. To detect DoS attacks we look at all log files produced by the Apache web server and to detect SQL injection attacks we look at all available MySQL and Wordpress logs. The methodology is similar to the one we described above but instead of */proc* files we parse application layer log files. We have developed a log file parser that checks log files periodically and records the value of each variable that is present. Examples of features that are extracted from log files include web server accesses, the number of requests that are being served, number of requests queued, database lookups and SQL errors.

Network Layer: At the network layer, examine the raw packet stream plus raw network statistics exported by the OS. We only consider packet header information as features. Packet analysis allows us to collect information such as the number of packets plus the kind of packet (TCP SYNs, TCP ACKs, etc). We primarily use the *netstat* tool which gives various network layer diagnostic information that we use as features. Examples include SYN packets, Sockets in ESTABLISHED state, sockets in CLOSED state and network bandwidth.

Table 3.1 summarizes the sources of the features we collect.The methodology outlined above allows us to collect a set of 365 features. This list is by no means exhaustive. For example, one potential source of evidence for SQL injection might be obtained by looking at the SQL queries themselves. This requires deep packet inspection, which we do not consider due to its computational overhead. Other evidence sources, however, can be added to our framework as desired.

In essence, the relevant features are selected by launching normal behavior and attacks. A large set of features is collected, and passed on to the feature selection algorithms. The algorithms each output a subset of features that they think are optimal. During runtime, these can then be sampled with higher frequencies to facilitate highly accurate detection.

3.3 Feature Selection

Our evidence collection module yields a large number of features (in our prototype we collect 365 features in total). The features are collected to classify each attack

and normal behaviors. Feature Selection (or choosing the best subset of features from a large set) involves two components. The first is an objective function and the second is a search algorithm. The objective function evaluates candidate subsets and returns a quantification of their "usefulness". This is then used by the search algorithms to select new candidate subsets. We want to select features that have high correlation with attacks and low cross-correlation across features (the latter identifies redundant and therefore unnecessary features.) We consider three search algorithms, viz., Linear Forward Selection (LFS), Sequential Backward Selection (SBS) and a Simple Genetic Algorithm (SGA).

Objective Function: We use the correlation based subset evaluator detailed in [116] as our objective function. This evaluation function has been shown to have good performance and it also ties in nicely with our search goal. It evaluates the usefulness of a subset by computing the correlation between the features and the classification classes (attacks and normal behavior.) It also tries to minimize the redundancy between features. Two features that are highly correlated (regardless of the labeled classes) are considered redundant and only one is selected. In simple terms, subsets of features that have low cross correlation and high correlation with the labeled classes (attacks and normal behavior) are preferred.

Linear Forward Selection: Linear Forward Selection [115] is an optimization of the popular search algorithm, Sequential Forward Selection [118] (SFS). In its simplest form, SFS starts with an empty set and sequentially adds features (i) such that at each step, $F(Y, i)$ is maximized, where Y is the set of previously selected features and F is a function that calculates the usefulness of a subset of fearures. SFS is essentially a simple

hill-climb search; it evaluates all possible single feature expansions of the current set. The feature that results in the highest score is added permanently and the algorithm terminates when no single feature expansion improves the current score. The problem with SFS is that the number of subset evaluations that must be performed grow quadratically with the number of features. This is not a big problem when the search space is small but becomes a concern for large sets. LFS improves on SFS by limiting the number of features that must be considered at each step to 1. This means that LFS has an upper bound on the running time given by $\frac{N(N+1)}{2}$ where N is the total number of features under consideration.

A forward selection algorithm however, cannot remove a priori added features that become redundant with the addition of new features. However, forward selection algorithms in general are known to perform well when the optimal subset of features is small. In addition, LFS has low computational overhead.

Sequential Backward Selection: Sequential Backward Selection [118] is the logical reverse of SFS. It starts with the full set of features and sequentially eliminates the feature i that "least" reduces the total value of the set under consideration (as measured by the objective function).

The main weakness of SBS is that it cannot re-examine the usefulness of a feature after it has been eliminated. SBS works best (in terms of the usefulness of its output) when the optimal feature subset is large because SBS, due to the fact that it starts with the full set, spends most of its time evaluating larger subsets.

Genetic Algorithm: Genetic Algorithms [75] are a class of search algorithm that try to emulate the process of Darwinian natural selection. A more detailed description

can be found in [75]. All genetic algorithms can be thought of as 5 stage processes. Initial population selection, fitness function application, selection, crossover and mutation. Genetic algorithms have been shown to produce very good results in a variety of domains [75]. However, they are also known to be computationally expensive.

A genetic algorithm begins with an initial population that consists of random subsets of solutions (which correspond to features in our context). The fitness function (synonymous with previously discussed objective function) evaluates the fitness of each member of this population. The genetic algorithm ranks each subset according to its fitness. Then, some of the fittest subsets are chosen to 'reproduce' to create a new generation of subsets. This process continues until a terminating condition is met. The question then is how this reproduction takes place. The reproduction of two pairs involves randomly selecting crossover points in the two pairs and combining them. As an example, consider two feature subsets that have been chosen to reproduce. The crossover process involves splitting both subsets at random points and combining the split of one subset with that of the other subset to come up with two new subsets. Each set of features is then subjected to a random mutation where elements in the set can randomly change (this probability is usually very low). The steps from selection to mutation are then repeated until a terminating condition is met or the algorithm is stopped. We use the Simple Genetic Algorithm [113] in our experiments.

The feature selection algorithms are given as input, the initial labeled set of features and they output subsets that they think are optimal. In Section 3.6, we provide a

detailed comparison of how the three algorithms described here perform with an evidence feature set for detecting DoS and SQL injection attacks.

3.4 Inference Engines

In this section we will describe the two inference engines we consider for detection viz, Dempster-Shafer Theory of evidence, and K-Means classification. In terms of traditional detection engines, DST is closest to supervised learning. However, it provides some advantages discussed later. K-means, on the other hand, is an unsupervised learning algorithm for classification. Our expectation is that the optimal set of features should be usable with any inference engine to accurately detect the presence or absence of attacks.

3.4.1 Dempster-Shafer Theory of Evidence

The Dempster-Shafer theory of evidence (DST) [187] is a theory for combining evidence and reasoning about uncertainty. We use it in our framework to reason about the quality of the feature subsets that are output by the feature selection algorithms in terms of detection accuracy. Every hypothesis is assigned a belief ranging from 0 to 1 where 0 means that there is no evidence to support a hypothesis and 1 means absolute certainty with regards to the hypothesis. DST is fundamentally different from Bayesian reasoning because belief in a hypotheiss and its negation need not sum to 1; in fact, both values can be 0 (meaning that there is no evidence either for or against the hypothesis).

Let $\Theta = \{\theta_1, \theta_2, ..\theta_n\}$ be the set of possible conclusions to be drawn, then the θ_is are mutually exclusive and Θ is exhaustive. The goal of DST in our context is to

predict the state of the system as a function of time. More precisely, let the set of known system states, for a particular class of attacks (e.g., DoS or SQL injection), be $SS = \{sa_1, sa_2, sa_3...sa_n, sn\}$ where the state $sa_i, 1 \leq i \leq n$ represents the state of the system when it is under attack a_i and state sn represents the state of system when it is operating normally. DST assumes that the current state of the system at time t , CS(t), is unknown but must be one of the values from the *Frame of Discernment*, SS. Note that each state in SS is observable.

CS(t) is determined by providing and combining *Basic Belief Assignments (BBAs)*. If SS is the frame of discernment then a function $m : 2^{SS} \rightarrow [0, 1]$ is called basic belief assignment if the following conditions hold:

$$m(\emptyset) = 0, \qquad \sum_{A \subset 2^{SS}} m(A) = 1.$$

The term m(A) is called A's BBA and is a measure of the ***belief*** that is committed to exactly A. In DST, the notions of *belief* and *plausibility* are used to reason about the certainty (or lack of) in the system being in a particular state. The belief function, *Bel*, is a mapping $Bel : 2^{SS} \rightarrow [0, 1]$ and is given by:

$$Bel(A) = \sum_{B \subset A} m(B).$$

The Plausibility Pl is a mapping $Pl : 2^{SS} \rightarrow [0, 1]$ and is given by:

$$Pl(A) = \sum_{B \cap A \neq 0} m(B).$$

As should be evident, the belief and plausibility with DST serve as upper and lower bounds on the degree of certainty of the system being in a particular state. The rest of this section will describe how we use system wide features (or observables) and DST to compute belief and plausibility about system states.

Observables: An observable in our context is simply anything that can be sampled or monitored to produce a time series. Relevant examples of observables are the number of total bytes received over a network interface and CPU consumption expressed as a percentage. We use the terms observables and features interchangeably. The observables might be important indicators of the presence or the absence of an attack. For example, one might expect the total number of bytes received over a network interface to be very high when the system is under a flooding based DoS attack. More specifically, the set $O = \{o_1, o_2, o_3...o_n\}$ is the set of observables (or features) used to detect attacks. A better O will result in higher detection rates.

An observable, depending on its current value, provides a belief over SS that provides a hypothesis on the CS. As an example, consider the number of SYN packets received in a given time window, denoted by A. As is well established, a large number of TCP SYN packets in a small time frame is highly indicative of a SYN flood attack[107]. When the value of A crosses a predefined threshold, it provides a belief in a SYN flood attack denoted by $B_A(SYN) = 0.8$ (say). The remainder of A's belief is always allocated to the frame of discernment specified as $B_A(SS) = 0.2$.

Combining Evidence from different sources: A core feature of DST is combining beliefs from independent sources of evidence. In our particular context, this can be

thought of as observing two different features (or observables), say A and B. When A and B exceed their pre-defined thresholds they provide beliefs in attack x given by $B_A(x)$ and $B_B(x)$ respectively. Combining different features results in a unified belief over a particular state in SS. Because we have evidence from multiple sources, our case is not as simple as using a single fusion operator (fusion operators are designed to work on exclusively dependent or exclusively independent beliefs). Thus, we use the averaging and the cumulative fusion operators (details in [134] and [185], respectively). The averaging operator is intended to be used with dependent beliefs and the cumulative operator is intended to be used with independent beliefs. We first use the averaging operator to combine subsets of dependent beliefs and then use the cumulative operator to combine the resulting independent beliefs. (dependent beliefs can be thought of as those belonging to the same sensor).

The Averaging function is given by:

$$\begin{aligned} AVG(x) &= \frac{B_A(x)B_B(SS) + B_B(x)B_A(SS)}{B_A(ss) + B_B(SS)} \\ AVG(SS) &= \frac{2B_A(SS)B_B(SS)}{B_A(SS) + B_B(SS)} \end{aligned} \tag{3.1}$$

The cumulative function is given by:

$$\begin{aligned} CUM(x) &= \frac{B_A(x)B_B(SS) + B_B(x)B_A(SS)}{B_A(SS) + B_B(SS) - B_A(SS)B_B(SS)} \\ CUM(SS) &= \frac{2B_A(SS)B_B(SS)}{B_A(SS) + B_B(SS) - B_A(SS)B_B(SS)} \end{aligned} \tag{3.2}$$

Belief and Plausibility: The application of the Dempster-Shafer framework results in belief and plausibility values for each attack. These allow us to reason about the usefulness of a particular subset of features in terms of providing high detection accuracy.

3.4.2 Clustering Algorithm

Clustering algorithms are one of the most popular classes of machine learning (ML) algorithms that are used for anomaly detection [108] [71]. Generally speaking, clustering is a technique for finding patterns in unlabeled data. We use the K-Means clustering algorithm because it is one of the simplest and relatively efficient of clustering algorithms which has been successfully used for anomaly detection.

It works by grouping similar objects into K disjoint clusters. We will only provide a high level overview of how the algorithm works here. Details can be found in [162] which demonstrates how to apply the algorithm for intrusion detection. Fundamentally however, the algorithm is built around the notion of a centroid.The centroid of a cluster is a point in the feature space that can be thought of as the most representative point for that cluster. Once the centroid for each of the K clusters is known, the algorithm simply compares each new instance to each of the K centroids to determine which one it is closest to. The algorithm has two phases, an offline clustering (training) phase, and an online classification phase.

Clustering: During clustering the goal is to train the system. Put another way, the goal is to determine the optimal centroids. The algorithm is fed a set of instances (which in our case are defined by the output of the feature selection algorithms). Clustering then is a five step process.

1. Initiate the number of clusters, K to some user defined value. In our case, K is the number of elements in SS, which corresponds to the number of attacks and normal behavior.

2. Initiate the K cluster centroids. This is typically done by arbitrarily choosing K data points from the set of training data.

3. Iterate over all training objects and compute the distance of each object to the centroids. Assign each object to the cluster with the nearest centroid.

4. recalculate centroids (ensuring that a previously chosen point is not chosen again)

5. Repeat step 3 until assignment of objects is static. i.e, between two different iteration with different centroids, the assignment of objects to clusters remains the same

For the distance measure, we use the Euclidean distance. Its effectiveness was demonstrated in[162]. At the end of the clustering phase we get K centroids that will be used as future references.

Classification: During classification (done online) each new data point is simply compared against all the previously computed K centroids and assigned to the one it is closest to. As in the case of our DST approach,, we perform this entire process seperately for DoS and SQL injection.

3.5 Experimental Setup

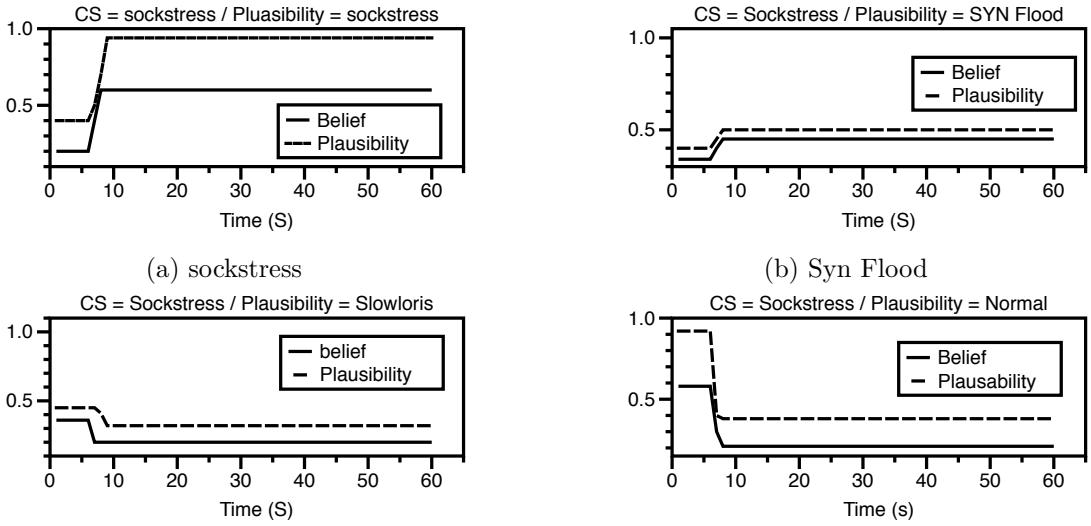
Testbed: We perform our experiments on the The Cyber Virtual Assured Network (CyberVAN) testbed [171]. The CyberVAN is a state of the art cyber security testbed that was designed to support experimentation in a virtual cyberspace. CyberVAN models hosts as full fledged virtual machines and models the underlying network using discrete event network simulation. The testbed can be used to model a realistic cyber network environment

with high fidelity. Virtual machines that act as end hosts are time synchronized with the discrete event network simulator. This enables CyberVAN to slow down VM time if the simulated network cannot keep up with real time such as in the case of high volume traffic.

Network Topology: Our network topology consists of one server machine, client machines from 50 different subnets with IP addresses widely distributed across the public Internet, and one attacker machine. The server has *CentOS* server version 6.6 and *Apache* version 2.2. The server hosts 20 different websites, each with a complex navigational structure. The client machines are all running *Ubuntu* version 14.04. Each client machine has a synthetic user that interacts with the server using a FireFox web browser. The clients run in one of two different modes, web-only or database-only.

The normal traffic (which keeps flowing regardless of whether or not an attack is under way) is generated based on the patterns of real users on the Internet as discussed in [104]. The synthetic users are emulated to use actual applications. For web-only applications the users interact with static web content using a Firefox web browser and for database-only, they interact with a Wordpress blog. The eventual traffic contains realistic short and long term HTTP and TCP connections.

Methodology: We consider two classes of attacks to validate our framework, viz.. Denial of Service (DoS) and SQL injection. Since we expect these two classes of attacks to manifest different (and often disjoint) kinds of symptoms, we experiment with only one class of attack at a time. Specifically, we first collect features explicitly for DoS attacks via



(a) sockstress

(b) Syn Flood

(c) Slowloris

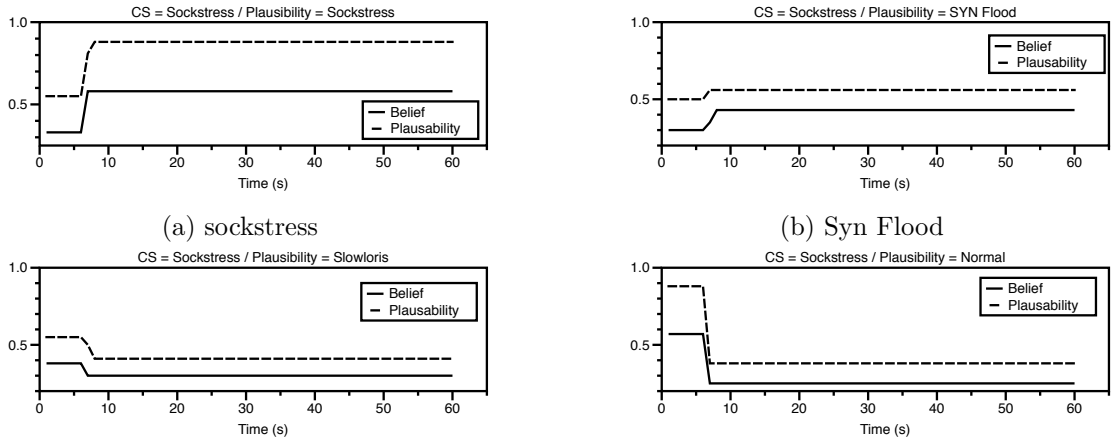
(d) Normal Traffic

Figure 3.2: Belief and plausibility during a Sockstress attack with features selected by LFS

the data collection approach outlined above. We test out three different TCP based DoS attacks. SYN Floods [107], Sockstress [32] and Slowloris [30].

For SQL injection we test two different attacks. The first attack exploits a Wordpress vulnerability wherein a specially crafted SQL query results in the entire database being returned as the response. The second attacks exploits another vulnerability wherein a specially crafted string opens up a shell and gives the attacker root access. SYN flooding is one of the oldest forms of DoS attacks and operates by flooding a victim with TCP SYN packets. Sockstress, is more complicated in that it actually completes the TCP handshake in an attempt to exhaust all sockets. Slowloris is an HTTP attack that opens numerous HTTP connections within a time and then periodically sends keep alive messages to hold them.

To test our approach against a previously unobserved attack, we build such an attack. Specifically, we employ the methodology described in [68] to intelligently combine a



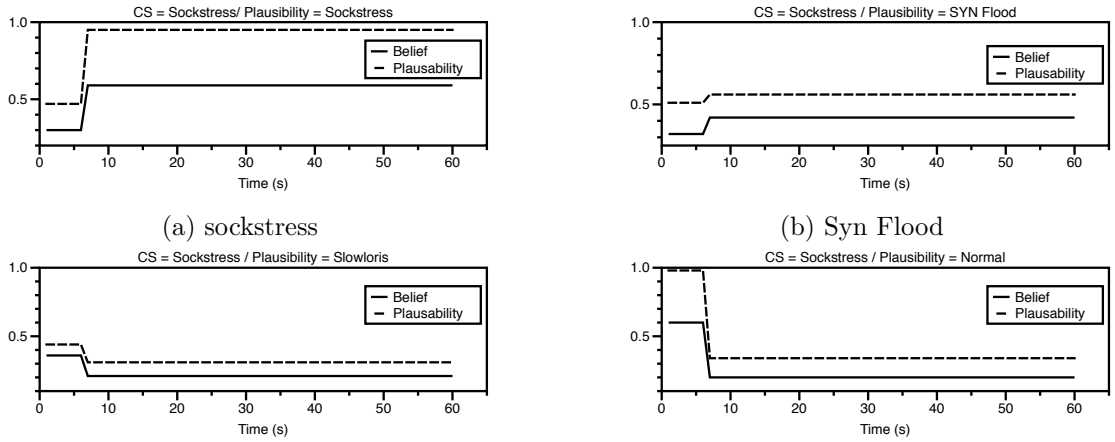
(a) sockstress (b) Syn Flood

(c) Slowloris (d) Normal Traffic

Figure 3.3: Belief and plausibility during a Sockstress Attack with features selected by SBS

SYN Flood and Slowloris attack. Each component of the attack, by itself, does no discernible damage and thus, cannot be detected by traditional detection approaches; but together the components consume ports and thus, constitute a powerful but stealthy DoS attack.

We monitor a total of 365 features for each state in SS. Each state is monitored for a minute after which we are left with a time series of each feature. This data is then labeled according to the state it belongs to and is then fed into our feature selection engine. Figure 3.1 depicts how the entire process works to produce a set of features. Once those features are selected they are fed into each inference engine seperately. For DST, we examine the generated time series and assign thresholds which, when triggered, output a pre-assigned belief in an attack. Taken as such these observables are just binary belief functions. For example, one obvious feature for detecting SYN floods is the number of SYN packets received. We assign equal beliefs to all features regardless of perceived importance. A better assignment of these beliefs can yield better detection accuracy but is out of the scope of this work. For K-Means, we train the system as described above setting the value



(a) sockstress (b) Syn Flood

(c) Slowloris (d) Normal Traffic

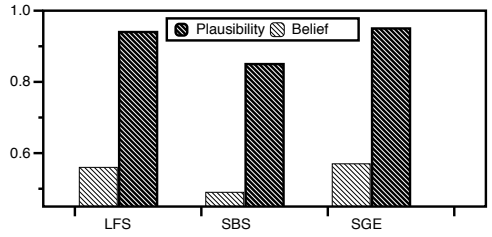
Figure 3.4: Belief and plausibility during a Sockstress Attack with features selected by SGA

	DoS	**SQL Injection**
LFS	Context Switches, Sys calls, Free Memory, HTTP Connections	Established Connections, DB Errors, Bytes Sent
SGE	Sys Interrupts, Sys Calls, Used Memory, HTTP Connections	DB Errors, Bytes Sent, DB lookups
SBS	Page Faults, CPU1 Utilization, Established TCP connections, Swap Space	CPU1 Utilization, Bytes Sent, Page Faults

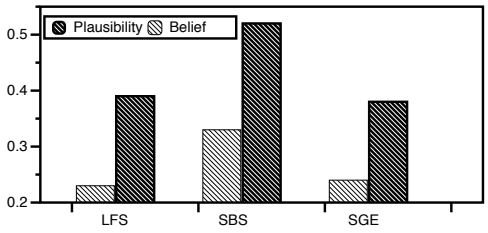
Table 3.2: A subset of features selected by each algorithm

of K to the cardinality of SS (the number of clusters is equal to the number of attacks plus one for normal behavior)

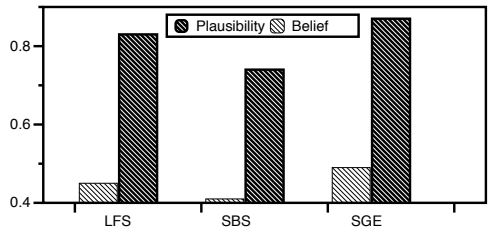
During runtime, the set of features that were output by the different algorithms (via offline training) are monitored. The attacks that were previously described are launched in real time. For DST, the observations (of the features monitored) are input into equations (3.1) and (3.2), and we obtain measures of the belief and the plausibility for each candidate scenario (different attacks and normal behavior). For K-means we simply monitor whether each attack (or normal behavior) is correctly classified.



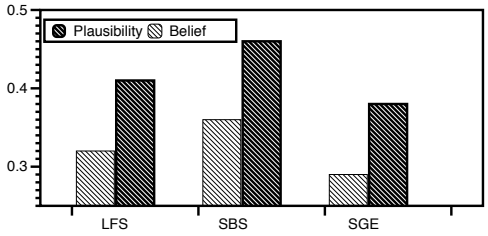
(a) DoS: Current State is the correct one



(b) DoS: Current State is the wrong one



(c) SQL Injection: Current State is the correct one



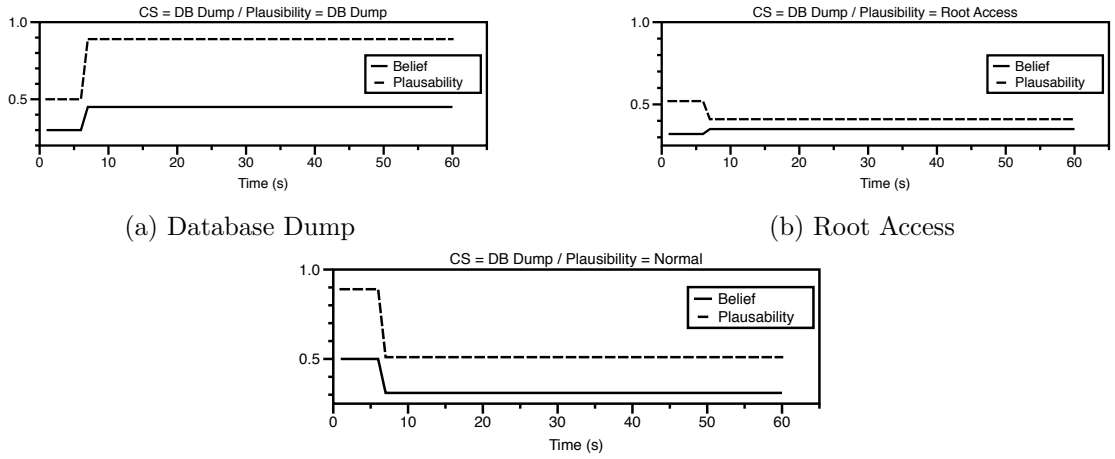
(d) SQL Injection: Current State is the wrong one

Figure 3.5: Average Belief and plausibility for DoS and SQL Injection

3.6 Experimental Evaluations

In this section we discuss the results from our experiments. For each algorithm, we are interested in evaluating how well the features that it selects are able to differentiate between the different kinds of attacks and normal behavior. Table 3.2 lists a small subset of the features (due to space constraints) that were selected by each algorithm. The selected features, due to the nature of modern systems, will be highly dependent on the execution environment. The table highlights the fact that LFS and SGE tend to select similar features while SBS does not. We will first present results using the DST inference engine, and later using the K-Means inference engine.

Detection using DST: Figures 3.2, 3.3, and 3.4 show how well the features chosen by LFS, SBS, and SGA, respectively, perform when a Sockstress attack is initiated. The figures shows how the belief and the plausibility vary over the time period of the



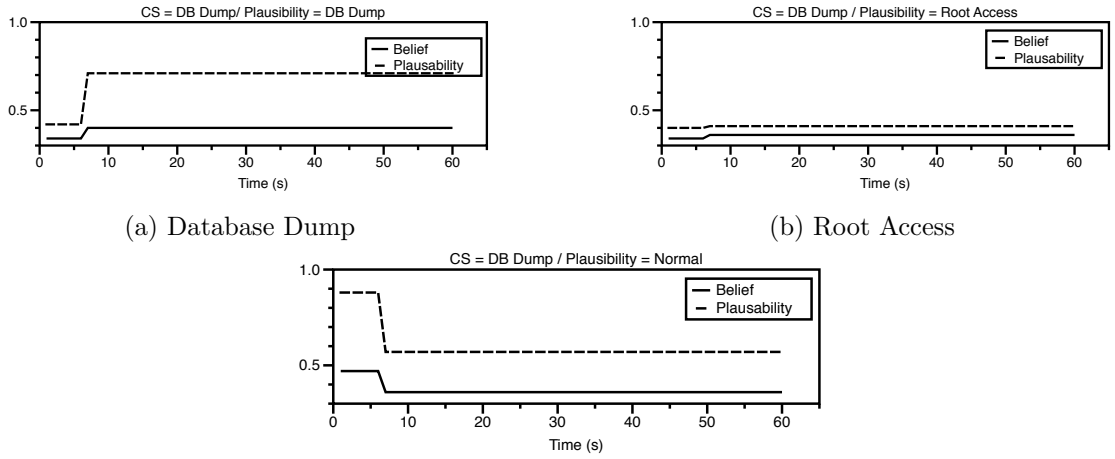
(a) Database Dump

(b) Root Access

(c) Normal Traffic

Figure 3.6: Belief and plausibility during a Database Dump Attack with features selected by LFS

attack. The attack is initiated at time 6 seconds. Figures 3.2a, 3.3a and 3.4a show how both the plausibility and the belief increase, indicating a strong conviction in the present state (sockstress). The other figures show that there are low values of belief and plausibility with respect to the other possible scenarios (e.g., normal behavior) indicating that DST has very little conviction that the current state corresponds to one of these. Figures 3.2d, 3.3d and 3.4d show how the belief and the plausibility in the current state being "normal" plummets once the attack is launched. The slight increase in belief and plausibility exhibited in Figures 3.2b, 3.3b, and 3.4b is due to the fact that SYN Flood and sockstress are similar kinds of attacks. For example, both involve large volumes of traffic and thus, sometimes trigger the same sensors. The interesting observation from these figures is that the detection accuracy (belief and plausibility) with LFS is comparable to that of SBS (they are within 2% of each other); however, SBS lags visibly. The belief and the plausibility results with all the algorithms, for each state, exhibit the same behaviors observed in Fig 3.2. For the



(a) Database Dump

(b) Root Access

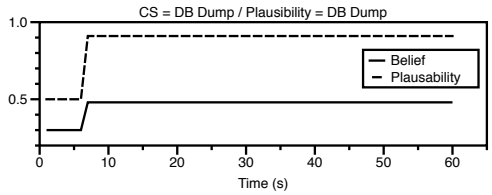
(c) Normal Traffic

Figure 3.7: Belief and plausibility during a Database Dump Attack with features selected by LBS

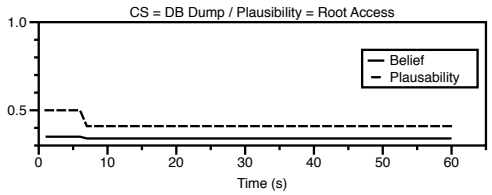
current "true" state these metrics increase and generally decrease otherwise. We omit the results for the SYN flood attack due to space limitations but they exhibit similar patterns of results.

In Figures 3.5a and 3.5b, we summarize the plausibility and belief results for DoS with the different algorithms. Specifically, we show the average value of these metrics observed (over all cases) with regards to the true or "correct" state (e.g., the belief and plausibility with Sockstress when it is actually in effect) and the "wrong" state (e.g., the belief and plausibility with Sockstress when Slowloris is in effect). We see that the values of these metrics are much higher with the correct case (low values are exhibited for wrong cases).

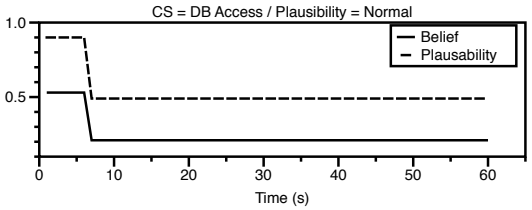
SQL injection results for the database dump attack, are detailed in Figures 3.6, 3.7 and 3.8. They exhibit behaviors similar to what was observed with DoS. However, across all the three algorithms, the difference is that the belief and the plausibility results are not



(a) Database Dump



(b) Root Access



(c) Normal Traffic

Figure 3.8: Belief and plausibility during a Database Dump Attack with features selected by SGA

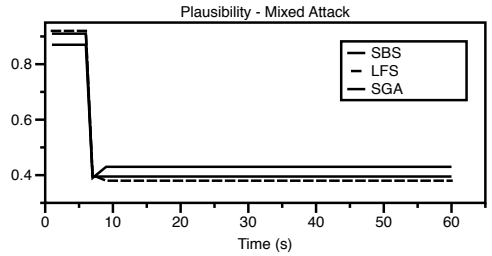


Figure 3.9: Plausibility for a mixed attack

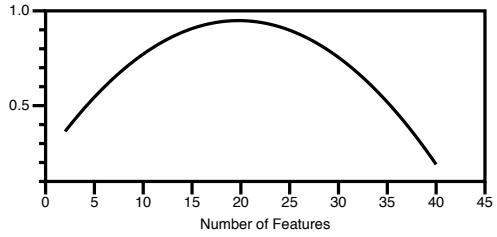


Figure 3.10: Change in plausibility for DoS

as high (compared to DoS) in the "correct" state, and not as low with the wrong "state". We believe this is because a lot of good features for SQL injection are in fact embedded in SQL queries which we do not consider in this work (this is left for the future).

The results with SQL injection are summarized in Figures 3.5c and 3.5d. Again, we see that the performance with SGA and that with LFS are still close (always within 10 % of each other). However, here SGA gives an approximately 6 % performance improvement over

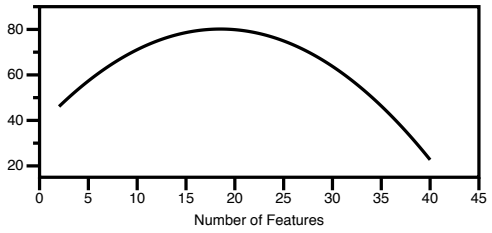


Figure 3.11: Change in accuracy for DoS (K-means)

	SYN	**Sock**	**Slowloris**	**Normal**
LFS	86	88	83	85
SGE	86	87	85	83
SBS	75	72	70	77

Table 3.3: Accuracy of k-means classifier under DoS attacks

	Root	**DB Dump**	**Normal**
LFS	74	77	75
SGE	75	78	74
SBS	62	65	66

Table 3.4: Accuracy of k-means classifier under SQL injection attack

LFS consistently, thus demonstrating that higher complexity could yield better accuracy. Both LFS and SGE outperform SBS again.

Detection using K-Means clustering:

To evaluate the features when K-Means is used for inference we use accuracy (percentage of samples that are correctly classified) as our metric. Tables 3.3 and 3.4 detail the results. For both SQL injection and DoS we see that the performance of LFS is comparable to SGE. As with DST we observe that we get better accuracy in detecting DoS attacks. This is due to the fact that a lot of good features for SQL injection are in the actual query accessible via DPI which we do not consider. We also observe that the performance of SBS lags similarly to the other two. However with K-Means, the discrepancy is much greater. This is greater testament to how completely SBS is outperformed. Some

LFS	SBS	SGA
2.9	10.8	65

Table 3.5: Average completion time in minutes

of the weaknesses inherent in SBS (for e.g. it cannot reexamine features that have been eliminated) are masked by DST because DST deals very well with noisy features.

Comparing algorithms: A comparison of performance across the different algorithms yields some interesting insights. First both LFS and SGE yield consistently better results than SBS. The fact that LFS outperforms SBS so consistently indicates that the ideal subset of features is small. SBS is known to underperform in such cases. LFS is also much quicker than SBS.

As mentioned previously, we expect SGA to perform well because of its resistance to getting caught in local maxima and minima. It also takes the most time to finish. However, in some cases we find that LFS does slightly better than SGA (e.g., see Figures 3.2a and 3.4a). The difference however, is always small (results are always within 6 % of each other) . These results suggest that local maxima and minima are unlikely and the use of LFS (much faster) can suffice for highly accurate detection.

Using more or less features than what is recommended by the algorithms: Using too few or too many features can hurt detection performance. To demonstrate this, we configured LFS to output multiple sized feature subsets. (i.e., the best 2, the best 4 the best 6 and so on). We then tested these feature sets with all the three DoS attacks considered and computed the average plausibility of predicting the correct state with DST and the average classification accuracy with K-means. The results are shown in Figures 3.10 and 3.11. We see that with DoS, the optimal number of features is somewhere

between 15-20 for DST and between 11-16 for K-means. A higher set could lead to wrong conclusions; a smaller set could reduce detection accuracy.

Mutated attack: Finally, we are interested in evaluating how well the chosen features hold up to an unknown attack. To do this, we launch the mixed attack that was described previously in Section 3.5. Figure 3.9 shows how the plausibility that the current state is normal drops in the presence of a mixed attack (signifying a high likelihood of an attack). LFS again outperforms the other two (SGA is still very close). K-means classifies the mixed attack as either one of the two mixes (depending on the dominant attack)

3.7 Related Work

There has been a lot of work done on feature selection in the domain of anomaly detection. However, most prior efforts (unlike ours) are tied to specific classification approaches [138], [164], [195]) They are also typically only concerned with network layer features. In [197], the authors evaluate various selection techniques, including genetic algorithms in the context of intrusion detection. However they are only concerned with features that originate from the network layer. Their approach is limited because their initial feature set (on which they apply feature selection) is itself manually selected. In [192] the authors develop a decision tree based genetic algorithm. They use decision trees to guage the performance of their algorithm which is not immune to noise or uncertainty. In [91] the authors compare and contrast two well known feature selection algorithms. They employ the DARPA intrusion detection data set [12] which only contains network; cross layer

features are not considered. [160], [170] and [195] are other examples which only consider network layer features for intrusion detection.

Other approaches such as [160] are concerned with extracting features so that classification accuracy is not hurt. This is fundamentally different from what we are trying to do because we are concerned with large data sets with potentially frivolous features.

Chapter 4

Jaal: Towards Network Intrusion Detection at ISP Scale

4.1 Introduction

Incorporating cybersecurity capabilities has become integral to networked system design today. However, there are continuing challenges having to deal with very large scale and complex attacks, and the ever-changing nature of attacks. In the recent past, we have seen an alarming increase in network based high-profile security breaches [39, 43, 86].

Today, ISPs predominantly focus on volumetric (e.g., DDoS) attacks by gathering traffic at ingress gateways and analyzing it at centralized scrubbers [37]. These services are delivered to an enterprise by an ISP on demand, i.e., a customer requests the services when it determines that its network is under attack or suspects an attack. The process is to copy packets to and from the enterprise at gateways, and forwarding these to a central Network

Intrusion Detection System (NIDS). Unfortunately, the approach of transferring copies of raw packets continuously towards detecting attacks suffers from scalability problems.

In this chapter, our goal is to design an efficient ISP-scale NIDS, that has the following properties: (a) it should detect a wide variety of attacks as with smaller scale IDSs such as Snort or Bro and, (b) it should not require the copy and transfer of raw packets to a central inference engine.

The key design principle that we follow is to "extract" the requisite information from a packet stream at monitoring points spread through out the network. Specifically, the monitors, (could be co-located with routers/gateways or placed at IXPs) process packets and create lightweight in-network *packet summaries* of drastically smaller volume compared to raw packets. These summaries can be used to draw inferences using rules similar to those used in smaller scale NIDS (e.g., Snort). They are sent by monitors to a central inference engine which processes them and issues alerts when attacks are detected. In rare cases when a highly accurate inference cannot be made with the summaries, the inference engine queries appropriate monitors (which store packets for short periods) for either finer grained summaries or associated raw packet traces for a time window of interest.

In designing an ISP-scale NIDS based on the above principle, we will need to address the following challenges. (a) How do we construct lightweight packet summaries while still achieving high detection accuracy? (b) How do we transform typical NIDS rules (e.g., that of a system like Snort) to find attack patterns/signatures using these summaries in lieu of raw packets? and, (c) Under what conditions should the system retrieve finer-grained summaries or raw packet traces to ensure a high accuracy of detection while keeping the

overhead low? In this chapter we design and implement an ISP-scale NIDS, *Jaal*, that addresses these challenges. Specifically, in designing *Jaal*, we make the following contributions:

- We design a novel algorithm that uses dimensionality reduction techniques to construct concise packet summaries that lend themselves to highly accurate network intrusion detection.
- We design an approach to transform a large set of IDS rules (specifically from Snort) to a new equivalent representation that can be applied to the generated summaries in lieu of the actual packets. We propose simple new, equivalent rules for those that cannot be automatically transformed.
- We implement *Jaal*, on a large scale SDN testbed that allows us to create complicated ISP-scale topologies, using network function virtualization (NFV).
- We evaluate the performance of *Jaal* using realistic ISP traces [52] and with a wide range of popular attacks. Our results show that with about 65% reduction in communication overhead, *Jaal* can achieve a detection accuracy of ≈98% with respect to these attacks which, we believe is the highest reported at these scales.

4.2 Synopsis

There are many recent alarming reports of large-scale distributed attacks targeting multiple data centers or enterprise networks simultaneously [40, 42, 47]. Unfortunately,

today such attacks are only detected much after the fact, and the targeted entities report them individually.

The Mirai botnet example: To exemplify the problem, consider the example of the DDoS attack caused by the Mirai botnet, which targeted Dyn's DNS infrastructure [40]. The attackers used compromised IoT devices (e.g., printers, cameras and home routers) spread across the Internet to launch one of the largest known DDoS attacks crippling various services across the Internet including Twitter, Airbnb, Github and Amazon, among others. In brief, a post-attack analysis revealed that the IoT devices were infected using a simple two-step process [46, 45]. First, such devices were discovered by continuously scanning IP addresses across the Internet for open ports. Second, once an open port was found the associated device was compromised, if it was vulnerable, using a short list of hard coded passwords (most devices were still using default username/password combinations), to gain root access. We analyzed the source code for Mirai (available publicly at [41]) and found that the scanning was primarily directed at destination ports 23 and 2323 (*File*: mirai/bot/scanner.c *Lines:* 117, 219, 223 [41]). A device, once subsumed into the Mirai botnet repeated the exact scanning activity mentioned above [45]. Note here that this scanning was only discovered after the attack had been launched and researchers had analyzed the source code publicly dumped by the attackers.

Need for ISP Scale detection. Such scanning, while simple to carry out, is inherently difficult to detect using current detection capabilities for two reasons. First, most IoT devices are used in homes where consumers typically do not use IDS systems. So both the incoming scan that is looking for open ports and the outgoing scan launched by the

device once it is infected, are missed. Second, and perhaps more importantly, even if we consider networks with some intrusion detection capability, the global scope of this scanning activity is only observable via a holistic view of a wide area (ISP-scale) network. This is because, from the perspective of a smaller scale (e.g., enterprise) network with detection capabilities such as those of Snort, a simple scan directed at two TCP ports for a small range of IP addresses (that are observed) is not a serious concern and would not likely trigger port scan alerts. Port scan alerts are only issued if a large volume of packets with a large set of different incoming ports are seen [58]; this was not the case with the Mirai scan which only scanned for two ports but across a large set of devices that were spread across a large number of administrative domains.

One could argue that a scan that targets the same two port numbers across an extremely large gamut of IP addresses (almost the entire IPv4 address range [45]) is concerning and should trigger alerts; however, this characteristic is only evident if the ISP-level traffic is analyzed at a detection engine. With such holistic visibility, a detection system could have identified infected devices long before the DDoS attack was launched (such scanning activity can be potentially detected in seconds as we show later in § 6.5).

The need for holistic ISP-scale detection capabilities have recently been widely advocated. For example, in response to the attack on Dyn, the security expert Bruce Schneier says that "DDoS prevention works best deep in the network, where the pipes are the largest and the *capability to identify and block the attacks is the most evident*" [44]. The fact that current detection frameworks and techniques lack the capabilities to detect coordinated attacks distributed across multiple organizations (such as the Mirai attack)

has also been highlighted by DARPA [50], which exemplifies the urgent need for NIDS that address this shortcoming.

While ISP networks have the global visibility required to detect such attacks, the technical challenges in realizing WAN-scale detection are yet to be overcome. The popularity of open source IDS's like Snort and Bro has proven how effective pattern matching is in detecting network-based attacks. However, using such methods directly in WANs is hard (as discussed later). While enterprises typically have a single entry point where an IDS such as Snort or Bro can be employed, WANs usually have multiple points of entry and egress. This means that no single location in the network can view (monitor) all the traffic. Given that multiple vantage points (or monitors) are needed to completely cover all traffic, the information collected (or generated) by these monitors needs to be "aggregated" to create a global view for analysis.

Challenges: There are various approaches that one could take to create such a global view. The first and possibly the most obvious approach would be for each monitor[1] in the network to forward copies of traversing packets to a central analysis engine. We tested the feasibility of this approach (details in § 6.5; see Fig. 4.7) and found that it causes a 70% loss in throughput (average rate at which packets that belong to normal traffic are proceesed at each router) and a 75% loss in detection accuracy (the fraction of correctly classified attacks out of all attacks).

We point out here that it is well known that modern open source DPI-based IDSs cannot cope well with high traffic volumes [85, 137]. In fact, they do not fail gracefully and

[1]The monitors can be realized in several ways. They could be implemented using a core router functionality (like Cisco's NetFlow [109]). The monitors can also be dedicated machines deployed at IXPs.

with traffic rates greater than 20 Gbps, packet losses of over 50% are experienced regularly; this can seriously affect the accuracy in detecting attacks (can result in approximately a 50% loss in detection accuracy). The only way around the loss in accuracy is to provision IDS clusters for peak load which can result very high costs and even with that, a large wastage of computation resources at off-peak times.

Current methods such as sampling/sketching are inadequate. One may argue that the heavy workloads due to copying raw packets, can be overcome by using state of the art packet sampling techniques [83]. In fact, ISP's typically employ rudimentary sampling techniques like NetFlow [109] to obtain a coarse view of network dynamics. However, while sampling can help in heavy-hitter detection, it results in poor accuracy with respect to fine-grained features needed for detecting a wide variety of attacks [173, 152], especially in cases where a information with respect to a large number of successive packets is required (e.g., DDoS). In particular, sampling fails to capture correlations across packets.

An alternative technique to sampling is sketching [141]. While sketching provides strong resource/accuracy guarantees, it is inherently a targeted measurement scheme in which one needs to construct a sketch for every measurement task (e.g., counting the number of unique IP addresses or measuring the entropy of IP addresses in a batch of packets). Unlike the summarization techniques we develop in *Jaal*, modern sketching techniques are also restricted to single dimensional measurements [148]. This lack of generality implies that sketching does not scale. The sketching technique in [148], which is arguably the most general sketch possible today, allows a single sketch to be used for multiple measurement tasks. However, it is still limited to a single dimension.

More concretely, consider the source IP address as the dimension of interest. By using the technique in [148], one can create a sketch that captures the number of unique addresses seen. This sketch can be used to detect heavy hitters, membership testing and entropy estimation [148]. However, even this sketch will only be able to answer queries about the source IP address. Intrusion detection signatures require correlations across packet dimensions. A simple example is a distributed SYN flood attack. To keep track of the source IP and the SYN flag, one would need a sketch that tracks these two fields. However, such a sketch can now no longer be used to answer queries about the IP or SYN flag alone. There is no way to decouple those two header fields in the sketch. Thus, to create sketches to detect attacks based on any combination of TCP/IP headers (18 header fields), a need a total of 2^{18} individual count-min sketches at every monitor. Assuming each count-min sketch to be 500KB [148], this would translate to 128GB of information being transferred by each monitor to a central location per measurement epoch, leading to a prohibitively large communication cost. In addition, such a brute-force combinatorial sketching method would not leverage correlations between header fields which may not be known to us a-priori and would only emerge through the actual traffic generated. Our proposed summarization scheme is able to identify and leverage those correlations automatically, by virtue of low-rank approximation.

Key idea: We envision that by extracting lightweight in-network "packet summaries" that retain the information needed by a NIDS, from traffic that flows through monitors, we can solve the challenge of scale while retaining performance and accuracy. This is in essence the key idea in building our framework *Jaal*.

Threat Model: While *Jaal* can handle all attacks that a NIDS like Snort can handle, we limit the scope of our evaluations to only transport layer attacks. This is because, together they constitute the most widely seen attacks in the wild [39]. We omit attacks that will require us to examine the payload given that these days payloads are often encrypted.

An IDS such as Snort also handles signatures relating to lower-layer attacks such as ARP scans. Since such attacks are local (link level and thus do not require global knowledge), we argue that they can be detected using a local "lower layer attack detector" in each local area network (LAN) independently. Thus, we do not try to detect these with *Jaal*.

We assume that monitors have already been placed in the network and their locations are static. Flexible monitor placement and management is beyond the scope of this work.

4.3 System Overview

Our goal is to build a NIDS at ISP scale, with capabilities similar to that of today's modern NIDS (e.g., Snort [175] and Bro [168]) deployed in enterprises, based on the key idea described in § 4.2. Evidence collected in the network and transferred to a detection engine must consist of concise yet informative summaries (instead of raw packets), and the detection engine must be able to process these summaries and provide inferences just as with Snort. One of the driving principles behind designing Jaal was generality. We aim for the techniques we develop to be equally amenable to detecting all attacks. Therefore, we make no assumption on the utility of any packet header field and treat all header fields

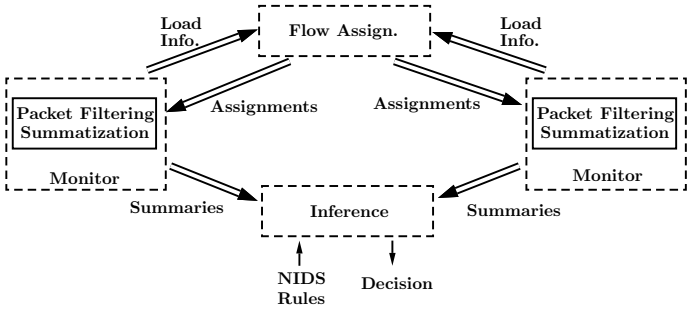


Figure 4.1: *Jaal* architecture.

as equally important. We follow modern IDS systems design which provide pipelines for querying any header field. We choose Snort as our baseline since it is the most popular IDS used today [31].

We meet this goal by designing a modular ISP scale NIDS *Jaal*, which comprises three modules viz., summarization, inference, and flow assignment (see Fig. 4.1).

Summarization module: This module runs on monitors in the network. Each monitor extracts headers of traversing packets and constructs a "summary" of these headers using a lower dimensional representation. The summary is then forwarded to the analysis and inference module.

Inference module: *Jaal* contains a centrally located inference module which, given a set of Snort-like, signature-based rules, transforms the rules into a format suitable for use with the summaries. Equivalent rules are proposed (again applied on summaries) for detecting attacks that cannot be be detected via simple comparisons with packet signatures. Packet summaries collected from monitors are compared with these new rules to detect threats.

Flow assignment module: The flow assignment module seeks to assign flows to monitors such that each flow is monitored only once and the maximum load across the monitors is minimized. This is important both for correct operation of the NIDS as well as for saving bandwidth. The problem is mapped onto a constrained load balancing problem and solved using a simple yet effective approximation algorithm.

A detailed description of each module in *Jaal* follows in each of the next three sections.

4.4 Packet Summarization

The goal of packet summarization is to produce a representative summary of a batch of packets that: (a) enables the analysis and detection of a wide class of attacks, and (b) allows *Jaal* to achieve high detection accuracy with low communication overhead.

Henceforth, we refer to the packets and the packet fields as the different *modes* of the data that we are summarizing. The number of dimensions in those modes (i.e., the number of packets that have passed through a monitor and the number of fields in each packet) is large; thus our goal is to reduce the dimension of both modes while preserving correlations between the dimensions of both modalities.

To support our goal, we employ *dimensionality reduction*, a family of techniques that approximate a dataset with high-dimensional modes, using a modified dataset with significantly reduced dimensions, while minimizing the approximation error. This reduction results in a more compact, yet high fidelity data representation which respects correlations in both modes of the data. We propose a practical two-step approach where each step

focuses on efficiently reducing the dimensionality of one of the two modes of the data. At the end of the process, we create what we call in-network packet summaries. One may envision a single-step approach to reduce both modes simultaneously; however this objective is computationally hard from an optimization point of view. Our approach can achieve any desired accuracy, by trading off communication cost. Specifically, the design parameters, determining the level of reduction, control the tradeoffs between the costs (i.e., the size of summaries sent to the inference module) and the detection accuracy.

4.4.1 Packet filtering and Normalization

We assume that there are monitors in the network to aid intrusion detection. Each such monitor filters and processes packets from flows (specified by a four tuple viz., source and destination IP addresses and port numbers) that are assigned to it (assignment of flows to monitors is done by a central engine; this is discussed in § 4.6). Transport and network layer headers are then buffered until the number of packets in the buffer, regardless of which flows they belong to, is equal to some pre-determined threshold n. We call this a batch of size n. Each batch of packet headers is organized in a matrix $\mathbf{X}$ with dimensions $n \times p$. Each row represents the p fields in the (TCP and IP) headers of a given packet.

Before we construct packet-summaries, we create a normalized version of $\mathbf{X}$ denoted by $\bar{\mathbf{X}}$. In *Jaal*, packet header fields are processed as vectors, and each header field is mapped to an entry in the vector. A measure of distance is used to decide whether two packets are similar. Since the raw magnitudes of the header fields values vary greatly, normalization of these values is needed to ensure that there is no bias towards fields that vary over a larger range. For example, consider a vector with only the TCP SYN flag and the

source IP address fields. Without normalization, the distance computations will be heavily dominated by variations in the IP address field. Thus, for any header field with value $x \geq 0$, we apply the transformation $\bar{x} = \frac{x}{\max(x)}$, where $\max(x)$ is the maximum possible value for that header field (x). Thus, we have $0 \leq \bar{x} \leq 1, \forall x$.

4.4.2 Dimensionality reduction: fields mode

We first apply Singular Value Decomposition (SVD), a dimensionality reduction technique, on the packet fields in $\bar{\mathbf{X}}$ to reduce its rank. By reducing the rank, we form a smaller-size representation of $\bar{\mathbf{X}}$ which provably approximates the original matrix well [133]. First, $\bar{\mathbf{X}}$ is decomposed to

$$\bar{\mathbf{X}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T, \tag{4.1}$$

where the columns of $\mathbf{U}$ are the left singular values of $\bar{\mathbf{X}}$, columns of $\mathbf{V}$ are the principle axes (or directions) of data in $\bar{\mathbf{X}}$, and $\mathbf{V}^T$ is the transpose of $\mathbf{V}$. $\mathbf{\Sigma}$ is a diagonal matrix with the singular values σ_i of $\mathbf{X}$ in descending order. The number of non-zero singular values is the rank of $\bar{\mathbf{X}}$, and the sum of squares of σ_i represents the amount of data variation in $\mathbf{X}$.

In practice, many data matrices exhibit a latent rank much lower than the actual rank measured by the number of non-zero singular values, or rather, by the number of linearly independent columns in the matrix. Intuitively, the reason is that, in practice, many columns (i.e., header fields of the packets we store in the matrix) are not exactly linearly dependent but they may be *highly correlated* and thus, approximately linearly dependent.

When this happens, we can reduce the size of the data by approximating the matrix using its latent rank, which is smaller than the observed rank.

Thus, in order to reduce the size of data sent for analysis and inference, while minimizing the approximation error (in the least squares sense), a lower rank representation of $\bar{\mathbf{X}}$ is produced by keeping only the largest $r \leq p$ singular values σ_i of $\bar{\mathbf{X}}$ and setting the remaining ones to zero, to get

$$\bar{\mathbf{X}}_p = \mathbf{U}\mathbf{\Sigma}_p\mathbf{V}^T. \tag{4.2}$$

It is provable that $\bar{\mathbf{X}}_p$ is the optimal rank-r approximation of $\bar{\mathbf{X}}$, in the sense of minimizing the Frobenius norm of $(\bar{\mathbf{X}} - \bar{\mathbf{X}}_p)$ [106]. Here, multiple σ_is may have small values, indicating that the latent rank of the matrix is smaller than the observed one. In that case, the dimensionality of $\bar{\mathbf{X}}$ can be reduced, while still retaining a high percentage of the information in the data by, for example, removing the smallest σ_is such that the sum of the squares of the retained singular values is at least 90% of the sum of the squares of all singular values.

In (4.2), all matrices have similar dimensionality as their counterparts in (4.1). However, since the last $p-r$ of singular values are set to zero, the corresponding columns in $\mathbf{U}$ and $\mathbf{V}$ can now be removed. This new representation is the truncated SVD representation of $\bar{\mathbf{X}}$ and is equivalent to $\bar{\mathbf{X}}_p$. It can be written as follows:

$$\bar{\mathbf{X}}_r = \mathbf{U}_r\mathbf{\Sigma}_r\mathbf{V}_r^T, \tag{4.3}$$

where the dimensions of $\mathbf{U}_r, \mathbf{\Sigma}_r, \mathbf{V}_r$ are now $(n \times r), (r \times r)$ and $(p \times r)$, respectively. In our system, the design parameters help decide which representation, i.e., $\bar{\mathbf{X}}_r$ or $\bar{\mathbf{X}}_p$, is more efficient as discussed at the end of § 4.4.3.

4.4.3 Dimensionality reduction: packets mode

Next, we seek to further reduce the size of the summary produced by each monitor via an even more compact representation of the packets in the reduced rank space. To do so, we seek to reduce the dimensionality *across* packets. We pose the problem to one similar to signal quantization, where the goal is to identify a set of representative values of a digitized signal which minimizes the approximation error. Thus, we seek to find a set of representative packets $\mathcal{R}$ that can approximate $\bar{\mathbf{X}}_p$ (or $\bar{\mathbf{X}}_r$). Ideally, $\mathcal{R}$ is constructed such that packets that are similar are mapped to the same representation. Let the size of the set of representative packets be $|\mathcal{R}| = k$. Formally, our problem can be posed as:

$$\min_{\mathbf{R},\mathbf{B} \in \{0,1\}^{n \times k} \bigcap \mathcal{RS}} \|\bar{\mathbf{X}}_r^T - \mathbf{R}\mathbf{B}^T\|_F^2, \tag{4.4}$$

where the $\mathcal{RS}$ constraint requires each row of $\mathbf{B}$ to sum up to 1, and the columns of matrix $\mathbf{R}$ are "centroids", effectively containing the packets $\mathcal{R}$, and matrix $\mathbf{B}$ is an assignment of each packet to a centroid. We use the Frobenius norm (i.e., Euclidean distance between a packet and its centroid) because (i) we have no prior knowledge about the distribution of packet vectors and, (ii) the problem admits very efficient approximations under this norm.

The above problem is known as *Vector Quantization* or *K-means clustering* and it is NP-hard in the above form [64]. However, there exist very efficient approximations when

using the Frobenius norm as the loss function, with the most widely used being Lloyd's algorithm [149]. In *Jaal*, we employ the "k-means++ algorithm" [70], an improvement upon Lloyd's algorithm that seeks to find a good initialization for the algorithm to converge faster on a good local minimum. We use this clustering algorithm since it is guaranteed to find a solution that is $O(\log k)$-competitive to the optimal clustering solution, and is known for fast convergence.

Note that $\mathcal{R}$ has k packet representatives; each represents a group (i.e., cluster) of similar rows in $\bar{\mathbf{X}}_p$. Increasing k increases resolution (more representatives) but increases the communication cost as well. We study how varying k affects detection accuracy for different attacks in § 6.5.

We propose *two* methods for processing and sending summaries to the inference engine. In the first, we build a *combined summary*, by applying the clustering algorithm on $\bar{\mathbf{X}}_p$. The output consists of k centroids $\tilde{\mathbf{X}}_p$, one for each cluster, as well as clustering metadata. The latter contains a membership counts vector $\mathbf{c}$, and is appended to $\tilde{\mathbf{X}}_p$ to form $\mathbf{S}_1^m$, which is then sent to the analysis and inference module. It is easy to see that the number of elements of $\mathbf{S}_1^m$ is thus $k(p+1)$, for each update from monitor m.

In the second method, we create what we call a *split summary*. Here, we apply the clustering algorithm on $\mathbf{U}_r$. The output consists of the k centroids $\tilde{\mathbf{U}}_r$ as well as the corresponding metadata $\mathbf{c}$ and $\mathbf{e}$. The summary in this case is the collection $\mathbf{S}_2^m = \{\tilde{\mathbf{U}}_r, \mathbf{\Sigma}_r \mathbf{V}_r^T, \mathbf{c}\}$. From (4.3), and since $\mathbf{\Sigma}_r$ is diagonal and can be sent as a vector of size r, the number of elements in $\mathbf{S}_2^m$ is $r(k+p+1)+k$.

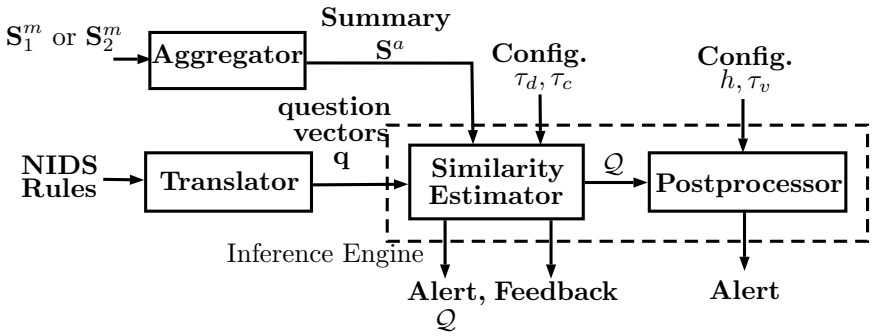


Figure 4.2: Inference Module.

Note that the information compiled in $\mathbf{S}_1^m$ is equivalent to that in $\mathbf{S}_2^m$, but is represented in a different format. More importantly, as discussed above the communication cost is different with each method depending on the design parameters r and k. In particular, when $r(k+p+1)+k < k(p+1)$, our system employs the split summary method and sends $\mathbf{S}_2^m$ to the analysis and inference engine; otherwise, $\mathbf{S}_1^m$ is sent.

4.5 Analysis and Inference

Jaal includes a centralized inference module. While this notion of central inference is similar to that with Snort or Bro, the inferences are based on summaries instead of raw packets. In brief, the inference module contains a translator that converts traditional IDS rules (specifically Snort rules) to a format that can be used with summaries. The inputs to this translator are "aggregated summaries" that are single consolidated representations of the summaries received from all monitors; they characterize all flows traversing the network in a given period of time. Then, the consolidated summary is checked against each transformed rule to detect attacks. Fig. 4.2 shows the different blocks in the inference module.

4.5.1 Aggregating summaries

Summaries generated by the different monitors need to be aggregated to get a global view. There are two ways to fetch summaries from monitors. In the first, monitors periodically send summaries to the controller. In the second, when a monitor accumulates a batch size (n) of packets, it constructs and ships the summary of this batch to the controller. At this point, the controller requests every other monitor to send its summary. In response, all monitors except those with fewer than n_{min} packets, send their summaries. Summarizing information using less than n_{min} packets incurs accuracy penalties because clustering and SVD generally do not perform well when data dimensionality is small. However, as shown in § 6.5, n_{min} is very small and not of concern in high speed networks.

Let the number of available monitors in the network be M. The aggregated summary, $\mathbf{S}^a = \left[\tilde{\mathbf{X}}_a \mid \mathbf{c}_a\right]$, with centroids $\tilde{\mathbf{X}}_a$ and membership counts $\mathbf{c}_a$, is composed by concatenating the summaries collected from all monitors in a tall matrix format. In particular, when monitor m sends its summary in the form $\mathbf{S}_1^m$, it is appended directly to $\mathbf{S}^a$. When $\mathbf{S}_2^m$ is sent, the previously removed $p - r$ zero-vectors in $\tilde{\mathbf{U}}_r, \mathbf{\Sigma}_r$ and $\mathbf{V}_r$ are first restored, and the matrices are multiplied to reconstruct $\tilde{\mathbf{X}}_p$. Then, the corresponding vector $\mathbf{c}$ is appended to $\tilde{\mathbf{X}}_p$ to create the same form as $\mathbf{S}_1^m$; this is appended to $\mathbf{S}^a$.

The number of rows in $\mathbf{S}^a$ reflects the total number of representative packets collected from all the monitors, and is thus at most Mk. Our results in § 6.5 show that this simple aggregation method is enough to achieve high detection accuracy with respect to a wide range of attacks.

4.5.2 Inference in Jaal

For ease of deployment, we seek to automatically translate Snort rules (since it is the most popular NIDS in use) to handle packet summaries. Snort has two main modules for detecting attacks. First, it contains a signature matching module that matches every signature against every packet, using pattern matching algorithms [54]. In *Jaal*, we translate such rules automatically to handle summaries. Second, Snort contains attack-specific *preprocessors*, to detect attacks that cannot be handled using signatures (e.g., port scanning). In *Jaal*, we design a module called the "postprocessor" that has a functionality that is equivalent to that of Snort's preprocessor.

Translator: This block takes as input a packet signature g (i.e., a Snort rule), and automatically translates it into what we call a *question vector* $\mathbf{q}$ of length p as follows. The value of an entry in $\mathbf{q}$ is the normalized value of the corresponding header field in g, and -1 in the absence of a corresponding header field in g (i.e., the field is irrelevant to g). *Jaal* measures the similarity of the rules captured in a question vector $\mathbf{q}$ to packet representatives ($\tilde{\mathbf{X}}_a$) in the summaries $\mathbf{S}^a$, using a simple distance measure as described later.

As an example, consider the translation of a specific Snort rule viz., a rule that relates to the SSH brute force attack [56]: *"alert tcp $EXTERNAL_NET any -¿ $HOME_NET 22 (msg: "INDICATORSCAN SSH brute force login attempt"; flow: to_server, established; content: "SSH-"; depth: 4; detection_filter: track by_src, count 5, seconds 60; metadata: service ssh; classtype: misc-activity; sid: 19559; rev:5;)."*

The rule postulates that an alert must be generated if 5 packets destined for the home network were received within the last 60s, with port number 22. To translate this rule into a question vector, *Jaal* initializes a vector of size 18 with -1 set for every position. Then, the position corresponding to the IP address is set to the normalized home network IP address and the position corresponding to port number is set to 22 (normalized version). This question vector is then used to make inferences as discussed below.

Similarity Estimator: Upon being provided with an aggregated summary, $\mathbf{S}^a$, this block measures the similarity between each question $\mathbf{q}$ in the transformed rule set, and every $\mathbf{x} \in \tilde{\mathbf{X}}_a$ in the summary. The distance function used is:

$$d_{\mathbf{q}}(\mathbf{x}) = \frac{\sum_{j:q_j \neq -1} |q_j - x_j|}{\sum_{j:q_j \neq -1} 1}. \tag{4.5}$$

The denominator in Eq. 4.5 normalizes the distance measure to account for questions with different lengths. If the distance is below a threshold τ_d, a match is declared and hence an alert is raised for the corresponding threat signature.

For questions that require a minimum number of matches (e.g, SYN floods), the similarity estimator sums all the counts $c_i \in \mathbf{c}_a$ corresponding to $\mathbf{x}_i$ with $d_{\mathbf{q}}(\mathbf{x}_i) \leq \tau_d$, and only raises an alarm if this sum is larger than τ_c. Here τ_d and τ_c are per attack parameters to be configured by a system administrator (τ_c may be directly carried over from Snort). In addition, the packet representatives in $\tilde{\mathbf{X}}_a$ matching $\mathbf{q}$ are collected in $\mathcal{Q}$.

Algorithm 1 Similarity Estimation

Input: question vector: $\mathbf{q}$, distance threshold: τ_d, centroids $\tilde{\mathbf{X}}_a$, counts $\mathbf{c}_a$, minimum count τ_c
Output: Binary Attack Classification
$sum = 0, \mathcal{Q} = \{\}$ **for** $\mathbf{x}_i$ *in* $\tilde{\mathbf{X}}_a$ **do**
    end
    $d_{\mathbf{q}}(\mathbf{x}_i) \leq \tau_d$
$sum \leftarrow sum + c_i$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \mathbf{x}_i$

if $sum \geq \tau_c$ **then**
    end
    OUTPUT: Alert, $\mathcal{Q}$

In some cases when more accurate analysis is required at the expense of higher bandwidth cost, feedback is sent to monitors for a raw batch of packets. We discuss this in § 4.5.3.

Postprocessor: As mentioned earlier, Snort employs preprocessors to handle distributed attacks that cannot be handled using signature-matching. In *Jaal*, we craft rules equivalent to those in the preprocessor, that can be used with summaries. Note here that in typical Snort implementations, these correspond to a small subset of the total attacks handled [58].

In crafting these rules, we observe that a common feature of such attacks (handled by Snort's preprocessor) is that the variance (i.e., the spread in the range of values) in a specific packet header field is large. Consider, for example, port scans, which are characterized by a large number of incoming packets, all with different port numbers. A large variance in port numbers indicates that a large number of distinct port numbers was seen and thus, warrants an alert. Similarly, distributed attacks, such as DDoS, are characterized by a large number of different source IP addresses.

Jaal uses a "postprocessor" to handle attacks exhibiting large variance across a subset of header fields. This postprocessor first processes packet representatives $\mathcal{Q}$, provided

Algorithm 2 Postprocessing

Input: Header field index h, variance threshold τ_v, centroids $\tilde{\mathbf{X}}_a$, counts $\mathbf{c}_a$
Output: Decentralized attack alert
  Initialize Empty array Z **for** $\mathbf{x}_i$ *in* $\tilde{\mathbf{X}}_a$ **do**
    end
    add $\mathbf{x}_i(h)$ c_i times to Z
  if $var(A) \geq \tau_v$ **then**
    end
    OUTPUT: Alert

by the distance estimator block, that match a given signature $\mathbf{q}$. Next it applies Algorithm 2, to measure the variance in a header field of interest h. It issues an alert for an attack if the variance is larger than a predetermined threshold τ_v. This threshold is to be configured as with Snort [58].

Example: To illustrate how the inference engine operates, consider the example of a distributed SYN flood attack. The $\mathbf{q}$ corresponding to this attack will have the SYN flag entry set and all other fields set to -1. The similarity estimator, using Algorithm 1 will declare a SYN flood attack if the number of packets matching this signature is greater than the threshold required to issue an alert with regards to this attack (τ_c). It also outputs the list of packet representatives $\mathcal{Q}$ that match the signature $\mathbf{q}$. To classify whether this attack is distributed, the postprocessor applies Algorithm 2 to every element in $\mathcal{Q}$, measuring the variance in the source IP header field. The attack is classified as distributed if the variance is above the predetermined threshold τ_v.

4.5.3 Trading cost for accuracy

Since *Jaal* uses packet summaries to infer patterns in packets, it is expected that the detection performance will be lower than that achieved if the raw packets were available.

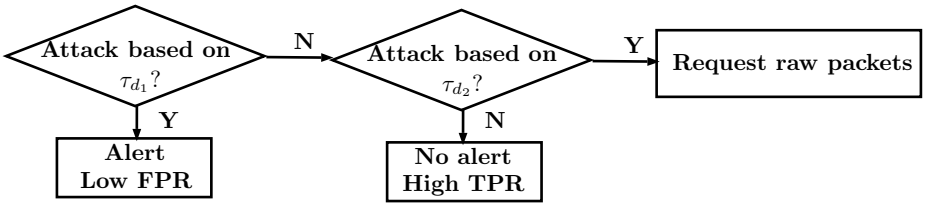


Figure 4.3: Feedback loop in *Jaal*.

Despite this, *Jaal* achieves reasonably high accuracy with low communication overhead as shown in § 6.5. To improve the detection performance further (with additional communication costs), we design a feedback loop that enables *Jaal* to realize multiple operating points on the detection accuracy-communication overhead tradeoff. In brief, based on the output of the inference engine, *Jaal*'s controller sends explicit feedback to certain monitors (discussed below) requesting finer granularity summaries or even raw packets for specific batches.

With the feedback loop, the inference in *Jaal* is logically performed in two stages using two threshold values τ_{d_1} and τ_{d_2} as shown in Fig. 4.3. To explain the rationale for this, recall from § 4.5.2 that τ_d is used to determine whether a specific centroid in the summary matches a given question vector. A large value for τ_d will successfully catch most attacks (high true positive rate or TPR) but may also result in a high false positive rate (FPR). On the flip side, a small value of τ_d will result in low FPR, but also misses attacks more often. Thus, we choose the first threshold τ_{d_1} such that the FPR is small, and choose $\tau_{d_2} > \tau_{d_1}$ such that the miss-detection rate is even smaller than that with τ_{d_1}.

Let the binary result of the threshold-based analysis using τ_{d_1} and τ_{d_2} be t_1 and t_2, respectively. Thus, we have four different output cases. When t_1 is positive and t_2 is positive (case 1), the system has high confidence that there is an attack and an alert is

raised. This because using τ_{d_1} results in low FPR. If t_1 is negative and t_2 is negative (case 2), no alert is raised. Here, the system relies on the high confidence with regards to the TPR when τ_{d_2} is used. If t_1 is negative and t_2 is positive (case 3), the controller asks the local monitors with the associated (uncertain) centroids in $\mathcal{Q}$ to send the actual packets corresponding to those centroids. The analysis is then done by pattern matching using traditional Snort rules and these raw packets. Requesting raw packets will decrease the overall FPR of the system, but will naturally increase the overhead. However, as we will show in § 6.5, this overhead is minimal and the feedback results in much improved detection accuracy. Finally, we note that the scenario where t_1 is positive while t_2 is negative (case 4) is unlikely (we never observed these in our experiments), since both analyses are done using the same summary $\mathbf{S}^a$ and thus it is expected that what is not missed in t_1 will also not be missed in t_2 (as t_2 guarantees higher TPR).

4.6 Flow Assignment

The flow assignment module seeks to assign flows to active monitors. In doing so, we have multiple goals. First, all flows passing through at least one monitor must be covered. Second, we require that a flow must be monitored by exactly one monitor. Duplicate monitoring of flows incurs unnecessary processing and bandwidth costs and more importantly, might lead to incorrect detection results. This is because *Jaal* uses summaries which do not retain information that could allow accounting for duplicate packet counts (same packet from multiple monitors) during inference. Third, in order to ensure that no monitor gets overloaded, we seek to ensure that the traffic monitored by the different

monitors is balanced, to the extent possible. Finally, flow assignment has to be very efficient and scalable (algorithm must be of low complexity).

This problem is challenging due to multiple reasons. First, each flow may only traverse a specific subset of monitors. Second, a flow can last for an unknown amount of time before it terminates. Moreover, flows can vary drastically in terms of "packet rate," i.e., the rate at which packets belonging to that flow are seen at an assigned monitor. This packet rate is used as a weight to represent the relative workload generated by the flow, in the assignment problem. Since a priori forecasts of flow arrival times, termination times, or weights is not possible, the system has to make its flow assignment decisions that satisfy the objectives above, in *real time*.

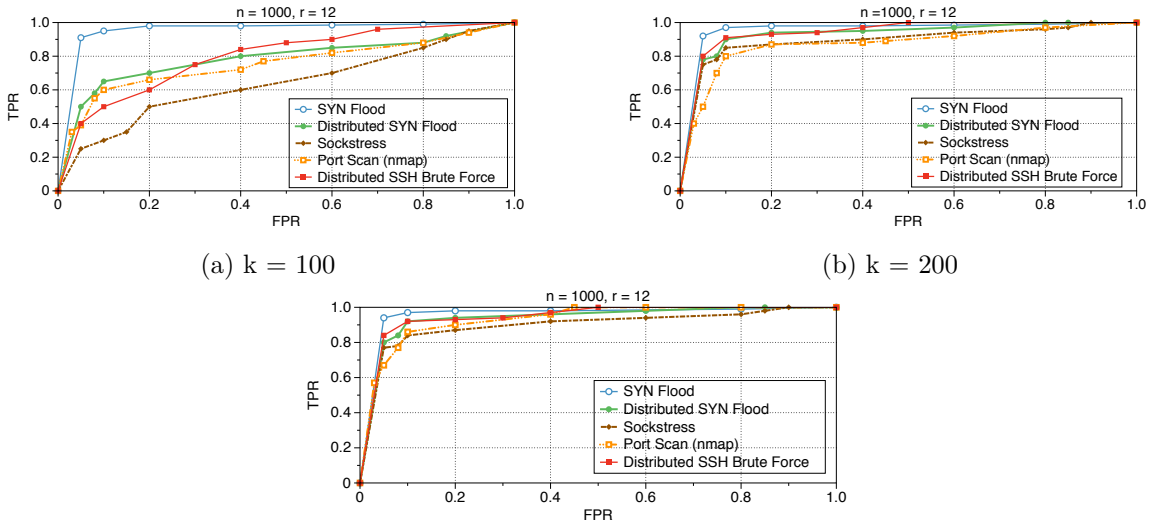
The flow assignment problem can be mapped to the *online optimization problem* [62], where the flows are the jobs to be assigned to M machines (monitors) upon their arrival, such that the maximum load across all monitors is minimized (i.e., load balancing). Upon the assignment of a flow to a monitor, the load on the monitor is increased by an amount equal to the weight of the flow. This increase is valid for the duration of the flow. The assignment will have to be non-preemptive, since it is impractical to reassign the other flows when a new flow arrives. We assume that monitors are homogeneous.

The metric used to evaluate online algorithms is the *competitive ratio* [80], the supremum, over all possible input sequences, of the maximum (over time and over monitors) load achieved by the on-line algorithm to the maximum load achieved by the optimal offline algorithm. One online algorithm performs better than another if it has a lower competitive ratio. For load balancing problems, the performance here is measured in terms of the

maximum load. Robin-Hood algorithm [74] has been shown to be optimal in solving online load-balancing of unknown duration tasks with assignment restrictions. In particular, it achieves a competitive ratio of $O(\sqrt{M})$, which is the lower bound for this class of problems. However, applying the Robin-Hood algorithm in practice is challenging because it requires the knowledge of incoming flow weights before an assignment decision is made. This is hard to do since the packet rates of a flow are not known a priori; estimates could be made (e.g., using machine learning) but it adds to the complexity. We omit the details of this algorithm in the interest of space (details are found in [74]).

Given the above challenges, We choose a simple greedy algorithm, which has been shown to achieve a competitive ratio of $\frac{(3M)^{2/3}}{2}(1 + o(1))$ [73]. The greedy flow assignment algorithm assigns an incoming flow f to the least loaded monitor within the subset of monitors on its path. This simple algorithm has two advantages. First, it does not require the estimation of flow weights to decide on the monitor to which the incoming flow is assigned. Second, as shown later in § 6.5, the assignment update is processed very quickly and thus, can potentially scale to WANs.

Since the number of flows traversing an ISP network is typically very large, making assignment decisions on per-flow basis and as new flows arrive and terminate, is not practical. Instead, we observe that subsets of flows, based on routing, can be grouped together. We call these constructs *flow groups*. In particular, a flow group is a set of flows that traverse a given common set of monitors. For a given flow group, we define the corresponding *monitor group* as the subset of monitors on the path of the flow group. Note that a monitor can belong to multiple monitor groups. In *Jaal*, a new flow is greedily assigned to the least



(a) k = 100

(b) k = 200

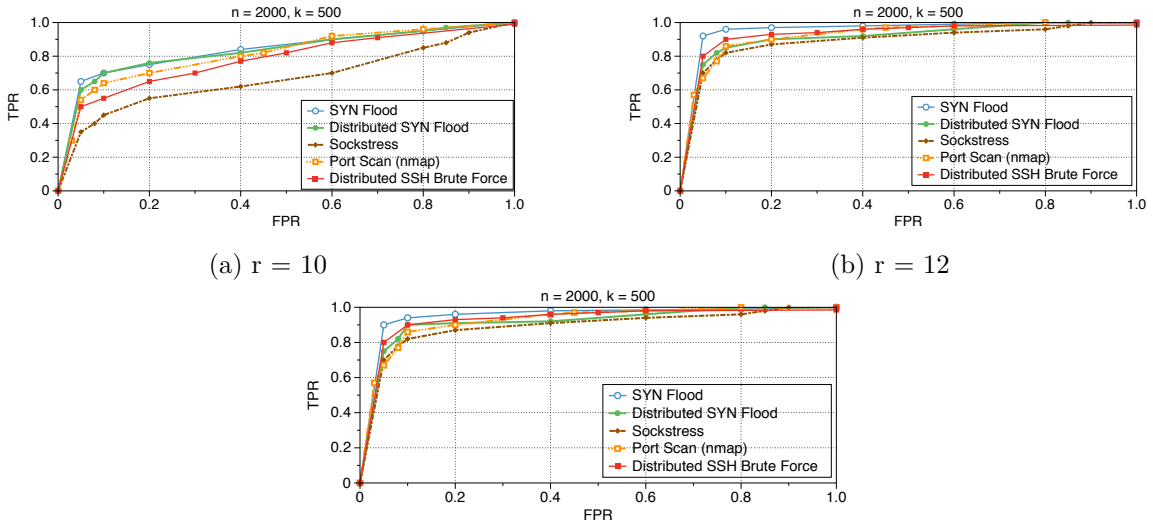
(c) k = 400

Figure 4.4: ROC curves for various attacks. Batch size = 1000, rank = 12, varying k, Trace 1.

loaded monitor in its corresponding monitor group. This setup allows us to implement a very efficient assignment algorithm that collects monitor load updates periodically, and then compute the assignment. Since flow arrivals and terminations events happen at arbitrary times, the performance of our greedy algorithm will approach the above theoretical bound of the greedy algorithm as the periodic update period P becomes smaller. In practice, we show in § 6.5 that it is comparable to that of Robin Hood algorithm.

4.7 Implementation

Central Components: The flow assignment, and the inference modules, are implemented at the central SDN controller using the open source Ryu SDN framework [55].



(a) r = 10

(b) r = 12

(c) r = 15

Figure 4.5: ROC curves for various attacks. Batch size = 2000, k = 500, varying rank, Trace 1.

We employ shortest path routing; thus, flow groups are just based on common source and destination prefixes.

As discussed in § 4.6, each incoming flow is assigned (based on load updates) to the least loaded monitor. This is done via an OpenFlow forwarding rule installed in the switch attached to the monitor. The entire module is written using Python's event-driven framework as a single threaded application. The module maintains a dedicated long lived TCP connection with each monitor. The flow assignment module polls monitors for load updates every $P = 2$ seconds (a larger value resulted in poor load balancing and a smaller value did not yield any significant improvements).

In the inference module, the new IDS rules are created offline and stored. The module executes a single threaded process that waits in an event loop for monitors to send summaries. It also maintains a long-lived TCP connection with each monitor and peri-

odically asks monitors for summaries. Once the summaries are received, they are checked against every locally stored question vector, and processed using the postprocessor, as described in § 4.5 and alerts if any, are logged.

Monitors: Monitors are implemented as network functions (NFs) at the SDN switches in Python, along with popular math and data mining libraries (NumPy, SciPy, pandas). They are instantiated by activating pre-stored VM images and attaching them to the chosen switches via a VLAN. The process is automated using an Ansible script. Each monitor executes two processes. The first process tracks load and responds to load queries when prompted. The second is responsible for the summarization tasks. Specifically, each monitor stores packets in a local buffer (NumPy array) and computes summaries. The centroids and their memberships are stored for one epoch (the periodicity with which the inference engine requests summaries) as a newly created hash table where the key is the centroid and the value is a list of actual packets associated with those centroids. If the monitor receives a request for raw packet dumps belonging to a specific centroid, it retrieves the list of packets using that centroid as the key and sends these to the inference engine. A hash table that is thus created, is deleted after the relevant epoch (2 seconds).

4.8 Evaluation

Next, we present our evaluation of *Jaal*. We use two ISP backbone traces from the MAWI group [52], Trace 1(2016/01) and Trace 2 (2016/02). to represent background traffic. We inject attack traffic in conjunction. Specifically, we consider five different kinds

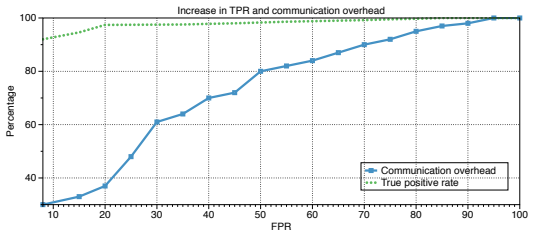


Figure 4.6: The increase in TPR and communication overhead as the acceptable FPR is increased.

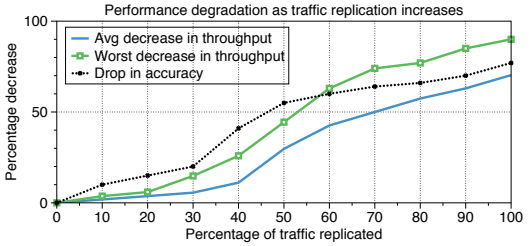


Figure 4.7: Performance degradation as the percentage of traffic that is replicated increases.

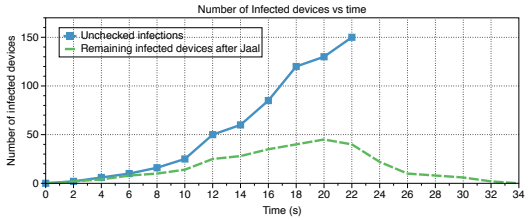


Figure 4.8: Mirai Attack: Unchecked infections versus infected devices shut off upon detected by Jaal.

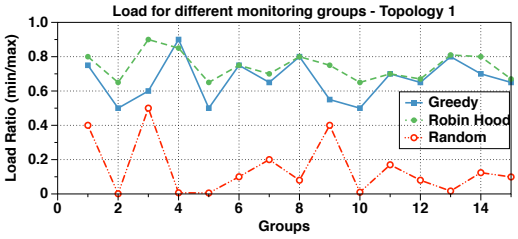


Figure 4.9: The load across different monitor groups for topology 1.

	Reservoir Sampling	*Jaal*
Distributed Syn Flood	54%	99%
Sock Stress	60%	98%
SSH Brute Force	42%	97%
Sockstress	56%	94%

Table 4.1: Comparing to reservoir sampling

of attacks: (i) SYN floods to represent DoS attacks, (ii) distributed SYN floods to represent DDoS, (iii) distributed port scans, (iv) distributed SSH brute forcing, and (v) Sockstress. DDoS, port scans, and brute forcing attacks are among the most common types of network-level attacks today [48]. Hence, we choose one of each type. We choose the Sockstress attack [32, 57] because it is more complex than other DoS attacks. It completes the TCP handshake and sets the TCP window size to 0, forcing the server to keep the connection alive for a long time. Finally, we also do a case study with the Mirai attack, and show *Jaal*'s effectiveness in countering it.

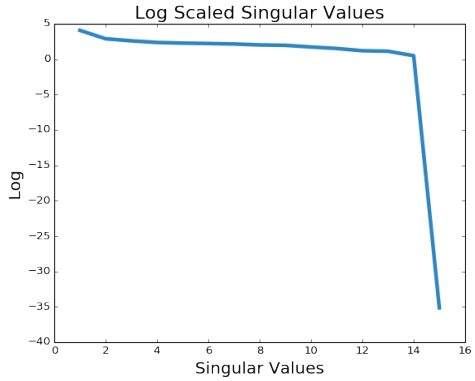


Figure 4.10: The magnitude of the singular values for a packet matrix of $n = 1000$.

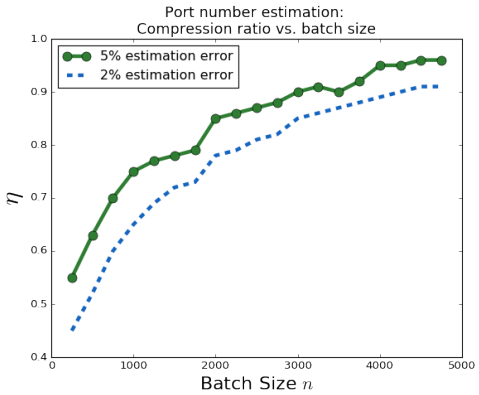


Figure 4.11: Percentage savings vs. the batch size for fixed error rates.

We now describe how we construct the different attacks. We first note that, while the MAWI traces might contain some malicious packets, it is very difficult to analyze the traces to determine which packets are anomalous and which are not as they are not labeled. In addition, running these traces through *Jaal* does not provide any intelligent information because the ground truth is not available. Thus, we treat the traces as benign traffic and explicitly inject malicious packets.

For each attack, we throttle the injected attack traffic to be at most 10% of the overall traffic. We get the volume of benign traffic by parsing the MAWI traces and predetermining the total volume of traffic that will be replayed at each time instant. The attack scripts then enforces the cap by stopping attack packets if the 10% quota has already been met. Note that socktress is a stealthy DoS attack and as such, does not need a large number of packets to succeed. Consequently, we did not have to enforce the 10% rule for it.

For distributed attacks we generate source IP addresses randomly from different subnets. This ensures packets take different routes and traverse different monitors. The total number of attacking IP addresses for each attack is approximately 200. For port scans,

we use the popular Nmap tool [53] to conduct scans. Nmap has a list of ports that it scans and we simply use those defaults.

Testbed and configuration: We implement *Jaal* on an in house SDN testbed that consists of 5 Dell PowerEdge 730 Servers (22 cores, 256GB memory, 12 ports) and two HP 3800 series switches (SDN enabled each with 48 ports) and 2 Arista 7280E switches (each with 72 ports). Note that we use SDN for the purposes of evaluating *Jaal*; it is not a necessary platform for the deployment of *Jaal*.

To represent the network, we use two realistic RocketFuel topologies [191] in our evaluations viz., Abovenet (which we call topology 1) and Exodus (which we call topology 2). Topology 1 has 367 routers and topology 2 has 338 routers. We create the topologies by instantiating the desired number of open vSwitch instances and connecting them using virtual links to match the configuration of the chosen topology. Our virtual topology can handle Gigabit traffic effectively, where the underlying SDN switches have 10GBE ports.

4.8.1 Detection Accuracy and Overhead

Overall results: In a nutshell, our experiments show that for all the attacks considered, *Jaal* achieves an average true positive rate (TPR) of ≈98% at about 9% false positive rate (FPR) with a communication cost of only 35% of that incurred by conventional IDS systems (i.e., copying raw packet headers and sending to a central engine for analysis). To achieve these, we need to select parameters that dictate the granularity of summarization, so as to yield good trade-offs between detection accuracy and cost. We use ROC (Receiver Operating Characteristic) curves towards choosing the appropriate configuration parameters (as discussed next).

ROC based analysis: Consider the case where $\tau_{d_1} = \tau_{d_2} = \tau_d$, i.e., no feedback loop is implemented. Two parameters influence *Jaal*'s tradeoffs between detection accuracy and communication cost, viz., the rank r, and the number of centroids k. We examine how accuracy changes for different values of r and k using ROC curves. ROC curves show the TPR versus the FPR and are typically used for understanding the tradeoff between the two metrics. Each combination of threshold values (i.e., τ_d, τ_c, τ_v) is a single point on the graph, and the TPR/FPR values are computed relative to ground truth. Each point is an average over 15 runs, each of which is 45 minutes long. Our experiments show that setting n to 20% of the size of the batch provides good resource consumption-accuracy tradeoffs. However, this might need to be re-evaluated by system admins in a real deployment. The process would be the same as the one we undertake above.

Varying number of centroids k: First, we fix $r = 12$, $n = 1000$, $n_{min} = 600$ and vary k and consider topology 1. In Fig. 4.4 the detection accuracy is shown for Trace 1 (similar results are observed for Trace 2 and are omitted in the interest of space). We observe that a value of $k = 200$ (i.e., 20% of the reduced dimension packets are sent for analysis and inference) yields very good detection accuracy for all attacks. Increasing k further yields diminishing returns in accuracy while increasing the communication costs. However, lowering k to 100 results in a significant detection penalty for all attacks except for SYN flood attacks. This is because of the boolean nature of flags. In other words, a packet is either as close as it can possibly be to a representative packet (centroid) with the SYN flag set, or is far away, depending on its SYN flag value.

Varying retained rank r: The rank of a matrix of raw packet headers, $\mathbf{X}$, is $\leq p = 18$. We study the distribution of the singular values of $\mathbf{X}$ and conclude that it is possible to retain 90% of the information in $\mathbf{X}$ by retaining only the top 16 values. To reduce the rank, we consider values for $r < 16$, as shown in Fig. 4.5 (with Trace 1 and topology 1). As apparent in Figs. 4.5b and 4.5c, $r = 12$ yields approximately similar performance to $r = 15$. Dropping r to 10, however, results in a high accuracy penalty for all attacks, as shown in Fig. 4.5a. We observe the similar results with Trace 2.

The Feedback Loop: From the ROC curves, we see that without the feedback loop, with properly chosen values of k and r, *Jaal* achieves an average TPR of ≈92% at about a 10% FPR, with a communication cost of only 30% of that incurred with raw packets. At the same time, if the TPR is chosen to be 98%, the FPR increases to 20%. We choose *attack specific thresholds* $\tau_{d_1} \neq \tau_{d_2}$, that yield these TPR and FPRs.

In Fig. 4.6, we plot the communication overhead vs. the TPR with the feedback loop. We also plot the average TPR for easy comparison. We see that with a value of τ_{d_2} chosen such that the TPR improves to 98%, the communication overhead (with the feedback loop implemented) only increases to 35% (from 30%) of that incurred with raw packets. Beyond this, the gain in TPR diminishes while the communication overhead rises sharply. Finally, we point out that with the feedback loop, the FPR is 9.1%, while the TPR is 98%. The reason for not seeing a further increase in TPR (the FPR drops) is that, since the traffic is already classified as attack traffic and only the data relevant to reducing false positives is retrieved, an increase in TPR is not observed.

Communication overhead: The communication overhead savings achieved by *Jaal* compared to sending raw packet headers as in conventional NIDS, is roughly proportional to k/n (if we ignore the raw packet overheads from feedback). After processing 2GB of traffic (in terms of headers only) system wide, *Jaal* only transmitted a total of ≈700MB. This corresponds to a 65% reduction compared to a traditional NIDS that requires sending all the 2GB. This is comparable to other specialized count-based sketches [148] but provides significantly richer information.

Feasibility of the vanilla approach of sending raw packets for inference: To test the feasibility of just copying and forwarding raw packets instead of generating summaries, we set up a realistic ISP topology and replayed backbone traces mixed with attack traffic (identical setup to the experiments above). The monitors (selected as previously) copy packets and transfer them to the central inference engine. We randomly vary the selection of the location of the inference engine across the experiments (a total of 25) to simulate different scenarios. The analysis engine runs Snort for the purposes of detection and we consider all the 5 attacks discussed earlier. Fig. 4.7 depicts the results of this experiment. We plot the percentage decrease in (a) the throughput and (b) accuracy on the Y-axis. The X-axis represents the percentage of traffic that is replicated for analysis purposes (fraction of packets that are copied and sent).

The network throughput reflects the average rate at which, normal traffic is processed at each switch (this takes a hit when it processes the copied traffic). The accuracy is measured as the percentage of attacks that the detection engine flags from among all the attacks injected. The throughput overhead reflects the "loss in throughput" (compared

to the baseline case without replication – this is what is in existence today) due to the introduction of the extra (copied) traffic. The results show that in the worst case (for one particular choice of the central analysis engine), the throughput overhead is 90%; in the average case the throughut overhead is 70%. Such high throughput hits will significantly affect network performance for the ISP (and the users subscribing to that provider). With *Jaal*, the replication level roughly corresponds to 35% which in turn corresponds to an average loss in throughput of less than 10% and a worst case hit of <20%.

We also see a 75% decrease in accuracy when all raw packets are sent to the inference engine. This loss is a direct artifact of missing attacks because of packet losses (arising both due to congestion and overloading of the inference engine). The loss in accuracy shown does not directly reflect *Jaal*'s accuracy – it corresponds to the loss in accuracy due to sampling (since 35% of the packets are replicated and transferred); as shown earlier *Jaal* does dramatically better in terms of accuracy.

Computation Costs: Our tests indicate that each monitor can handle rates of 300Mbps easily, indicating that the combination of SVD and k-means is not compute intensive. We omit detailed results because of space.

Case study of the Mirai attack: To evaluate *Jaal*'s efficacy in countering the Mirai attack, we emulate the attack on our testbed (using the published source code [41]). A randomly chosen node in the network initiates the Mirai scan and infects vulnerable devices. An infected device starts scanning for other devices and in turn infects them if possible. We randomly selected 150 nodes in our network to be vulnerable.

We compare two scenarios. In the first, there is no detection and response in place; we let the emulation run and track the total number of infections as time progresses. In the second, we configured *Jaal* to detect the scan (high variation in destination IP for common target ports, which are in our case 23 and 2323). We assume that the administrator shuts down traffic from an infected node whenever the scan is detected. *Jaal* detects the scan with an accuracy of 95% within 3s. Figure 4.8 shows the result of this experiment. We see that due to the brute force nature of the scan, the number of infected devices rises almost exponentially if left unchecked. With *Jaal*'s detection and the consequent response in place, the total number of infected devices never rises above 50 (infected devices are detected within 3s regardless). This means that in the worst case, there is a three-fold decrease in the number of devices that could have launched the subsequent DDoS attack. If the DDoS attack is triggered later, the number of devices could be significantly smaller (as seen in the figure) since additional devices are detected and disabled.

Comparison to Reservoir Sampling: Next, we compare *Jaal*'s performance with that of a system using a state of a art sampling technique, viz., *reservoir sampling* [200] [103]. For fair comparison, we set up reservoir sampling with roughly the same communication overhead incurred when *Jaal* uses $r = 12, k = 200, n = 1000$. Specifically, we set the size of the reservoir to 250 and then wait until the sampler has processed 1000 packets prior to shipping the samples. Table 4.1 shows the comparison results. Since reservoir sampling keeps a fixed-size running uniform sample of the entire stream, attack packets sent over a short period of time will get "diluted" in the sample by a large number of non-attack packets and thus will not be well represented. This results in poor detection accuracy. Note

that it is possible to bias the sampling in favor of specific header fields, but this would not be a "general" approach towards detecting a large class of attacks.

4.8.2 Individual Module Performance

Next, we examine the performance of *Jaal*'s modules.

Greedy flow assignment: We compare *Jaal*'s greedy flow assignment to the optimal online algorithm (i.e., Robin Hood). The weights for Robin Hood are given (we know the ground truth) and the assignment is done on a per-flow basis; this is an ideal but impractical scenario. We also consider an algorithm which randomly assigns flows to any monitor in the corresponding monitor group. We consider Topology 1, and fix the number of monitors to 25. We use a load update period $P = 2$s. In Fig. 4.9, we plot the time averaged load for different monitor groups j. We see that the greedy assignment closely mirrors the performance of the Robin Hood algorithm (with deviations of 10% on average and 14% in the worst case). The random assignment performs poorly as expected. The results are similar for topology 2 and are omitted in the interest of space.

Dimensionality reduction using SVD: Fig. 4.10 depicts the variation in the magnitude of singular values (recall § 4.4.2) for $n = 1000$. The drastic drop in magnitude beyond the top 14 values shows that the lower values are either zero or are very small and can thus be ignored compared to the dominant singular values. In fact, as seen in Fig. 4.5, $r = 12$ gives us the best trade off between accuracy and communication overhead.

Variance estimation: As discussed in § 4.5, variance in fields such as port numbers, is used in *Jaal*'s postprocessor to detect distributed attacks. We examine how good is *Jaal*'s estimate of the variance in the destination port header fields, as we vary k,

for different batch sizes n. We observe that the error in variance estimation is less than 5% when $k/n > 0.2$, i.e., when summaries cost only 20% compared to sending raw packet headers, and $n \geq 1000$. Next, we study the effect of batch size on the communication costs. In Fig. 4.11, we plot the compression ratio $\eta = 1 - k/n$ (which is proportional to the communication cost saved) vs. the batch size n for two different maximum variance estimation errors ϵ. As the batch size increases, *Jaal* attains better compression ratios for a given maximum error ϵ. For example, for $\epsilon = 5\%$, and $n = 2000$, *Jaal* achieves a compression ratio of about 85%. To ensure very low ϵ (alternatively very high accuracy) when packet arrival rates are low, a monitor either needs to use a smaller η (which may be acceptable given the lower load) or wait until a larger number of packets are accumulated prior to summarization (thus delaying inference).

Discussion on TPR and FPR: Finally, we analyze why attacks are missed and why there are false positives. We examine the centroids to which packets get assigned by parsing the entire list of centroids and the packets that have been assigned to them and determine which malicious packets are assigned to centroids representing normal traffic and vice versa. We find that both kinds of errors occur when attack packets are similar to background traffic and the clustering is not able to differentiate between them. However, we point out that both the TPR of 98% and FPR of 9.1% that *Jaal* achieves, are well within the reported acceptable performance levels of a NIDS such as Snort or Bro [34, 167, 84].

One could further request finer grained summaries or raw packets, when an alert is to be raised to reduce false positives (with increased cost); conceivably, even random

retrieval of full traces when an attack is flagged, could reduce the FPR. However, we leave these possibilities to future work.

4.9 Related Work

In this section, we discuss relevant related work.

NIDS: There have been attempts to address the scalability issues in centralized NIDS [122, 188] but only to the extent of scaling to enterprise scale networks. These solutions do not hold up to the ISP scale intrusion detection. For example, the approach in [122] assumes that there will be a single entry and exit point into the network which is not true for large WANs. The framework presented in [188] requires the transfer of raw packets to clusters to make inferences, which as discussed, leads to both performance and inference accuracy degradation.

Network Monitoring, Sampling, Sketching: Packet sampling has been proposed for heavy hitter detection [135], packet length estimation [208] and flow size estimation for small flows [125]. In [94], the authors provide an API for collecting flow statistics at different aggregation levels. These approaches however, are tailored towards measuring only a few pre-specified metrics of interest. In [182, 183] a more general sampling strategy is developed; the strategy however, is only effective in measuring aggregate statistics over streams (e.g. heavy hitter detection, entropy estimation). This is not conducive to general intrusion detection where retaining the correlations across individual headers is necessary.

There has been progressive work on estimating various network related metrics via sketching [148, 173]. While sketches, unlike sampling, offer cost and accuracy guarantees,

they suffer from the same fundamental problem of being restricted to measuring some aggregate statistic over a specified dimension. Our solution can handle the generality of NIDS rules by retaining finer grained information.

4.10 Discussion

In this section, we discuss a few avenues for future work as well as areas where *Jaal* can be improved.

Payload-based Attacks: *Jaal* was not primarily designed to detect payload based attacks. We believe that payload monitoring by ISP's raises some important privacy questions that deserve debate. However, *Jaal* can handle some rudimentary payload-based attacks with a simple extension. Specifically, one approach to detect the presence and/or count of certain keywords (e.g., a specific malicious website, or the term ".exe" which signifies the presence of an executable) is to construct a term frequency matrix using a batch of packets - a popular technique used in sentiment analysis and recommender systems [177]. This matrix can then be treated the same way as the headers-only batch is considered in this chapter.

False Positives: One area of concern with deploying *Jaal* is the high FPR. We argue that the TPR and FPR rates achieved by *Jaal* are significantly better than what is possible today. In other words, there are no systems to perform effective intrusion detection at these scales with these TPR/FPR rates. Future work will look into reducing this false positive rate (e.g., we will examine if using multiple windows of packet summaries and correlating the of inferences from those windows can help in this regard). Note that the

high FPR is a problem inherent to signature based systems and is not unique to *Jaal*. Since a system such as *Jaal* has never been deployed at ISP-scale networks, it is unclear what the implications are in terms of FPR at that scale. However, we expect analysts to parse logs just as they would for an enterprise IDS. Establishing a dialogue with ISP's about the deployment of *Jaal* is left to future work.

Applicability: While we showcase *Jaal*'s performance with a specific set of attacks, we expect *Jaal* to detect any attack that can be characterized by a Snort/Bro-style packet signature. However, we acknowledge that attacks that only need a few malicious packets to succeed might be more challenging. For example, the TCP reset attack needs only one packet with the RST field set to induce malicious activity. Currently, *Jaal* cannot handle such attacks because a single packet will likely get assigned to an existing centroid that is not representative of it. Fixing this problem requires changes to the clustering algorithm and we defer improvements along this direction to future work.

We also note that *Jaal* can be used alongside smaller-scale (e.g., enterprise) IDSs. *Jaal* can also be used in conjunction with sketches. For example, sketches could be used for simple heavy hitter detection while *Jaal* can be used to detect more complicated attacks requiring correlation across multiple header fields.

Adaptive attackers: Finally, a last avenue of future work is evaluating how robust *Jaal* is to an intelligent attacker that is aware of how *Jaal* works. Whether or not an attacker can craft packets to explicitly bias the summarization process is something we intend to explore in future work.

Chapter 5

Streaming Lower Quality Video over LTE : How Much Energy Can You Save ?

5.1 Introduction

Current reports indicate that video streaming to smartphones is experiencing an unprecedented growth [9]. The emergence of LTE (Long Term Evolution standard) [186], which offers significantly higher throughput compared to the previous generations of cellular networks, has fostered this growth. Streaming video over a cellular network however impacts the battery consumption of a client device. While User Equipment (UE) and specifically smartphones, have grown in complexity with better displays and faster CPUs, the battery technology has not been able to keep up. LTE, due to its ability to sustain significantly

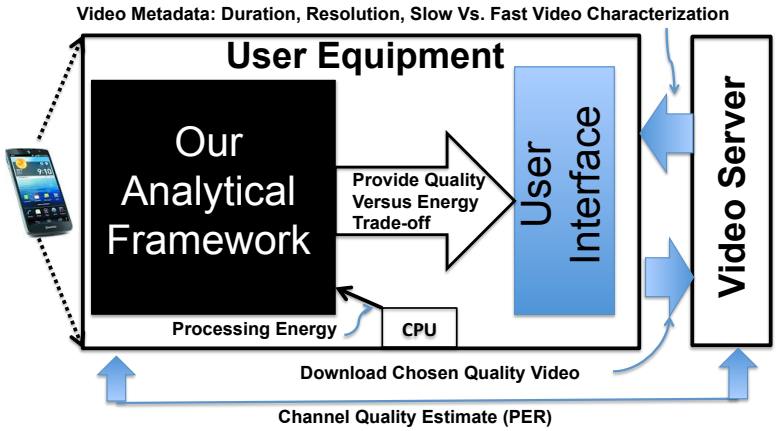


Figure 5.1: Applicability of our framework.

higher user throughput compared to 3G, exacerbates the battery problem during video downloads, since the higher downlink/uplink data rate translates to higher energy consumption [129]. It has been shown that the radio interface is a significant power consuming resource [129].

Today, adaptive bit rate streaming has become a common practice for streaming video [199]; by detecting the user's bandwidth, the quality (resolution/bit rate) of the video stream is adjusted so as to improve the user's quality of experience. However, to the best of our knowledge, changing the quality of the video to lower the energy consumption on the user's smartphone has not been previously studied. In particular, a user may choose a lower quality video stream, even when bandwidth/CPU resources are adequate, to reduce her battery drain. We seek to explore this dimension in this work.

As one might expect, a user can decrease the energy consumed on her smartphone when downloading a video, by downloading a lower quality version of the same video. In some cases (poor channel conditions), downloading a lowered quality video could even

improve user experience (prevent stalls); in fact, there has already recent work that advocate the use of lowered video quality (albeit in a wireline setting) to enhance user experience during downloads [66]. However, today there do not exist any tools that allow the user to get an estimate of how much energy she can save by choosing a lower quality video for downloads over cellular connectivity. Such an estimation is intricately hard because of the following reasons: **(i)** The estimation has to be made without downloading any of the versions of the video; in other words, it has to be based on a set of parameters that characterize the video to be downloaded; **(ii)** The savings from downloading a lower quality version would depend on how the UE state transitions (described later) [186] are affected by the arriving video traffic. This in turn would depend on the channel conditions perceived by the user at that time. These challenges essentially require that any framework must be holistic and tie in the interactions between the video flow characterization, the LTE scheduler and the energy transitions due to traffic arriving at the UE.

Goal and Vision: In this chapter, we seek to develop an analytical framework which takes as inputs, factors that influence energy transitions at the UE (i.e., video characteristics, channel conditions) and yields as output an estimate of the energy savings possible with a lowered quality video download of a given stream. A pictorial representation of how our framework can be applied is shown in Fig. 5.1. Either the UE or the video server can perform a small set of calibration measurements to estimate the channel quality in terms of PER. Alternatively, a model that maps the signal strength to PER could be used to estimate the PER at the UE side. The video server would provide metadata [210] from the video clip chosen for download in the form of resolution, a coarse characterization of

slow versus fast video (using tools such as AForge [4]) and the duration of the video. The UE will also locally estimate the energy required to process the received video frames for the different versions of the video. Our model yields as output an estimate of the energy consumed on the network interface with the different resolutions of the video, the user seeks to view. This combined with the the processing power provides the user with an estimate of the total energy with the different versions. She can then make an educated decision on the version of the video to download from the server.

Contributions: As our primary contribution we build a mathematical framework to capture the interactions between video traffic, the LTE scheduler, and the energy state machine at the UE. In addition to being useful for near real-time estimation of energy savings from choosing lowered quality videos for downloads, it provides a fundamental understanding of how and why different input factors influence energy consumption. In essence, the framework considers a general model of video traffic, and characterizes the arrival process of video packets at a client device (UE) after they traverse an LTE-based wireless link. The arrival process in turn calibrates the transitions between the different energy states in which the UE can reside, and the likelihood of being in each of those states. We validate our framework via extensive simulations and through experiments on a real smartphone in a variety of scenarios, thereby demonstrating its accuracy (the results are within ≈ 5 % of the real measured values) as well as generality.

Some interesting insights arising from our work are:

- Choosing a lower quality video stream incurs a small penalty in terms of the video PSNR (Peak Signal to Noise Ratio) but results in significant energy savings; specifically, a PSNR

reduction of 10.1% can fetch energy savings of the order of 375 J for a video of duration 15 minutes. When a user views videos over extended periods, the energy savings can therefore be significant.

- While as expected, streaming higher resolution videos result in higher energy, the increase depends on whether it is fast or slow motion video. For example, in good channel conditions, moving to a lower resolution from a higher resolution results in a 480 mW ($\approx$ 26 %) reduction in power (energy consumed per unit time) for fast motion video, but a 418 mW increase ($\approx$ 23 %) for slow motion video.
- In poor channel conditions, moving to a lower resolution results almost in identical power savings for slow and fast motion video ($\approx$ a 19.5 % decrease in power). However, the savings in milliwatts is higher for fast motion video.
- For typical video transmissions, one observes that the time spent in some of the LTE energy states is insignificant regardless of the resolution.

Scope: Our framework primarily accounts for the energy consumed by the network interface on the UE. In addition, there is a processing energy consumed on a device for processing/playing back the video frames; this energy is device dependent and we use empirical results that are driven by experiments (this can be measured locally on any device).

For validation, we assume that videos are streamed using fixed bit rates. However, our framework can be applied to adaptive bit rate streaming as discussed in Section 5.6. We employ video resolution and PSNR as the metrics for quantifying video quality. It has been shown that the perceived video quality on mobile devices is affected by the size of the

screen, user mobility, and ambient light [207]. Accounting for these factors is beyond the scope of this work and will be considered in the future. For analytical tractability, we also assume that the user is stationary during the course of video streaming.

While we validate our framework via simulations and experiments, we do not implement the complete system shown in Fig. 5.1. To implement such a system, we will need to make changes to the video server so as to deliver the appropriate metadata to the client UE, and also have a dynamic PER estimation tool for the LTE link; these are beyond the scope of this work and will be considered in future work.

5.2 Relevant Background

In this section, we describe aspects of LTE that we seek to capture in our analytical framework.

The Radio Resource Control (RRC) State Machine: The LTE RRC state machine captures the different *energy* states that a UE can be in and has two primary states: **rrc_idle** and **rrc_connected**. The latter has three modes as shown on the right side of Fig. 5.2. If the UE is in **rrc_idle**, then any data exchange (even corrupted) triggers a transition to the **rrc_connected** state. The UE then enters the continuous reception mode and monitors the physical downlink control channel (PDCCH), on which control information is delivered from the base station (referred to as enB in LTE jargon [186]). At this time, the UE also starts its *continuous reception timer*, T_c. If no packets are received before the expiry of this timer i.e., in T_c, the UE enters the Short DRX mode. In this mode the UE alternates between ON and OFF periods (called DRX cycles) to save energy. If during any

ON period, the UE receives data or has data to send, it returns to the continuous reception mode.

Upon entering the Short DRX mode, a different timer, T_s, is set. If there is no data transfer (received or sent) prior to the expiry of this timer, the UE enters the Long DRX mode. The Long DRX mode is similar to the Short DRX except that it has longer DRX cycles and a bigger timer value (T_l) associated with the time prior to exiting this state. Thus, $T_{tail} = T_c + T_s + T_l$ represents time for which no packets should be either received or sent in order to return to the **rrc_idle** state, and is referred to as the LTE tail period.

Packet transfers in LTE: The transitions between the states in the LTE RRC state machine are dictated primarily by packet receptions during video streaming. This in turn is handled at the MAC (and the PHY) layer of the LTE protocol stack. Our focus in this work is on the MAC layer, and we abstract the PHY in terms of packet error probabilities. Thus, we describe this layer in some detail below. A more detailed description of all the layers of LTE can be found in [186].

MAC and Physical Layers: The MAC layer implements a Hybrid Automatic Repeat reQuest (HARQ) protocol for reliable packet transfers. It also dynamically selects the Modulation and Coding Scheme (MCS) to be used at the PHY, based on the channel conditions. Transmissions in LTE are organized into frames that are 10 ms long. Each frame is divided into ten 1 ms subframes. The LTE transmission time interval (TTI) specifies the granularity at which packets are scheduled and this is done once every subframe (TTI = 1 ms).

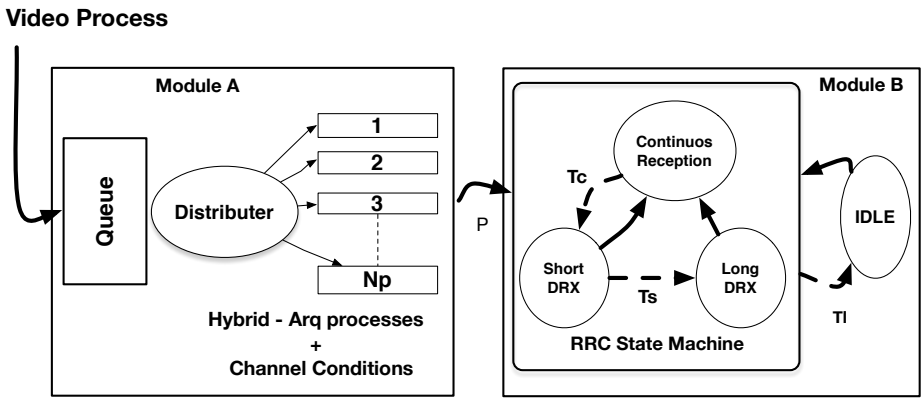


Figure 5.2: A depiction of the system considered in our framework. Module A represents the LTE scheduler and Module B represents the UE RRC machine. The output of Module A, which is the probability of a packet being sent in a TTI, p, is the input for Module B.

The HARQ process: The HARQ process is different from a regular ARQ process in how it sends data and retransmissions. In LTE, *incremental redundancy* and *chase combining* are both supported. With these approaches retransmissions add required redundancies and are combined with prior transmissions to increase the likelihood of packet success.

LTE uses multiple HARQ processes simultaneously. The number of these processes, N, are chosen such that it is greater than the round trip time (RTT) of a single HARQ process (in time slots). The multiple processes transmit packets one after the other (round-robin), as a continuous stream without waiting for acknowledgments (ACKs) or negative acknowledgments (NACKs). Each process will have received an ACK or a NACK by the time it is its turn to transmit again. The number of HARQ processes for FDD (frequency division duplexed) LTE is 8 (the RTT with LTE is typically < 8 ms [186]). The number of transmission attempts that a single HARQ process makes before dropping the packet is generally 5, i.e., it makes 1 transmission and 4 retransmission attempts.

5.3 Our Analytical Framework

In this section, we build our mathematical framework for understanding the trade-offs between video quality and the battery consumption at the UE. As expected, the UE energy consumed will decrease if one were to lower the quality of the video that is downloaded. However, the savings will depend both on the type of video (resolution, slow versus fast video) as well as the channel conditions (poor versus good link quality).

To reiterate, we envision our framework to be used as depicted in Fig. 5.1. Prior to sending a video stream, the sender characterizes the video in terms of its type (slow or fast motion) and resolution. The sender and the receiver also perform a set of calibration measurements to estimate the link quality in terms of the PER. These parameters are then input to our framework, which then outputs the expected energy consumption at the UE for a download of a particular duration.

We first model the video input process. Subsequently we characterize how these video packets are processed by the LTE scheduler and thereby affect the transitions between the energy states at the UE. The notation used is summarized in Table 5.1.

5.3.1 Video Input

The input process characterizes the arrival of video packets into the LTE buffer. We assume that the video is composed of I, P and B frames [1]. A segment corresponding to an I frame is typically much larger than the MTU (maximum transmission unit) of the network and must be fragmented into multiple packets. The I frames are also less frequent than the much smaller P or B frames. The P and B frames are typically smaller than the

[1] For details on video representations please see [82].

MTU supported by the network. Since in terms of size, P and B frames are similar, we do not distinguish them in our model. In essence, we seek to capture two different phases of arrival which correspond to that of I frames and P/B frames, respectively. A natural choice for such a setup is the Markov modulated Poisson process (MMPP), which represents a doubly stochastic Poisson process [127]. The first state of the MMPP represents the arrival of I frames and the second, the arrival of P/B frames. The rate of transition from state 1 to 2 is r_1 and that from state 2 to 1 is r_2. When the process is in state 1, packets that are generated due to I frames arrive at a rate λ_1; in state 2, packets generated due to P/B frames arrive at a rate λ_2. The process can be represented by the infinitesimal generator R and the rate matrix Λ, given by:

$$R = \begin{bmatrix} -r_1 & r_1 \\ r_2 & -r_2 \end{bmatrix}, \qquad \Lambda = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \tag{5.1}$$

The steady state vector $\boldsymbol{\pi}^{mmpp}$ (which represents the probability of being in state $i, i \in \{1, 2\}$) is given by

$$\boldsymbol{\pi} = (\pi_1, \pi_2) = \frac{1}{r_1 + r_2} (r_2, r_1) \tag{5.2}$$

The expected arrival rate of the 2-MMPP process, λ_{avg}, is given by $\boldsymbol{\pi}\boldsymbol{\lambda}$, where $\boldsymbol{\lambda}$ is the column vector with the diagonal elements in Λ, i.e., $\boldsymbol{\lambda} = \Lambda.e$, where, $e = (1, 1)^T$.

Parameterizing video quality: In order to use the above representation we need to map the given video quality to the parameter λ_{avg} (this parameter influences the energy consumed as we will see later). Videos can be categorized as fast or slow motion videos [4]. Fast motion video is characterized by successive video frames that have little in

Parameter	Description
$\boldsymbol{\pi}^{mmpp}$	Steady sate vector of the 2-MMPP video process
R	Infinitesimal generator for the 2-MMPP video process
Λ	Rate Matrix of the 2-MMPP process
p	The probability of receiving something in a given TTI
N_p	The number of HARQ processes
r	Maximum number of transmission attempts of a single HARQ process
$\boldsymbol{\pi}^{harq}$	The steady vector of a single HARQ process
$\boldsymbol{\pi}^{rrc}$	The Steady state vector of RRC state machine Markov Chain

Table 5.1: Key parameters in our mathematical framework

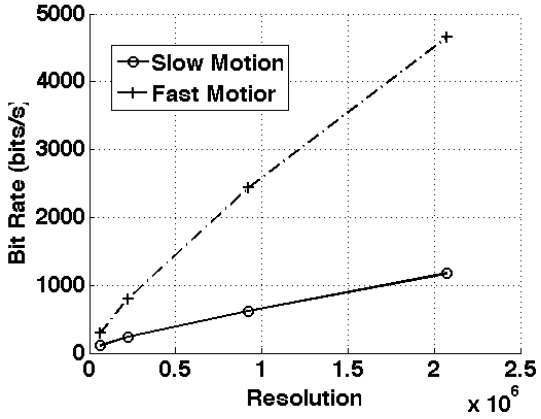


Figure 5.3: Pixel Density (Resolution) VS Bit rate

common while slow motion video is characterized by successive frames that have a lot in common. Accordingly, fast-motion videos consume more bits in encoding than slow-motion videos. If the effective bit rate is known ($Bitrate$), $\lambda_{avg} \approx \frac{Bitrate}{E(Packet\ \ Size)}$.

If the video server can provide information with regards to the bit rates associated with different versions of the video clip as metadata, λ_{avg} can be readily computed. One can also empirically compute λ_{avg} from the resolution and the category of the video (slow or fast motion) if the bit rate is not readily available as follows.

The resolution of a video stream is essentially a function of the number of pixels per frame; the higher this number, the higher the resolution. To map the resolution to the bit rate, we perform measurements across 5 video streams for each type of video (fast or

slow). We find that the bit rate is directly proportional to the resolution of the stream as shown in (shown in Fig. 5.3). The proportionality index depends only on whether the video is of fast or slow motion. With such a characterization (fast versus slow), we find that the prediction error is $< 5\%$. Thus, given a certain video type and its resolution, the server can determine the average arrival rate of packets to the MMPP process (using these offline measurements). Stated otherwise, the quality of the video in terms of its resolution can be used to characterize its arrival process to the LTE scheduler.

Fig. 5.3 demonstrates this relationship for each type of video. From the figure we can directly infer the decrease in λ_{avg} when the resolution is changed.

5.3.2 Modeling LTE effects

Next, we characterize the behavior of the RRC state machine at the UE given (i) the type of video being streamed and its resolution, and (ii) the state of the channel. The system has two distinct parts as shown in Fig. 5.2. The first part (Module A) reflects the process of transfer of the video traffic over LTE to a specific UE. The second part (Module B) characterizes the RRC state machine at the UE. The overall objective here is to determine the expected time spent in each of the RRC states. Module B essentially takes as input p, which represents the probability of receiving a packet (either a decodable packet or a corrupted packet) in a TTI. This probability essentially depends on the arrival process of the video flow, the functionality of the LTE scheduler (Module A) and the channel quality of the wireless link.

Assumptions: We make the following assumptions for analytical tractability. *First,* we assume that the UE is relatively stationary and thus, the channel conditions do

not change (slow fading) for the duration of a transmission. However, we assume that they can vary between transmissions due to fading, and thus the likelihood of a packet succeeding in a transmission attempt is independent of what happens in other attempts. *Second,* we ignore synchronization issues. Note that we are only interested in the *average* energy due to the UE being in a state and not the transient energy behaviors while in a state (e.g., we only account for the average energy because of being in the Short DRX state).

To begin with we assume that the UE in question is scheduled every TTI. We relax this later to account for the possibility that it gets scheduled once every N_p TTIs, on average.

Packet processing at the LTE transmitter: The packet processing at the LTE transmitter consists of two parts. First, we have an MMPP/G/1 queue to which the video packets arrive. The server of this queue essentially acts as a distributor and places the packets in one of N_p HARQ processes. Note that in order for the distributor to place a packet, at least one of the HARQ processes must be empty. Next, the HARQ process delivers the packet to the UE. First, we determine the service time distribution for our MMPP/G/1 queue and thereby compute its utilization. We later discuss how we map this to the probability of Module B receiving a packet in a TTI.

Characterizing the service time: The service time is influenced by the functions of the LTE HARQ processes. We denote it by S, which is essentially the time it takes for a packet to be assigned to a HARQ process by the distributor. We first describe how a single HARQ process functions and then discuss how we determine the distribution of S considering the N_p processes together.

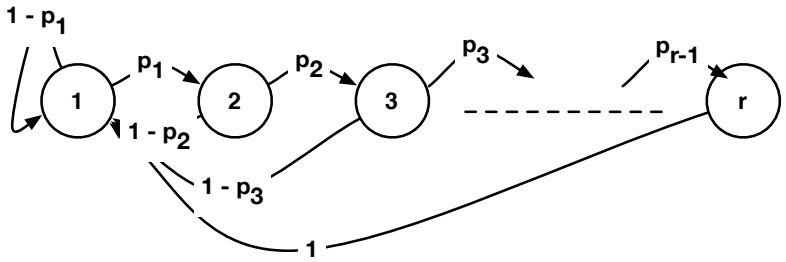


Figure 5.4: Markov Chain representing a single process hybrid-arq. Each state $i, i \in \{1, 2...r\}$ represents the transmission attempt and r is the maximum number of transmissions allowed.

Model of a HARQ process: We model a single HARQ process as a finite state discrete time Markov chain as shown in Fig. 5.4. The number of states, r, corresponds to the maximum number of transmission attempts allowed (after which the packet is dropped). The initial state (state 1) refers to a state wherein the HARQ process receives a new packet from the distributor. p_i is the probability of a packet being received by the UE in error, while in state i. A successful transmission while in state i, occurs with probability $1 - p_i$, and results in a transition to the initial state (state 1) and the process gets the next new packet. Because of additional FEC or change to a lower MCS (this is how HARQ functions), later re-transmissions have a greater chance of success, i.e., $p_i \geq p_{i+1}$ [2]. It is easy to see

[2]In the NS3 LTE simulator we use [20], we see that all the p_is are nearly equal although this relation holds in general.

that the transition probability matrix for the Markov chain, $\boldsymbol{P}$, is thus:

$$\boldsymbol{P} = \begin{pmatrix} 1-p_1 & p_1 & 0 & \cdots & 0 \\ 1-p_2 & 0 & p_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ \vdots & 0 & 0 & \cdots & p_{r-1} \\ 1 & 0 & 0 & \cdots & 0 \end{pmatrix} \tag{5.3}$$

The steady state probability vector, $\boldsymbol{\pi}^{harq} = (\pi_1, \pi_2, ..\pi_r)$, is then computed by solving the equations $\boldsymbol{\pi}^{harq}.\boldsymbol{P} = \boldsymbol{P}$ and $\sum_i \pi_i^{harq} = 1$. Specifically,

$$\pi_1^{harq} = \frac{1}{1 + \sum_{i=2}^{r} \prod_{j=1}^{i-1} p_i}, \qquad \pi_{k,\forall k>1}^{harq} = \pi_1^{harq} \prod_{i=1}^{k-1} p_k \tag{5.4}$$

Joint consideration of the HARQ processes: To derive the service time distribution, we need to characterize how the N_p HARQ processes function together as a whole. To recap, the N_p processes are served in a round robin fashion. In every TTI, there is only one HARQ process that is scheduled for transmission. Since $N_p >$ the RTT of in terms of TTI's, each transmitting process will receive an ACK or NACK by the time it is its turn to transmit again.

Deriving S: The distributor assigns the packet at the head of the queue to next available HARQ process. The time taken for this assignment, S, is essentially the time it takes for the distributor to find a *free* HARQ process.

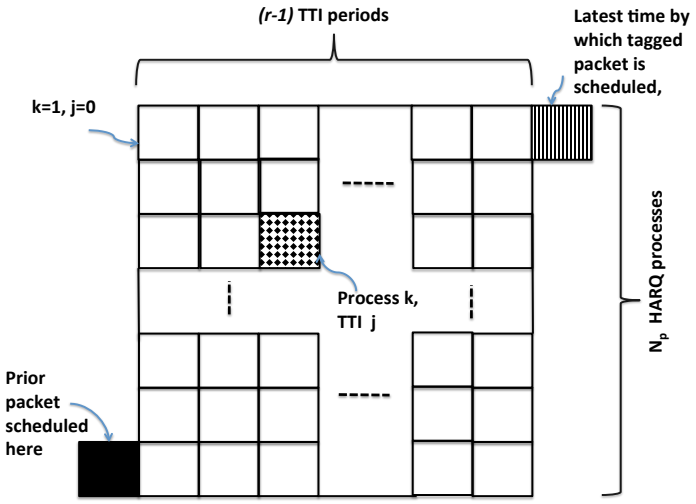


Figure 5.5: The state of the LTE HARQ processes

Consider a packet, k, that is to be next assigned to one of the HARQ processes. For k to experience the "maximum assignment time", S^{max}, all the HARQ processes must be occupied for the maximum possible duration i.e., each process must perform the maximum number of (re)transmission attempts after k reaches the distributor. S_{max}, is then, the sum of (i) one TTI for the initial transmission of packet $k-1$ and (2) the $r-1$ transmission attempts made by each of the N_p processes subsequently (each of these processes must be occupied by a packet and must have at least performed one unsuccessful attempt already; else at least one would be empty and packet k can be assigned). It is easy to see that S^{max} is thus given by: $S^{max} = (r-1)\,N_p + 1$.

To aid the discussion on the calculation of S, we refer to Fig. 5.5. The rows in the figure correspond to the HARQ processes, while the columns correspond to time in terms of TTIs. Without loss of generality, assume that the prior packet that was assigned by the distributor, was assigned to the last HARQ process viz., process N_p; in the figure this

$$\mathrm{P(S = jN}_p + k) = \begin{cases} (\sum_{l=j+2+1}^{r} \pi_l)^{(k-1)}.(\sum_{l=j+2}^{r} \pi_l)^{(N_p-k-1)}.\prod_{m=1}^{j} p_m).(\sum_{l=j+2}^{r-1} \pi_l(1-p_l)+\pi_r) & j>0, k \neq N_p \\ (\sum_{l=j+2+1}^{r} \pi_l)^{(k-1)}.\prod_{l=1}^{j-1} p_l(1-p_j) & j>0,\ k=N_p \\ ((1-\pi_1)^{(k-1)}.\pi_1 & j=0, k<N_p \\ (1-\pi_1)^{(k-1)}(1-p_1) & j=0, k=N_p \end{cases} \tag{5.5}$$

preceding packet (relative to a tagged packet that we consider) is assigned to the black TTI (we refer to each block in the figure as a "slot" from here on). At this point, the tagged packet enters the distributor.

Let us assume that the tagged packet is assigned to a HARQ process, $jN_p + k$ slots later ($j \in \{0, r-1\}$, $k \in \{1, N_p\}$). This corresponds to the shaded slot in the figure. For this to happen, the following must hold true: **(i)** The processes from 1 to $k-1$, must be occupied for $j+1$ slots; **(ii)** The processes from $k+1$ to $N-p$ must be occupied for j slots; and, **(iii)** The process k must be occupied for j slots and must be free in the $(j+1)^{st}$ slot. The probability of the tagged packet assigned as above is given by Equation 5.5; in the following, we elaborate on how we arrive at this result. For simplicity, we simply refer to π_l^{harq} (recall Equation 5.4) as π_l.

Let us first consider the simple case wherein $j = 0$. If the process to which the packet is assigned (process k) is one of the first $(N_p - 1)$ processes (not the one to which the previous packet was assigned to), the conditions that must be satisfied are (i) the preceding $k-1$ processes must have been occupied and the k^{th} process is free. The likelihood of this is $(1-\pi_1)^{(k-1)}\pi_1$. If the process in question is the last (N_p^{th}) process, the requirement is

that all the previous processes were occupied and the previous packet (transmitted in the black slot) was successful. The likelihood of this is $(1-\pi_1)^{(k-1)}.(1-p_1)$.

Next, let us consider the case where $j \neq 0$. To begin with let $k \neq N_p$. The probability of the first condition in the aforementioned list holding true for each of these processes, is essentially: $\pi_2.p_2.p_3 \ldots p_{j+2+1} + \pi_3 p_3.p_4 \ldots p_{j+3+1} + \ldots\ldots + \pi_{(r-1)-(j+1)}.p_{(r-1)-(j+1)} \ldots p_{r-1}$. Using Equation 5.4 it can be shown that this long expression is simply $\sum_{l=j+2+1}^{r} \pi_l$. Together, for the $k-1$ processes (assuming that they are independent because of varying channel conditions between transmissions from these processes, due to fading), the probability of the first event is $(\sum_{l=j+2+1}^{r} \pi_l)^{(k-1)}$.

Similarly, $(\sum_{l=j+2}^{r} \pi_l)^{(N_p-k-1)}).\prod_{m=1}^{j} p_m$ gives us the probability of the second event. The last term corresponds to process N_p to which the packet preceding the tagged packet was assigned. For this packet, it is known that a transmission attempt was made in the black slot, and the last term accounts for this.

Finally, let us consider the the last required event. Since, process k is one of the first (N_p-1) processes, this event occurs with a probability $\sum_{l=j+2}^{r-1} \pi_l(1-p_l) + \pi_r$. This essentially corresponds to failed attempts in the first j slots followed by either a packet success or a packet drop for that process (k) in the j^{th} slot.

If $k = N_p$, things are slightly different since we know when the previous packet was scheduled. Thus, the likelihood of this process being free for the first time at the j^{th} TTI is simply $\prod_{l=1}^{j-1} p_l(1-p_j)$ for $0 < j \leq r-1$.

Determining the likelihood of a packet reception in a TTI: Having characterized the arrival and the service processes, we next determine the likelihood of a packet

reception (either decodable or corrupted) in a TTI at the UE. A reception either transitions the UE to the active state or keeps it in one of the composite modes in that state.

The UE receives a packet in a TTI if the corresponding process has a packet to send. If not, there is no reception. Here, we make the following approximations. If the queue is non-empty it is unlikely that any of the N_p processes is empty and thus, the UE will receive a packet in each TTI. If the queue is empty and N of the N_p processes are occupied, the likelihood of the UE receiving a packet in a TTI is:

$$\beta = \sum_{N=1}^{N_p} \frac{N}{N_P} P(\text{No of Pkts in System} = N). \tag{5.6}$$

The probability of the MMPP/G/1 queue being non empty is simply given by:

$$\rho = \lambda_{avg} E(S) \tag{5.7}$$

Thus, the probability of the UE receiving a packet in a TTI is given by:

$$p = \rho + (1 - \rho)\beta. \tag{5.8}$$

If ρ is high, the second term in Equation 5.8 tends to zero. On the other hand, if ρ is small, the value of N and thus, β is even smaller (meaning that if the queue is empty, the likelihood that some of the processes are occupied is very small). Thus, we ignore the second part and approximate $p \approx \rho$. We later validate that this approximation is reasonable via simulations (where we don't make this assumption).

5.3.3 Impact on the LTE RRC State Machine

Next, we seek to capture the impact of p on the LTE RRC finite state machine (FSM). We model the state machine as a finite-state discrete-time Markov chain parametrized by p, the probability of receiving a packet at a given TTI. The Markov chain is shown in Fig. 5.6. Initially the chain is in the idle state. A packet reception (with probability p), results in a transition to the continuous reception state (consisting of states 1 through T_c). A state transition from a higher RRC power state to lower state occurs if no packet is received (with probability $1-p$) for T_i, where T_i reflects the timer value associated with the currently occupied RRC state. Any packet reception triggers a timer reset and the machine transitions to the continuous reception state if in any other occupied state. The transition probability matrix for this FSM, $\boldsymbol{P}$, is given by

$$\boldsymbol{P} = \begin{pmatrix} 1-p & p & 0 & \cdots & \cdots & 0 \\ 0 & p & 1-p & 0 & \cdots & 0 \\ \vdots & p & 0 & 1-p & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & p & 0 & \cdots & \cdots & 1-p \\ 1 & p & 0 & \cdots & \cdots & 0 \end{pmatrix} \tag{5.9}$$

The steady state probabilities $\boldsymbol{\pi}^{rrc}$ of being in each sub-state depicted in the figure is then computed using:

$$\boldsymbol{\pi}^{rrc}\boldsymbol{P} = \boldsymbol{P} \tag{5.10a}$$

$$\sum_i \pi_i^{rrc} = 1 \tag{5.10b}$$

To calculate the probability of each individual RRC state we can simply sum the probabilities of being in any of the component sub-states of that state (i.e., π_i^{rrc}'s that correspond to sub-states i within that state). From equations 5.10 one can compute the following:

$$\pi_{IDLE}^{rrc} = (1-p)^{T_{tail}} \tag{5.11a}$$

$$\pi_{RRC\_Connected}^{rrc} = 1 - \pi_{IDLE}^{rrc} \tag{5.11b}$$

$$\pi_{ContinousReception}^{rrc} = 1 - (1-p)^{T_c} \tag{5.11c}$$

$$\pi_{ShortDRX}^{rrc} = -(1-p)^{T_c}((1-p)^{T_s} - 1) \tag{5.11d}$$

$$\pi_{LongDRX}^{rrc} = (1-p)^{T_c+T_s} - (1-p)^{T_{tail}} \tag{5.11e}$$

where $T_{tail} = T_c + T_s + T_l$. Given the expected energy consumed in each TTI when being in each of the states of the RRC state machine, and the specifications of a video stream (resolution, slow versus fast) and its duration, one can now compute the expected energy consumed by network interface from the download.

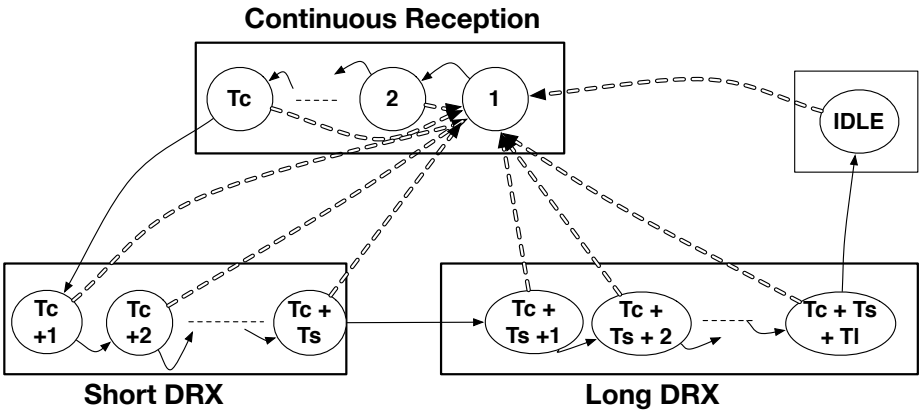


Figure 5.6: Markov chain representing the LTE RRC state machine. State 0 corresponds to the Idle state. States 1 through T_c represent the continuous reception state. Dotted and solid arrows represent transitions with probability p and $1 - p$, respectively.

Resolution Types	EGA, CGA, HD, FHD
Streaming server	Darwin
Encoding Protocol	MPEG4
Wireless Device	Samsung Galaxy s5

Table 5.2: Experimental Setup

Type	Protocol	Resolution	Fast Motion (Kb/s)	Slow Motion (Kb/s)
CGA	h.264	320x200	293	111
EGA	h.264	640x350	802	236
HD	h.264	1280x720	2433	619
FHD	h.264	1920x1080	4656	1174

Table 5.3: Details of types of videos used

Energy due to packet receptions: We wish to point out that while being in each of the RRC_CONNECTED states results in a baseline energy expenditure, there is additional energy consumed when packets are received [129]. This increase is directly proportional to the rate at which packets are received while in that state. This is especially important for the continuous state, since increased rate of packet receptions leads to this state almost inevitably. In such conditions, since ρ linearly increases with rate, the increase in energy is linearly proportional to ρ.

5.3.4 Impact of LTE Scheduling

Thus far, we assumed that the tagged UE is scheduled every TTI. However, depending on the number of users and the provider's policy, this may not be true. In between

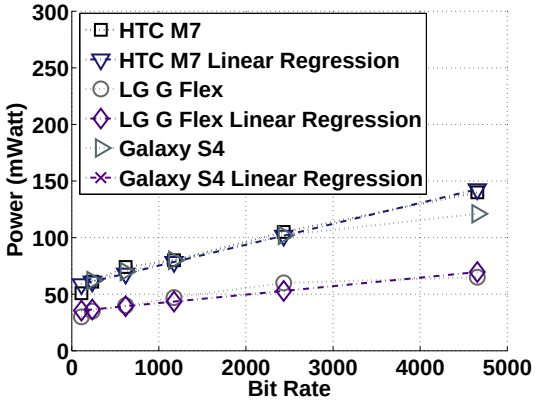


Figure 5.7: Bit Rate Vs CPU power

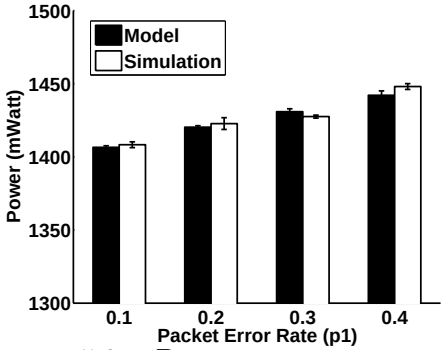


Figure 5.8: Power consumption versus PER for slow motion, low resolution video

transmission attempts, there maybe TTIs where the UE is not scheduled, and this results in an additional service time component, viz., a waiting time wherein, no process is active and the distributor is simply in a wait mode. If the UE is scheduled every W slots (W is a random variable) and if we assume that this is independent of the service time rendered to a packet[3], the expected service time is scaled up by a factor $E(W)$ due to this scheduling policy. Thus, the probability that the queue is non-empty is now $\hat{\rho} = \rho E(W)$.

The likelihood of the UE receiving a packet in a TTI depends on whether or not the UE was scheduled in that TTI; the probability of the UE being scheduled in a TTI is $q = \frac{1}{E(W)}$ (we assume that in general $E(W)$ is small; if not, the UE is not scheduled for long durations and the queue may become unstable). If scheduled, a packet is received with probability $\hat{\rho}$. Thus, the probability of receiving a packet in any arbitrarily chosen TTI is

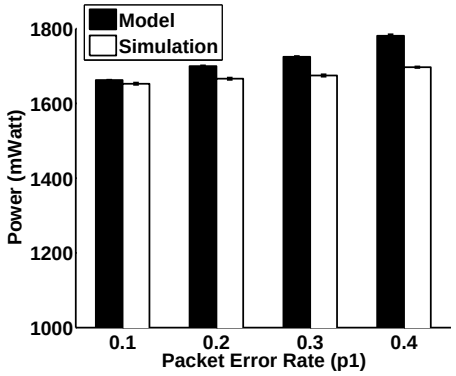


Figure 5.9: Power consumption versus PER for slow motion, high Resolution video

$\hat{\rho}.q = \rho$. Thus, in essence, our prior analysis applies in this more general case as well. We have validated this via simulations.

5.3.5 Power consumption due to processing

The power consumed at the UE includes the power due to the processing of the received frames (decoding and playing the video) in addition to the power consumed on the (LTE) network interface. Our model only explicitly accounts for the latter. Higher resolution videos are typically larger in size and thus require higher processing/computing resources (and therefore power) at the UE.

The power consumed for processing is device dependent (i.e., it depends on the CPU and battery of the device). If the bit rate of the video stream is known (can be provided by the video server), the UE can locally estimate the processing power that will be

[3]This is reasonable since the external load is independent of what the UE is attempting to download.

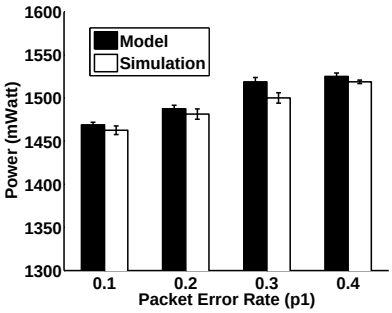


Figure 5.10: Power consumption versus PER for fast motion, low resolution video

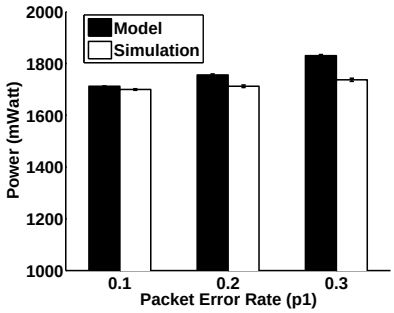


Figure 5.11: Power consumption versus PER for fast motion, high resolution video

consumed in addition to the LTE interface power discussed thus far. To show the viability of this approach, we experimentally profile the CPU power consumption for 3 different smartphones (Samsung Galaxy S5, HTC One M7 and LG G Flex) for downloading videos with different bit rates using the PowerTutor tool [212]. The results from our experiments are plotted in Fig. 5.7. We see that, for each phone, the power consumed due to CPU activities (processing/playback) increases (roughly) in a linear fashion with the bit rate; thus, the local device can easily estimate this power for different bit rate video clips once it does an initial calibration with a few video downloads.

Combining this CPU power with the estimated power consumed on the network interface (obtained using our model), we are able to get a estimate of the total power consumed by different versions of a video stream.

5.4 Evaluations

In this section, we validate our analytical framework via simulations and experiments. We consider different types of video, and vary channel conditions to understand how video downloads affect the UE energy consumption. We first describe our simulation and experimental setups, discuss how we compute energy with our model, and later discuss our results.

5.4.1 Configurations

Simulation settings: We use NS3[29] in combination with the LTE module developed by the LENA Project[20]. We use the default channel model used in this project. Details can be found in [20]. Since we are only interested in the energy consumed at the UE, we set up a simple LTE topology with one enB serving a single UE. We log all the PHY packets that the radio processes (including packets that are in error). We vary the channel conditions. We use 8 different video clips each of which is about 60 seconds long. Videos are tagged and processed using the EvalVid [16] tool and then passed on to the enB to be transmitted to the UE. For each experiment, we perform 20 simulation runs.

Experimental Setup: Our experiments are on a Samsung Galaxy S5 LTE phone over the AT & T LTE Network (we did experiments with other models and T-Mobile and the results were similar). We connect our LTE phone to a Monsoon Power Monitor [22] and measure the power when different types of video (discussed next) are being downloaded from a Darwin server. We collect power traces for each resolution of fast and slow motion

videos 10 times and estimate the average power. The details of our experimental set up are in Table 5.2.

Video: We use four popularly used video resolutions listed in Table 5.3. For each video, we translate the resolution into a bitrate, and estimate λ_{avg} as discussed in Section 5.3.

Parameters for our analytical framework: To compute the power consumed, we use our analytical model in conjunction with the results in [129]. Specifically, from [129], we use the following: (a) In the idle state a constant power of 594 mW is consumed (594 mJ of energy is consumed in 1 second). In the continuous state, $power = \alpha\eta + \beta$. Where $\alpha = 51.97$ (power/Mbps), β is the baseline power in this state and is equal to 1288.04. η is the rate of reception in Mbps (and can be computed from ρ). For the Short DRX and Long DRX states, the power alternates between the power in the idle state and an active power; this active power is approximately 1680mW (note that this is larger than the continuous reception baseline power due to the power due to switching states). For the Short DRX state the switch from idle to active periods happens every 20 ms and for the long DRX state it happens every 40ms. We also use the results from [129] for the timer values that dictate the RRC state machine transitions. Specifically, $T_c = 100\ ms, T_s = 20\ ms, T_l = 11450\ ms$. We also set the value of all the p_is to be the same as p_1 as we observed in our simulations that these did not change by much from p_1.

Processing power at UE: The power consumed at the UE includes the power due to the processing of the received frames in addition to the power consumed on the network interface. Our model only explicitly accounts for the latter i.e., the power consumed

due to the LTE interface. Higher resolution videos are typically larger in size and thus require higher processing/computing resources (and therefore power) at the UE. To estimate the energy consumption due to processing, instead of reinventing the wheel, we simply use the powerTutor tool [212]. This allows us to profile and estimate the power consumed by the CPU for the purposes and rendering the video. Combining this estimate with the estimated power consumed on the network interface (obtained with our model), we are able to get a estimate of the total power consumed by different versions of a video stream.

5.4.2 Results and Inferences

Slow motion video: In Figs. 5.8 and 5.9, we present both the analytical and simulation results for slow motion videos of the lowest and highest resolutions. We see that the analytical results match well with those from simulations. We notice that at low packet error rates (PER), there is about a 280 mW (16.9 %) decrease in power when we switch from the high resolution video to low resolution video[4]. As the PER increases, the decrease is more pronounced (19.1 %, 340mW). This is because at high PERs, there are increased retransmissions, which essentially increase the probability of being in one of the active states. Furthermore, the increase in receptions trigger power increases even when in the continuous mode due to packets in error.

Fast motion video: Figs. 5.10 and 5.11 present analogous results with fast motion video. First, again, the analytical results match well with simulation results. We notice that with fast motion video, transitioning to a lower resolution results in about a

[4]To lower the video resolution, we switch from FHD to EGA.

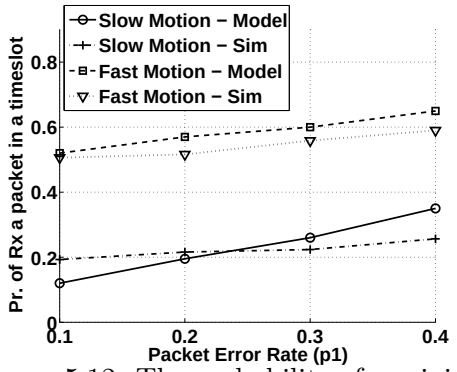


Figure 5.12: The probability of receiving a packet in a TTI (decodable or corrupted) for low resolution videos

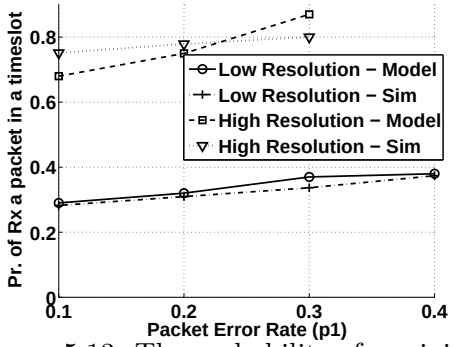


Figure 5.13: The probability of receiving a packet in a TTI (decodable or corrupted) for high resolution videos

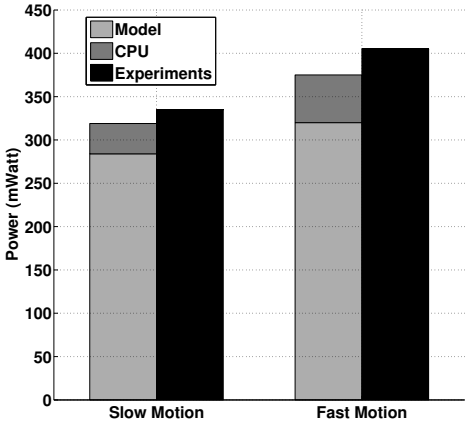


Figure 5.14: Power reduction from lowering resolution from highest to lowest level under good channel conditions: Analytical and experimental results

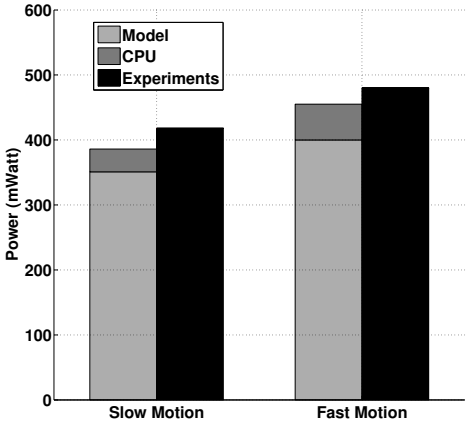


Figure 5.15: Power reduction from lowering resolution from highest to lowest level in bad channel conditions: Analytical and experimental results

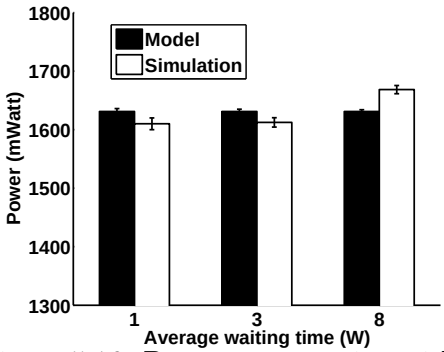


Figure 5.16: Power consumption with different expected waiting times ($E(W)$).

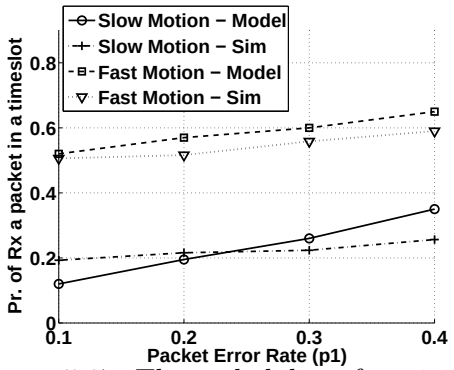


Figure 5.17: The probability of receiving a packet in a TTI (decodable or corrupted) for low resolution videos

318mW (17.8 %) reduction in the consumed power in good channel conditions. In bad channel conditions we see a 384 mW (19.8 %) decrease in power consumption. This is because, with this type of video at high resolutions the packet arrival rate is as is high; the retransmissions further increase the energy consumed. We observe here that the UE is mostly in the continuous state regardless of whether the channel condition is good or bad. Note here that at $p_1 > 0.3$, the queue became unstable and we could not gather meaningful results with the simulations.

Probability of reception in a TTI with slow and fast motion videos: In Figs. 5.17 and 5.13, we depict the probability of receiving a packet (decodable or corrupted) at the UE, in a TTI. As one might expect, the probability increases as the PER increases since there will be a higher number of retransmission attempts. Further, also as expected, this probability is higher for fast motion video, and for higher resolution videos, since the bit rates are higher. Most importantly, we see a close match between the simulation and

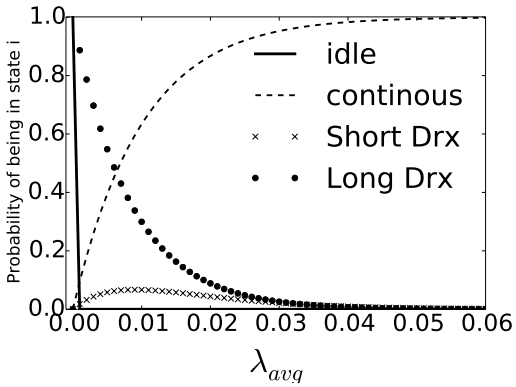


Figure 5.18: The change in state occupancy with λ_{avg}

analytical results; this shows that the assumptions made for analytical tractability do not influence the results by much.

Comparing analytical results with experimental results: In Fig. 5.14, we compare the results obtained using our framework, with that from real experiments on our Samsung Galaxy S5 LTE phone. First, we consider good channel conditions (4-5 bar coverage). Since we do not have access to the LTE PHY interface, we simply map the coverage level (5 bar) to a rough empirical characterization of the PER. Specifically, in this case, we set the PER to be 0.1 as a conservative estimate. The total power with our approach is the sum of the LTE interface power and the CPU processing power as discussed in Section 5.3.5. We observe that the results derived from our approach are fairly good estimates of the total energy consumed in reality (within 5 % of the measured energy savings). We see that the power savings are significant with both fast and slow motion video when the user chooses a lower quality version. The savings are higher with fast motion video since, there is a bigger reduction in the video bit rate ($\approx$ 405 mW) as compared to slow motion video ($\approx$ 335 mW).

	Decrease in PSNR	Bad Channel	Good Channel
Slow-Motion	10.11%	$376.2J$	$301.5J$
Fast-Motion	14.5%	$432J$	$364.5J$

Table 5.4: The energy saved as resolution is changed from high to low with the corresponding decrease in PSNR

In Fig. 5.15, we plot the results with bad channel conditions (1 bar coverage). Here we empirically choose the highest PER that we could tolerate (in our simulations) without the queue becoming unstable (0.39) when we use our model. The total power saved is the sum of what is saved on the network interface and due to processing. We observe again that our results once again match very well with the results from the experiments (within 5% of the experimental results). We find that the savings increase to $\approx$ 480 mW and $\approx$ 418 mW respectively, for fast and slow motion video, potentially due to a decrease in the number of retransmissions.

Finally, in Table 5.4, we show the decrease in PSNR that comes with a corresponding decrease in energy for 15 minute long videos. Interestingly, a small decrease in PSNR (49.5 dB to 44.5 dB for slow motion and 48 dB to 41.5 dB for fast motion, which corresponds to 10.11 % and 14.4 % in the two cases, respectively) results in considerable energy savings (e.g., 376 J for slow motion video and 432 J for fast motion video in bad channel conditions). The reason for only a slight reduction is that the PSNR decreases only due to a reduction in resolution; due to TCP there are no losses that degrade quality on the channel itself. Thus, by lowering the video quality slightly, the user can conceivably gain significant energy savings.

Impact of LTE scheduling: In Fig. 5.16, we show the impact of scheduling the UE once every W TTI on average. W is chosen as per a uniform distribution between 1 and twice the average value. We observe that the power consumed does not change by much with varying W. This validates our analysis in Section 5.3.4.

Times spent in each of the LTE states: In Fig. 5.18 we show the probability of being in each LTE energy state. We see that if λ_{avg} increases beyond an extremely low value, the UE is almost never in the IDLE state. As one might expect, as λ_{avg} increases the likelihood of being in the continuous mode increases, while the likelihood of being in the Long DRX state decreases. To begin, the likelihood of being in the short DRX state increases; here the time is shared between this state and the continuous state. But after a certain point, the probability of being in this state decreases and the UE is almost always in the continuous state.

Power consumption decreases with resolution: For completeness, we present a plot wherein we show the variations in the consumed power with fast and slow motion video as we vary the quality in terms of resolution. We assume $p_1 = 0.1$. The difference in power consumption between FHD and CGA are as reported earlier. Interim resolutions offer different trade-offs between power and video resolution.

5.5 Related Work

Almost all of the power models for LTE are empirically derived. In [129], the authors empirically derive a power model based on an experimental study of LTE performance. However, the work is mainly intended for developers and does not provide an

understanding of how the energy consumption varies with different types of downloads, or in varying channel conditions. In [181], the authors experimentally characterize how variations in signal strength can affect UE power consumption. They develop an approach to schedule communications during periods of strong signal strength. However, they do not account for traffic characteristics (video) on RRC state transitions. The authors in [101] study the impact of signal strength on battery drain and evaluate various energy saving schemes for 3G networks; however, the impact of traffic patterns is not considered.

The authors in [131] present a model of the LTE radio interface. In [79], the authors optimize the LTE timers to save energy. Zhou et al., [215] investigate the trade off between saving power and wake up delay. In [139] the authors investigate the effects of varying the LTE parameters on user experience. However, none of these efforts are focused on video, nor do they capture the interactions between the video characteristics and the LTE energy states. In [102], the authors model the LTE HARQ process as a Markov chain to evaluate the energy implications of the HARQ process but the model does not capture the characteristics of video traffic or how the LTE scheduler influences transmissions.

Some of the work that looks specifically at energy consumption due to video transfers over LTE (e.g., [189]), target the optimization of the LTE timer values to reduce energy; they cannot be directly applied to determine energy savings due to lowered video quality. In [205], the authors study how YouTube traffic affects energy. He et. al. present a model for saving energy for videos streamed over a wireless link in [120]; however, their work focuses on video encoding and does not address radio power. Lu et.al [150] try to minimize transmission power based on the state of the wireless channel. Mohapatra et. al.[159] identified

several architectural techniques which can be coupled with OS level approaches to save CPU and memory energy while streaming video. In [76] the authors model the interdependence between different video packets to determine the optimal retransmission scheme for HARQ processes but do not consider the energy implications.

There is other work, where video delivery is optimized for quality (e.g., [165] and citations therein). These efforts do not explicitly consider energy due to video streaming.

5.6 Discussion

Adaptive bit rate video: Current adaptive bit rate streaming technologies such as Adobe and Silverlight (used by Youtube and Netflix, respectively) are relatively simple. They work as follows. The server stores different bit rate versions of the same video. When the client requests a video, the server sends the client a list of different bit rate videos it has in its storage. The client then makes decisions on what bit rate video it wants to download on a per chunk basis, where a chunk is simply a small portion of the video [161, 61, 193]. Chunk sizes (in terms of viewing time) can range anywhere from 2 seconds to 10 seconds. The default bit rate is applied to the first chunk and usually corresponds to the maximum available bandwidth. Clients select the highest bit rate that is supported by their bandwidth [132]. This bandwidth is estimated by looking at the time it takes to download a chunk and the size of that chunk [132]. Current adaptive bit rate clients try to maximize bit rate given this single constraint on bandwidth. We argue that energy is a concern for clients and our model allows this factor to be taken into perspective.

Our experiments show that transferring a high-resolution video over a channel with high PER results in very high energy consumption. Bandwidth aware clients could still request this video because it can still be transmitted over the channel (bandwidth suffices). However, the user may not want to spend that much energy. Further, while PER affects the bandwidth of the link, other factors such as load or how busy the enB is at that time, could be arguably more important in determining the bandwidth available for a session as shown in [88].

Our model applies to adaptive bit rate videos; the discussion was focused on fixed bit rate videos for the purposes of clarity. Some of the experiments compute energy savings assuming a fixed bit rate video, again for the purposes of making things clear. We point out that our results, in particular Figs. 5.8 to 5.11, demonstrate the effectiveness of our model in predicting power consumed at short time scales. The same experiments also demonstrate the accuracy of our model across different bit rates. Recall that two primary inputs to our model are PER and video bit rate. Power estimates can be made as often as necessary whenever one of the inputs changes.

An energy aware adaptive bit-rate scheme would make similar chunk wise decisions as its non-energy aware counterpart. The client now has an added constraint, namely power. The UE can estimate the power consumption with any bit rate video given the state of the channel using our model. The client would now estimate (a) the highest bit rate video given its bandwidth constraints (say $Bitrate_B$ and (b) the highest bit rate video given an energy constraint ($Bitrate_E$). It would choose the bit rate that satisfies both these constraints i.e., min($Bitrate_B$, $Bitrate_E$).

Impact of buffering: One can conceive a simple model which assumes that the entire video is buffered at the server and then downloaded continuously. It is then displayed to the user. In such a case, there will be no RRC state transitions (as discussed later, it will be in a continuous reception state). Now, one might argue that one can calculate the time for which the interface is active provides an indicator of the energy spent. One could proportionally increase this time based on the perceived packet error rate.

In practice however, this model is insufficient because of two important characteristics. First, studies have shown that major video streaming services (e.g., Youtube and Netflix) do not buffer too much. The buffering is often limited to a couple of seconds [161]. Since clients often terminate videos prematurely, the service providers are concerned with wasted bandwidth and for this reason never buffer more than what is necessary. Netflix clients sometimes buffer up to 2 minutes because Netflix hosts videos that are often hours long and some extra buffering is supported in case clients want to skip ahead.

Because of the above reason, it is quite possible that as network conditions change, the server side buffer becomes temporarily empty (and thus, the video is not continuously downloaded) which could trigger RRC state transitions. Our model is generic enough to take care of such cases; it is also applicable to the case where the server has a steady stream of video packets for a client for continuous downloads.

Second, We also point out here that the video stream is not necessarily transferred at the bit rate specified at the server. If that was the case, one could simply use a linear model to determine how the energy would vary with bit rate. The video traffic is shaped not only because the server may have varying loads (and thus transfers video at the specified bit

rate on average, but in a bursty manner), but also because of retransmissions on the wireless channel due to varying channel characteristics. Thus, it is inevitable that the packets will experience some form of queuing at the LTE server.

In summary, our model applies in cases where there is no buffering, any level of buffering or in the extreme case where a client buffers the entire video prior to playback. The video in question will still have to be transferred over an error prone wireless channel (which will result in queuing and retransmissions). If a client is allowed to buffer the entire video clip, all packets from the clip are inserted into the server buffer i.e., that buffer does not become empty until the entire video is transferred). This simply translates to very high values for λ_1 and λ_2. Our model can be still applied and the power consumption for different qualities can be calculated accordingly.

Motion in a video: Our framework accounts the type of dynamics in a video stream (slow or fast motion). For ease of discussion, we have assumed that the video is homogeneous in this regard. However, a video may transition between periods of fast motion to slow motion and vice versa. Tools such as AForge [4] can estimate the expected durations of fast and slow motion in such cases; with these estimates (done over chunks of buffered video at the server side), the expected energy savings can be easily computed with our framework.

Mobility If a UE is moving about a lot, this will cause the state of the channel to change (PER will change). In such cases, our model can simply recalculate power consumption with the new PER (can be potentially measured as videos are downloaded). However

when a UE moves out of the coverage area of an enB an LTE handover will be necessary. Our model does not capture this characteristic.

Chapter 6

Efficient Transport of Data Summaries over Named Data Networks

6.1 Introduction

With the emergence of Internet of Things (IoT) and dissemination of online social content, sources of various kinds continuously generate streams of data. The number of such sources is growing continuously, leading to an exponential growth in the available data for a consumer [96, 93]. In fact, consumers may experience a data deluge leading to information overload if they get all the data pertaining to a subject of interest [155]. One way to cope with this data deluge, is via data summarization services which enable clients (whether humans or computer systems) to retrieve samples with user-specified granularity. These

representative samples, or summaries, can then be used in analysis and decision making processes.

To exemplify the above, consider a smart city scenario [81] wherein sensors continuously gather data about traffic conditions. On their path to the destination, smart cars contact road infrastructure hot-spots for updates. In this scenario, collected data may have significant redundancy, local data at different repositories have semantic overlap, and network conditions are diverse. Consumers are likely to be interested in receiving content with varying granularity of detail as quickly as possible. As a second example, consumers interested in getting a sample of top stories from a variety of news media may be interested in quickly retrieving only an overview, based on which they may then choose to get more details only on certain stories. Data summarization can be an effective solution in delivering the right level of detail to consumers.

There are a set of challenges that will need to be addressed in order to deliver summaries from a set of producers to consumers. First, retrieving summaries from different producers independently will create redundant content, thus wasting communication resources and defying the purpose of summarization. Solving this problem requires the efficient creation of a global representation of the content available at the different producers. Furthermore, the network must be able to match a consumer's request with what is available at the various producers and retrieve the proper representative samples. In many applications, consumers are not interested in the source of the content or where it is, but rather the content itself and how fast it can be retrieved. Finally, since the network conditions between the consumer and the plurality of producers can be diverse (varying

bandwidth and link delay), the transport framework has to be intelligent to retrieve the content from the "best" producer, i.e., the content that fulfills the consumer's requirement with the best performance (e.g., minimum latency).

In this chapter, we develop *NEST*, a transport framework which efficiently retrieves an extractive summary of a dataset distributed across multiple connected producers to requesting consumers, with low latency. Our framework is developed on top of the Named Data Networks (NDN) infrastructure, an implementation of the Information Centric Networks (ICN) paradigm. NDN is a pull-based network architecture which supports the forwarding of content from producers to consumers using *hierarchical names*. It offers a way to seamlessly map content to interests and thus, we argue that it is natural to leverage its abilities towards achieving the efficient transport of content summaries from a diverse set of producers to consumers. Using NDN, *NEST* allows producers to converge to a *common namespace*, wherein objects that are very similar are *named* similarly, linking the summarization problem to NDN's name-based forwarding. *NEST* is designed as an end-to-end protocol that runs at the hosts and does not require changes to the underlying NDN infrastructure.

NEST first creates a global hierarchical representation of the dataset by synchronizing the producers' local datasets with minimal overhead. Subsequently, this representation is used to generate an ordered list of names that is sent to interested consumers to guide them in retrieving content summaries of varying granularity. In this list, object names are "constructed" such that, from a set of similar objects at different producers, an object is seamlessly returned from the producer with the most favorable network conditions for each

request based on the ordered list, thereby achieving minimum latency. In building *NEST* we make the following key contributions:

- We develop a distributed synchronization algorithm that capitalizes on recent advances in distributed clustering to create a global view of the shared dataset, towards realizing summaries of varying granularity.
- We develop interest-name design rules that automatically and opportunistically adapt to varying network conditions to minimize latency in delivering summaries to consumers over the underlying NDN.
- We implement *NEST* on Mini-NDN, a network emulator for NDN. We then perform extensive evaluations using datasets collected from Twitter. Our results show that *NEST* exploits producer diversity to reduce latency by up to 100% compared to baseline summary transport strategies that retrieve specific data objects.

6.2 Background

ICN has become popular recently for presenting an alternative and future architecture for the Internet as it becomes more content-centric rather than host-centric, and NDN [213] is one typical implementation. In NDN, consumers send interest messages requesting specific content using hierarchical names, where one data message is returned for each interest message. The interest is generated by a consumer to indicate that she seeks to retrieve a matching object. Partial prefix matching is used when checking whether a named object matches an interest name. In addition, intermediate routers employ multi-path forwarding

rules to pass interest messages to the next hop until it reaches producers of the named content. Producers then return data messages where the payload is the data object with a matching name. Data messages are forwarded along the *reverse paths* corresponding to that taken by the interest message. Whenever a router receives a data message, the entry for the corresponding interest is removed from its Pending Interest Table (PIT) after it is forwarded. Any subsequent data messages for the satisfied interest are suppressed, i.e., not forwarded. If caching is enabled, intermediate routers keep a copy of the data messages for a specified period of time, and returns it for subsequent matching interests from any consumer. A key feature of the interest message format is the exclusion option; consumers use this optional field to specify object name suffixes that they do not want to retrieve for a given name prefix in the interest message.

Recently, [145] was proposed as a summary transport protocol for NDN in which an ordered names list is created from a hierarchical tree representation of a dataset. The structure of the tree is such that objects sharing a longer name prefix have more semantic overlap. Thus, when a sample (i.e., summary) of the content under the tree is requested, returning objects in a shortest-shared-prefix-first order minimizes information loss (relative to retrieving all data) over all different orders of a given sample size by reducing semantic redundancy. As more data objects are transported according to this order, finer granularity details about the topic are retrieved. However, only one producer was considered. In data summarization applications in which clients are interested in a sample of the dataset distributed across multiple repositories, dynamic network conditions cause clients to experience very different network delays relative to the different producers who generate the samples.

In addition, retrieving redundant content from different producers creates summaries with poor quality. Thus, when multiple data producers share similar data objects, opportunities to improve latency performance by retrieving *any* of a set of similar objects are created. However, to do so, a flexible and adaptive data transport scheme is needed. The novelty of *NEST* is that it allows consumers to realize the advantage of *producer diversity* to retrieve summaries with minimum latency while requiring no changes to the underlying NDN architecture.

6.3 System Design

NEST comprises two main functional components viz., (a) Producer Synchronization (ProdSync) and (b) Producer Diversity Summary Transport (*PDST*) protocol. The first component creates a hierarchical representation of the dataset shared by multiple producers, while the second component manages the transport of data objects between producers and consumers on the NDN. In the following, we discuss the design details of each of the two components.

6.3.1 Producer Synchronization

In many applications, data is collected from sensors and cached at a connected set of repositories (i.e., producers) for further processing and dissemination to consumers. Since, transporting objects from individual producers to consumers independently can result in performance penalties and wasteful transfers, summarization inherently requires synchronization between producers. Thus, the first challenge in efficiently delivering sum-

maries from the set of producers, which can of the order of tens, to consumers is to derive a global view of the available data. However, for large datasets, it is not practical to transmit local datasets or large samples thereof to a centralized location for processing.

To overcome this challenge, we develop ProdSync, an iterative distributed clustering algorithm in which producers exchange meta-data of clustering solutions and samples from their local subsets of the dataset while incurring minimal communication overhead. This enables the construction of a tree representation in which each producer maintains information about the position of their local data points in the tree. This representation also enables the producers to construct an ordered List of Object Names (LON) which is then used to guide the delivery of summaries to consumers.

Notation: In the following, we introduce notation that will help in the description of ProdSync. Suppose we have a set of connected producers $\mathcal{N}$, of size N. Let the global dataset be denoted by $\mathcal{P}$, where $\mathcal{P}_i \subset \mathcal{P}$ is the local subset corresponding to producer i. Suppose $\mathcal{L}$ is the set of tree node labels on $\mathcal{T}$, and let $r \in \mathcal{L}$ be the root node. Each node $l \in \mathcal{L}$ will hold a subset of $\mathcal{P}$, denoted as $\mathcal{P}^l$. We also define $\mathcal{P}_i^l = \mathcal{P}_i \cap \mathcal{P}^l$ and $\mathcal{N}^l = \{i \in \mathcal{N} : \mathcal{P}_i^l \neq \phi\}$. In each iteration l, each producer i solves an instance of k-means clustering and sends local information $\mathcal{S}_i^l$ (to be made precise) to a designated coordinator n_l, where the coordinator for tree node r, i.e., n_r, is called the root coordinator. Finally, let the distance measure between a pair of data points p and q be given by $d(p, q)$.[1]

Algorithm details: In Alg. 3, we outline the ProdSync algorithm. The algorithm solves a distributed clustering problem over the datasets $\mathcal{P}_i, i \in \mathcal{N}$ to generate a hierarchical

[1]Vector space representation of data as well as similarity measures vary depending on application and data type. While we use specific representation in Section 6.5, the effect of these on the clustering quality is of separate interest and is beyond the scope of this work.

Algorithm 3 ProdSync

Input: Similarity threshold τ, set of producers $\mathcal{N}$
1: Initialize: $\mathcal{L} = r, \mathcal{N}^r = \mathcal{N}$ **repeat**
| P
  until
2: ;
  ick a tree node $l \in \mathcal{L}$
3: Select coordinator $n_l \in \mathcal{N}^l$
4: Solve local clustering problem on $\mathcal{P}_i^l$
5: Exchange local clustering costs c_i^l
6: Coordinator collects samples $\mathcal{S}_i^l$ from all producer $i \in \mathcal{N}^l$
7: Coordinator solves global clustering on $\cup_i \mathcal{S}_i^l$
8: Coordinator delivers solutions $\mathcal{G}^l$ to all producers in $\mathcal{N}^l$
9: Producers send coordinator local tree info $\mathbf{u}_i^l$
10: Add new nodes Update $\mathcal{T}, \mathcal{N}^l, \mathcal{L}$: $\mathcal{L} \leftarrow g \; \forall g \in \mathcal{G}^l$
11: $\max_{p,q \in \mathcal{P}^l} d(p,q) < \tau$
Output: Hierarchical tree representation $\mathcal{T}$

representation $\mathcal{T}$ for the shared dataset $\mathcal{P}$, capturing similarities between data points in a hierarchical form. For each node l in the tree $\mathcal{T}$, a coordinator is first selected, [2] which collects information about local clustering solutions from all producers in $\mathcal{N}^l$. The coordinator then solves a global clustering problem on the samples collected in $\mathcal{S}_i^l \; \forall i \in \mathcal{N}^l$ and then shares the solution with other producers. Each producer then updates the coordinator with meta-data $\mathbf{u}_i^l$ that helps in creating the tree representation of the dataset and names for objects. Specifically, each producer sends *cluster membership counts* based on the global solution. Iterative clustering stops when distances between all points in a tree node are less than a given similarity threshold τ.

To optimize the amount of data exchanged between producers, we consider a recently developed distributed clustering algorithm [77] based on the construction of ϵ-coresets [119]. It guarantees a bounded clustering cost relative to the centralized solution, at the minimum communication cost, where the clustering cost is the sum of squared distances

[2] We defer a discussion of how we implement coordinator selection to Section 6.4.

between each point in the dataset and its corresponding cluster center. This protocol was later shown to be communication-optimal by the authors in [92]. We note that the communication-optimality metric used in [92] is the number of data points exchanged between the producers in the network. However, this metric is also correlated to the convergence time of the ProdSync algorithm; the more messages are exchanged between producers, the more time it takes for ProdSync to converge. In the following, we briefly discuss the details of this algorithm.

*O*ptimal distributed clustering: For every node on the tree $l \in \mathcal{L}$, given the set of producers $\mathcal{N}_l$, and a designated coordinator n_l, the optimal clustering algorithm constructs a set of cluster centers $\mathcal{G}_l$ at the coordinator, representing all data subsets $\mathcal{P}_i^l \forall i \in \mathcal{N}_l$. Specifically, the algorithm distributes the construction of an ϵ-coreset of the shared dataset $\mathcal{P}^l$ where each producer first solves a k-means clustering problem on the local dataset $\mathcal{P}_i^l$ and then non-uniformly samples $\mathcal{P}_i^l$ based on the clustering costs c_i^l collected from all other producers $i \in \mathcal{N}^l$. In this sampling, a point with higher cost is sampled with higher probability. Each producer constructs its local portion of the ϵ-coreset, $\mathcal{S}_i^l$, which consists of the samples and their corresponding weights. The samples $\mathcal{S}_i^l$ are then collected at the coordinator and the global clustering problem is solved on the weighted samples $\cup_i \mathcal{S}_i^l$.

Complexity analysis: At each iteration $l \in \mathcal{L}$, the *optimal distributed clustering* requires two rounds of message exchanges between the coordinator and other producers, where first the costs of local clustering solutions are collected by the coordinator and then the sum cost is shared with all producers. The communication overhead is $O(Nm)$, where m is the number of edges on the graph connecting the set of producers $\mathcal{N}$.

To process a node $l \in \mathcal{L}$, each producer i solves an instance of k-means clustering problem on the local dataset $\mathcal{P}_i$ (Steps 4 and 7). This problem is an NP-hard problem [64]. However, there exist efficient approximations such as the Lloyd's algorithm [149], with time complexity $O(|\mathcal{P}_i|ksw)$, where s is the dimensionality of vectors representing the data points and w is the number of iterations needed for convergence. It was shown that, in practice, k-means converges in linear time with respect to the number of data points [69]. . The number of nodes on the tree, i.e., $\mathcal{L}$, can vary between $O(\log_k |\mathcal{P}|)$ to $O(|\mathcal{P}|)$. In practice, we pipeline the processing of tree nodes such that communication delay does not contribute a purely additive component to the total processing time. We study this in detail in Section 6.5.

6.3.2 Producer Diversity Summary Transport

In this section, we develop an efficient transport protocol, called Producer Diversity Summary Transport (*PDST*) that takes the output tree $\mathcal{T}$ from ProdSync, transforms it to an ordered LON that is delivered to interested consumers. Consumers request representative objects based on the LON, and *PDST* delivers data objects from producers to consumers with minimum latency, where latency is defined as follows.

Definition 1 *The transport latency $T(i)$ corresponding to a given interest message i is total time delay between sending the interest message and the reception of the corresponding data message.*

Fix an interest-name i and let $\mathcal{N}(i) \subset \mathcal{N}$ be the set of producers with data objects similar to i. Let $T_j(i)$ be the transport latency when data is returned from producer j. The objective

of *PDST* is to minimize the latency in retrieving data objects, whenever *similar* objects are available at multiple producers, i.e., achieve $T^*(i) = \min_{j \in \mathcal{N}(i)} T_j(i)$. While doing so, the protocol must adapt to dynamic network conditions, and specifically varying link delays.

Satisfying the minimum transport latency and adaptability to dynamic network conditions, are challenging problems for multiple reasons. First, it is undesirable that consumers maintain state information for all available producers based on the history of received data. Moreover, explicitly and continuously measuring transport latency for objects from different producers will incur non-negligible communication overhead. The novelty of *PDST* is that it achieves the aforementioned goals by crafting interest messages in a format that capitalizes on the features of NDN. Specifically, *PDST* exploits NDN's forwarding characteristics viz., multi-path and partial prefix match forwarding. It also leverages the fact that intermediate routers suppress multiple data messages retrieved in response to a single interest message and only forward the first match. *PDST* does not require any modifications to NDN and is only run at the producers and consumers as an application.

PDST achieves minimum transport latency and adapt to varying network conditions using three different processing steps at the different participating network entities. Below, we discuss the different steps of *PDST* in detail before we formalize the result.

Root-coordinator-side *PDST*

First, names of all data objects at the leaves of the tree are automatically created. In particular, during clustering, tree nodes are given labels (e.g., '0' for the left branch and '1' for the right branch in 2-means clustering), and data objects at the leaves are named by concatenating label names from the tree root node to the leaf, similar to [144]. However,

unlike [144], the root coordinator does not have actual data points from all other producers, but rather the *counts* of data objects under each leaf from each producer. This information is collected in Step 9 of Alg. 3. Thus, the root coordinator can now create a representative tree using the counts of data points at each branch from each producer, as described in the following example.

Suppose producer a and producer b have two and three data items, respectively, under a tree leaf with the prefix $v = /r/0/0/1/0/1$. Now, the root coordinator can simply add items under this leaf as $/r/0/0/1/0/1/a0$, $/r/0/0/1/0/1/a1$, $/r/0/0/1/0/1/b0$, $/r/0/0/1/0/1/b1$, $/r/0/0/1/0/1/b2$. Based on the user-specified and application-dependent similarity measure, items under v are deemed similar. At the same time, each of the producers a and b, will fix some order for their local data objects, based on user-specified weights corresponding to each data object (e.g., content popularity, freshness, etc). Note that while the local ordering of data objects at each producer can be used to send higher-weighted objects first, the relative ordering between data objects at different producers is not needed at the root coordinator. This is because our design gives more priority to improving transport latency by retrieving the next data object (based on local weight-based ranking) from the producer with the best network conditions.

The next step for the root coordinator is to process the data object names such that a special symbol ($\tilde{}$) is concatenated at the end of all object names under any tree leaf where data objects belonging to multiple producers exist. Thus, for leaf v, the object names will be processed to be $/r/0/0/1/0/1/a0\tilde{}$, $/r/0/0/1/0/1/a1\tilde{}$, $/r/0/0/1/0/1/b0\tilde{}$, $/r/0/0/1/0/1/b1\tilde{}$, $/r/0/0/1/0/1/b2\tilde{}$. This symbol will later be used by the consumer to

construct interest messages that allows retrieval of data objects from the producer with the minimum transport latency.

Given the hierarchical tree representation $\mathcal{T}$ created using ProdSync as described in Section 6.3.1, the root coordinator can transform $\mathcal{T}$ into an LON by traversing $\mathcal{T}$ from the root to the leaves and returning the name of the object at the leaf. During traversal the branches are selected such that an object with the shortest-shared-prefix, with respect to previously returned object names, is returned.

Finally, this processed list is sent to the consumers upon request. In particular, when a consumer requests a summary of a dataset under the prefix $/r$, the root coordinator will send a data message with the LON for data objects under the corresponding tree. Note that the LON is generally of much smaller size compared to data objects (e.g., in social media, a tweet could have an image or video object embedded in it).

Consumer-side *PDST*

Denote any last level subtree of $\mathcal{T}$ with objects from multiple producers, as an *opportunity subtree*. Each consumer running the *NEST* application will first request the LON under some tree root $/r$. The consumer sends interests for items in the LON in the given order. However, the interest names used will vary depending on whether the object belongs to an opportunity subtree (which is marked by the special symbol ˜). For such objects, a *producer diversity opportunity* exists and thus the consumer can take advantage of it. In particular, for any object name in the LON ending in ˜, the consumer sends the interest message with the partial name up to the label of the corresponding parent node of

the object in $\mathcal{T}$. For example, if the next data object name in the LON is $/r/0/0/1/0/1/b0\hat{}$, the consumer sends the interest message $/r/0/0/1/0/1/$, instead.

It is easy to see that this interest will be forwarded by the underlying NDN to all producers with data objects under the corresponding opportunity subtree. Thus, all producers will respond with data objects from the opportunity subtree, and only one data message will be forwarded to the requesting consumer while other messages will not be forwarded. To avoid retrieving duplicate objects that were retrieved previously using the same partial name, the consumer employs the *exclude* option in the interest message. In particular, it includes the last component in the name of the objects retrieved previously. For example, when an interest message is sent with the partial name $/r/0/0/1/0/1/$ and data object $/r/0/0/1/0/1/b0$ is retrieved, the next interest message for an object belonging to the same opportunity tree will be $/r/0/0/1/0/1/(-b0)$.

One of the main advantages of crafting the interest messages as described above is that the framework automatically adapts to changing link delays. Thus, two consumers sending the same interest message will get potentially different (but semantically similar) data objects from different producers. Furthermore, as network conditions change over time, a consumer may get data objects from other producers because of latency advantages. Since *PDST* capitalizes on NDN forwarding rules, it also works when caching at intermediate routers is enabled without the need to modify software running on them. Specifically, caches will also use partial prefix matching and return objects with matching names unless the last name component is included in the exclude field of the interest message. If no matches are found in the cache, the interest will be further forwarded.

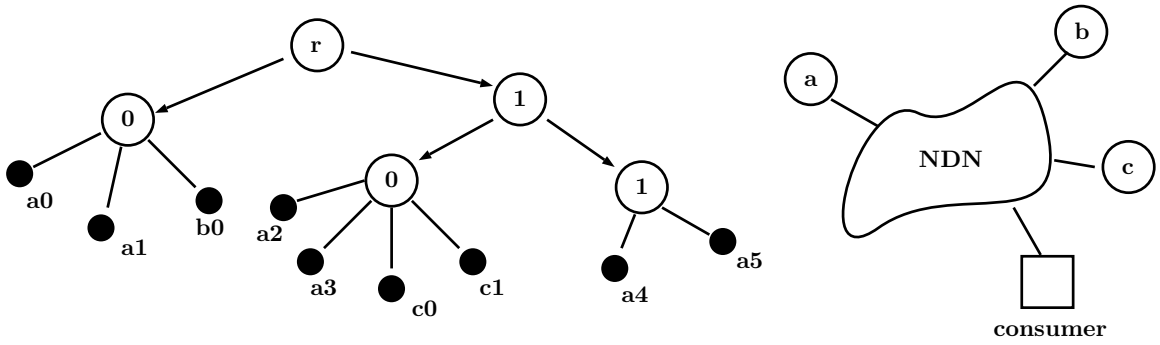


Figure 6.1: Example network and tree with three producers (a,b,c) and a consumer.

Producer-side *PDST*

On the producer side, each producer will maintain an ordering of the local data points based on user-specified and application-dependent weights. For example, in a social media application, the popularity of the content could be used as the weight. In a sensor network application, more recent events or measurements could be weighted highly if freshness is desired. Whenever the producer receives an interest message with a partial name, it responds with a data object from the subtree specified by the name prefix, returning the first object in order after the excluded objects. For example, if the interest message received by producer b is $/r/0/0/1/0/1/(-b0)$, then it returns object $/r/0/0/1/0/1/b1$. Note that the object name $b1$ might not be the actual object name at producer b, but rather a pointer to a data object under the given prefix which is second in order based on the weights. This order is maintained locally by each producer.

In Table 6.1, an example LON is shown. The corresponding network with one consumer and three producers, as well as the hierarchical data representation $\mathcal{T}$ are shown in Fig. 6.1. In the example network, the transport latency to producer c is the lowest, then

	NEST's LON	Interest name sent	Data name received
1	$/r/0/a0^{\tilde{}}$	$/r/0/$	$/r/0/b0$
2	$/r/1/0/a0^{\tilde{}}$	$/r/1/0/$	$/r/1/0/c0$
3	$/r/0/a1^{\tilde{}}$	$/r/0/(-b0)$	$/r/0/a0$
4	$/r/1/1/a4$	$/r/1/1/a4$	$/r/1/1/a4$
5	$/r/0/b0^{\tilde{}}$	$/r/0/(-b0, -a0)$	$/r/0/a1$
6	$/r/1/0/a3^{\tilde{}}$	$/r/1/0/(-c0)$	$/r/1/0/c1$
7	$/r/1/1/a5$	$/r1/1/a5/$	$/r/1/1/a5$
8	$/r/1/0/c0^{\tilde{}}$	$/r/1/0/(-c1)$	$/r/1/0/a2$
9	$/r/1/0/c1^{\tilde{}}$	$/r/1/0/(-c1, -a1)$	$/r/1/0/a3$

Table 6.1: Example of *PDST* operation.

producer b, and then producer a. The second column is processed by the root coordinator in *NEST* and represents the LON delivered to consumers requesting a summary of content under the prefix $/r$. In the third column, the interest names that the consumer sends are shown. The names of the data objects received by the consumer in response, are shown in the last column. Note that the interest names in the third column are adapted based on names of data objects received so far (i.e., from previous rows), as listed in the last column. For example, consider row number 6. Here, the interest sent is for a data object that is under the prefix $/r/1/0/$. Since the consumer previously received the object $/r/1/0/c0$ (in row 2), it now includes $c0$ in the exclude field of the interest message. The NDN forwarding will pass the interest message $/r/1/0/(-c0)$ to all producers, but it will reach producer c first since it has the best network conditions with respect to the consumer. Now, producer c will check its local dataset for data objects under prefix $/r/1/0/$ and with rank order subsequent to object $c0$, returning object $/r/1/0/c1$. Data objects returned from other producers will then be suppressed by intermediate routers since the interest message would have been already satisfied by object $/r/1/0/c1$.

We formalize our main result in the following.

Proposition 2 *Fix an interest-name i. PDST achieves minimum latency $T^*(i)$.*

Proof. We only consider $|\mathcal{N}(i)| = 2$, zero processing delays and no caching. Extending the proof to other scenarios is straightforward but cumbersome. Suppose p_j is the minimum-delay forward path to producer j, which is composed of m_j links, with instantaneous link delays $x_e(t_{ef}), e \in \{0, 1, \cdots, m_j\}$. Here, t_{ef} is the time instant the interest message is transmitted on link e in the forward direction. Suppose paths p_1, p_2 coincide in the first v links. Thus, we have $T_j(i) = \sum_{e=0}^{v-1} x_e(t_{ef})^{p_j} + \sum_{e=v}^{m_j} x_e(t_{ef}) + \sum_{e=0}^{v-1} x_e(t_{eb}) + \sum_{e=v}^{m_j} x_e(t_{eb})$, for $v \in \{0, 1, \cdots, \min(m_1, m_2)\}$. Note that backward path is identical to the forward one since data messages are routed to the consumer on the reverse path of the corresponding interest message. It can be seen that $T_1(i) - T_2(i) = \sum_{e=v}^{m_1} x_e(t_{ef}) + \sum_{e=v}^{m_1} x_e(t_{eb}) - (\sum_{e=v}^{m_2} x_e(t_{ef}) + \sum_{e=v}^{m_2} x_e(t_{eb}))$, which is the difference between the round trip delay from the vertex (router) v^* at which paths split to producers. Now, when $T_1(i) - T_2(i) > 0$, the data message from producer 2 arrives first at v^* and is the only message forwarded to the consumer, implying that $T(i) = T_2(i)$, concluding the proof. ■

We note that Proposition 2 implies that *PDST* minimizes latency even if the link delay varies while the interest or data message has not been received at the destination.

Pipelining interests: PDST uses an adaptive pipelining window, which controls how many pending interests are allowed at any given time. In addition to being limited to a maximum size W, the window size is adapted based on the LON and the progress made thus far in processing the list. In particular, the consumer can send interests from the LON until a new entry requires sending a partial name which is *already* used in a pending interest,

or until the maximum window size is reached, whichever is smaller. This design prevents retrieval of duplicate objects, since names of previously retrieved objects are added to the exclude field of subsequent interests, with the same partial names. We evaluate the choice of W in Section 6.5.

Caching: Before we conclude this section, we discuss how caching affects the performance of our system. As the number of consumers increase, it is expected that caches at intermediate routers will return data objects more often, improving latency performance with respect to a scenario wherein caching is disabled. This in turn could reduce the producer diversity opportunities that *NEST* tries to exploit to improve performance; the data is already cached en route. However, as will be shown in Section 6.5, the marginal gain in latency reduction even when caching is enabled is large. In addition, the combined gain of *NEST* and caching can be substantial.

6.4 Implementation

We implement *NEST* on Mini-NDN [23], an NDN network emulator based on the popular Mininet[21] virtual network environment. In Mini-NDN, a network topology is specified in which nodes are connected via links parameterized by link delay, bandwidth as well as loss percentage. Each node in the network is capable of running NDN applications, forwarding NDN packets according to the specified routing policy, as well as caching forwarded content.

Mini-NDN accomplishes these NDN functionalities by running an instance of Named Data Link State Routing Protocol (NLSR) [28] and NDN Forwarding Daemon

(NFD) [27] on each instantiated node in the network. NLSR is a routing protocol responsible for populating NDN's Forwarding Information Base (FIB) while NFD is a network forwarder that is fully capable of forwarding NDN packets according to a diverse set of routing strategies.

Since Mini-NDN emulates the actual operations of NDN networks, one primary advantage is that the applications developed and tested on Mini-NDN can be readily operational on the NDN testbed [24] or other actual NDN networks.

A depiction of *NEST*'s different components is shown in Fig. 6.2. Each producer in the network runs the two functional components of *NEST* (ProdSync and *PDST*) simultaneously while the consumer runs *PDST*. First, producers in *NEST* run the "*NEST* Sync" application, which is responsible for implementing ProdSync and creating "*NEST* tree". Here, we use k-means clustering with $k = 2$. The *NEST* tree is then passed to the "*NEST* Prod" application. In this application, the tree is transformed to an LON which is then used to guide the transport of data items. Finally, the third application is the "*NEST* Consum" application running at each consumer. This component is responsible for sending specially crafted interest messages. These interest messages are responded to by the *NEST* Prod application running at each producer.

We implemented all the applications in Python. In the *NEST* Sync application, clustering information of every tree node is kept in a data structure that holds the state of the computations and data exchange between the coordinator and non-coordinator producers. State and data are encoded into control messages, where an interest control message requests the start of computation or delivers the notification that a computation is completed, while

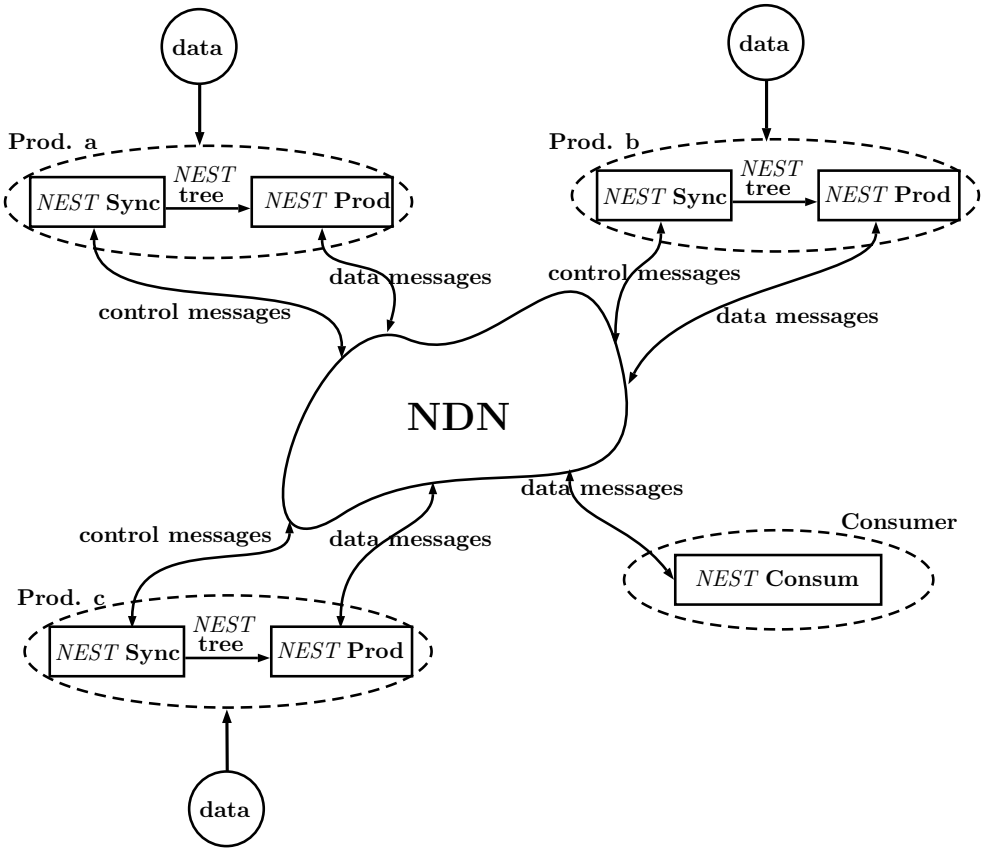


Figure 6.2: A Network with three producers and a consumer running *NEST*. Each box represents an application running on producers or consumers.

the corresponding data control message delivers an acknowledgment or the requested data. On the other hand, in the *NEST* Consum application, the consumer implements a pipelining window of pending interests and a method to transform the LON to interest messages with partial names. *NEST* Prod implements a partial interest name match function to select messages to be sent to the consumers.

6.5 Evaluation Results

Setup: Our evaluations are based on a dataset collected from Twitter. Over a period of time from Dec 2016 to Mar 2017, tweets were collected using Twitter's streaming API and a set of search keywords for trending topics in politics, sports, and entertainment. Overall, we use a dataset of about 80K tweets in our evaluations.

In each experiment, we randomly distribute a sample of the dataset uniformly across the set of producers. We first pre-process the collected tweets to remove stop words, special characters, links and attachments, producing tokens. These tokens are then transformed to a high dimensional vector representation by computing the product of term frequency and inverse document frequency (*tf-idf*) [174], a popular method for text vectorization. We use the `sklearn` library [169] vectorizer to achieve this task.

In Mini-Net, links connecting the producers and consumers are characterized by the link delay, the bandwidth, and the message loss rate. In our experiments, we fix the bandwidth and loss rates, and vary the link delays. We note that in Mini-Net, each host in the network runs the NDN stack and thus can be used as a producer, a consumer, and a forwarding switch, simultaneously. In addition, hosts have content stores and thus can cache data objects. In our experiments, we use a network topology similar to the NDN testbed [24] and we have two consumers and a varying number of producers for different experiments as will be discussed in the following subsections.

In the following, we define terms that we use in our evaluations. Let the *summary block* with size B be the number of data objects the consumer has to fetch in order to have a satisfactory summary. The *block latency* t_B is the delay from sending the interest

Table 6.2: ProdSync convergence time.

N	**3**	**5**	**7**	**9**
Time(s)	34	38	83	122

message for the first data object in the block, until the successful reception of the data message corresponding to the last data object in the summary block. This quantity is directly proportional to the per interest latency defined in Section 6.3.2.

We divide the evaluation results into two parts. In the first, we evaluate the performance of the ProdSync algorithm and quantify performance in terms of the convergence time and the clustering quality. In the second, we focus on the latency performance of *PDST* and compare it to a *baseline* summary transport protocol with no producer diversity, i.e., a system in which the LON is used to retrieve data objects from specific producers, similar to the protocol in [145].

6.5.1 Producer Sync

We consider networks with different numbers of producers and distribute a dataset of $4000N$ tweets uniformly at random over the N producers. We use Euclidean distance to measure similarity between different vectors representing tweets, and use a similarity threshold $\tau = 0.9$ as the stopping criterion for ProdSync. For the coordinator selection, in each iteration, we let the producer with the smallest ID perform the coordination tasks for the corresponding tree node. Producers are connected with link delays of 10milliseconds.

We first evaluate ProdSync's convergence time. For scenarios with $N = 3$, 5, 7 and 9 producers, we repeat the experiment 10 times and report the average convergence time in each case. Table 6.2 outlines the results. It can be seen that ProdSync's convergence time is

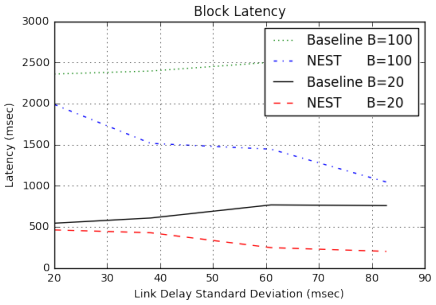


Figure 6.3: Latency performance for different link delay variance.

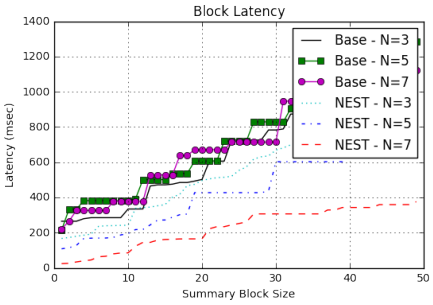


Figure 6.4: Latency performance for different number of producers.

approximately linear in the number of producers and the dataset size. In many applications (such as traffic monitoring, news stories updates), major dataset changes happen on the order of hours. Thus, ProdSync provides a practical means for constructing the global representation of the distributed dataset as it can be run periodically at a rate that is faster than the rate of data evolution. For completeness, we also report the average value for clustering costs for an example scenario with five producers. In this experiment, the average number of tree nodes was 13.3 and the average clustering cost per data point is 0.47.

6.5.2 Latency Performance

In this section, we evaluate the latency performance of *PDST* after LON has been delivered to the consumers. We compare the performance to the baseline protocol.

Link delay variance

First, we vary the *link delay variance* and measure the incurred latency. We fix the maximum pipelining window size to $W = 10$. To change variance, we vary the link delays

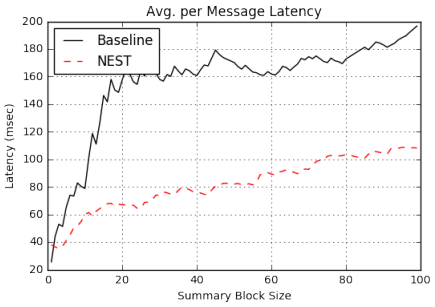


Figure 6.5: Per message latency.

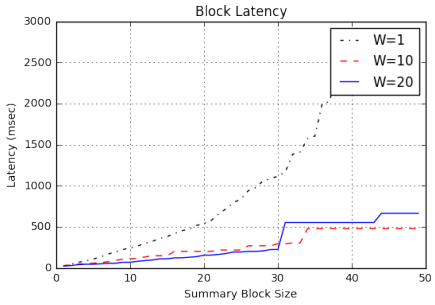


Figure 6.6: Latency performance for different pipelining window sizes.

from the producers to the consumer in the range $5 - 200$ milliseconds, while maintaining a fixed average. Here, we consider a topology with five producers and one consumer, and we consider two different summary block sizes $B = \{20, 100\}$. As shown in Fig. 6.3, the block latency improves as the link delay variance increases. This is because *NEST* effectively checks if similar objects exist at producers with better network conditions and fetches objects from those producers first. Essentially, objects with slow retrieval times are pushed to the end of retrieval order. Compared to the baseline system, *NEST* improves the block latency performance by more than 40% when the link delay standard deviation is 50 milliseconds.

In Fig. 6.4, we plot the block latency t_B for a varying B. We also show the performance for topologies with different number of producers. First, we observe that while the baseline system performance does not change when number of producers change, *NEST* fully utilizes producer diversity. In particular, as the number of producers increases, diversity improves and block latency decreases.

We also study the per message latency when $N = 5$, where W and link delays are chosen as in previous experiments. In Fig. 6.5, we plot the average per message latency

vs. different B. The figure shows that by taking advantage of producer diversity, the latency improvements can be as high as 100%. Note that as the block size increases for a fixed dataset size, this gain is expected to decrease. We study the effect of the ratio $\frac{B}{|\mathcal{P}|}$ in Section 6.5.2.

Pipelining

Next, we study the effect of the maximum pipelining window size W on the block latency. In Fig. 6.6, there are five producers with link delays similar to those in the previous experiments. Note that *PDST*'s adaptive pipelining does not send new interest messages while pending interests with the same partial name exist to avoid duplicate object retrievals. It is seen that pipelining improves block latency compared to a simple stop and wait approach ($W = 1$). In addition, consumers will experience more packet losses as W increases and thus more bandwidth wastage. The figure shows that diminishing gains are attained with increasing W. We find that $W = 10$ allows *NEST* to achieve the best latency performance.

Impact of dataset size

Next, we study the effect of the dataset size on the block latency. It is expected that as the ratio $\frac{B}{|\mathcal{P}|}$ decreases, the latency gain of *NEST* increases. In other words, when the requested summary block size B is comparable to the dataset size, consumers may not avoid fetching objects from producers with unfavorable network conditions. We fix five producers with link delays similar to previous experiments. Fig. 6.7 shows that, for a given

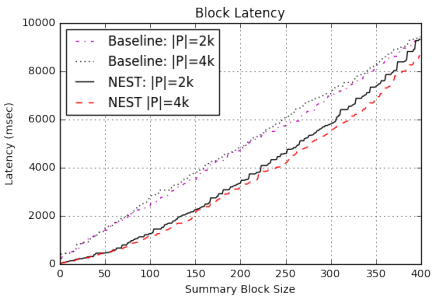


Figure 6.7: Latency performance for different sample size $|\mathcal{P}|$.

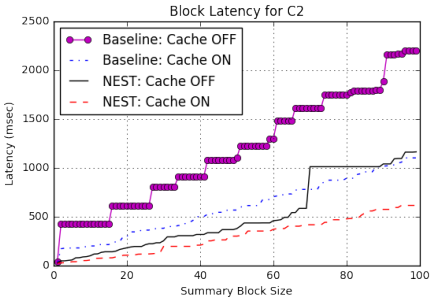


Figure 6.8: Effect of caching on latency performance.

block size B, the latency reduction gain is only slightly reduced when the sample size is halved. When $|\mathcal{P}| = 2000$, *NEST* can achieve a positive gain for a summary block sizes as large as 20% of the dataset, which is reasonable for applications with large datasets in which consumers are interested only in data summaries.

Caching

Finally, we study the performance of *NEST* when caching is enabled in the underlying NDN. In this experiment, the topology has two consumers and five producers with similar link delays as in previous experiments. Both consumers are using the same LON. We introduce a delay between the time each consumer starts fetching items to see the effect of caching. Caching allows intermediate nodes to temporarily store content that was previously forwarded to other hosts in the network. As seen in Fig. 6.8, *NEST* yields latency performance improvement of about 50%. Compared to the baseline performance with caching disabled, the combined latency reduction gain is more than 70%.

6.6 Related Work

There has been prior work on selectively sending a representative subset of data instead of the entire dataset [114, 151]. However, unlike our work, these efforts are application specific. Moreover, these approaches try to optimize for energy efficiency in contrast to our goal of minimizing latency in retrieving the summary. Achieving our goal requires an approach that is much different from those proposed in these efforts.

More recently, optimizing latency in NDN has been considered in [63, 99, 156]. In [63], architectural changes to NDN are proposed to improve the support for low latency applications such video conferencing. In [99] and [156], multi-path routing as well as network coding are employed to improve performance of video streaming. Unlike these approaches, *NEST* does not require changes to the underlying NDN infrastructure and operates as an end-host application. Thus, we argue that it is much more general and easy to deploy.

Distributed dataset synchronization was recently addressed in [216]. The goal is to efficiently synchronize the state of a group of hosts for applications such as group text messaging. This is different than the problem we consider when we address producer synchronization since we do not require all hosts to have the full dataset.

The closest works to ours are [144] and [145]. The former creates a tree representation and object names from a given dataset for creating summaries, while the latter transforms the tree into an ordered list for transport. However, their model considers only a single producer. While we use a similar approach towards summarization, we address a different set of challenges wherein the dataset is distributed across multiple producers.

In particular, transport performance becomes a primary issue which was not addressed in these works.

Chapter 7

Conclusions

In this thesis, we propose a variety of frameworks to tackle security and performance problems in modern networks. We begin in chapter 2 by evaluating the limitations of IDS systems. We demonstrate that by intelligently combining a plurality of low intensity TCP-based DoS attacks, an attacker can evade traditional single scalar threshold based intrusion detection systems. We argue that multiple features need to be considered for efficiently detecting such stealthy attacks. Via extensive experiments, we identify such a set of features, and jointly examine them in a new simple, yet effective detection framework. We demonstrate, via extensive experiments, that our approach can detect stealthy attacks effectively unlike traditional approaches.

In lieu of manually choosing features as done traditionally, In chapter 3, we develop a framework that automates and sequentializes the process of feature selection for highly accurate intrusion detection. In building our framework, we design and implement a comprehensive evidence collection framework, and undertake an in-depth study to gain

an understanding of which search algorithms to use for feature reduction. Our approach is agnostic to the engine that uses these features to perform inference. We demonstrate the efficacy of our framework with DoS and SQL injection attacks. We demonstrate that the features (which are automatically chosen) work very well in terms of providing high detection accuracy with respect to the true states (attack and normal scenarios) the networked system is in

In chapter 4, we tackle the problem of ISP scale intrusion detection. Challenges of scale, flexibility and complexity have hindered the realization of a NIDS that can be deployed on an ISP scale network. We propose a framework *NEST* that addresses this long-standing challenge by exploiting in-network processing to generate fine grained, yet concise packet summaries, which can be analyzed centrally to deliver highly accurate inferences with regards to a wide range of attacks. *NEST* reduces overheads by over 65% compared to sending raw packets (state of the art today) while achieving a detection accuracy of over 98%. This detection accuracy, we believe, is the highest reported for intrusion detection at ISP scale.

In chapter 5 we turn our focus to performance issues. We build an analytical framework to capture the energy consumption due to video downloads over LTE. Our framework takes as input, the type of video (fast vs slow motion and resolution), the link qualities experienced (in terms of PER) and the power consumed in each LTE state. It provides a quick and effective means of determining the power consumption with various types of videos under different network conditions. We validate our framework via extensive simulations and real experiments

Finally in chapter 6, we target the problem of delivering a summary of a large dataset to consumers from a set of producers with low latency. Retrieval of such a summary which is essentially a set of representative samples of the dataset, is becoming popular in many emerging applications. We propose *NEST*, an efficient data transport framework which leverages the NDN architecture towards achieving this goal. Our novel framework opportunistically fetches data from the producers with good network conditions relative to consumers after constructing a global view of the dataset shared between producers and establishing similarity relations between data points. Our experimental results show that large latency reduction gains can be achieved compared to baseline strategies that do not exploit producer diversity. The gains are especially noteworthy (up to 100%) when the number of producers and link delay variations are large.

Bibliography

[1] https://www.arbornetworks.com.

[2] 3GPP LTE. http://www.3gpp.org/LTE.

[3] Adobe: Smartphone Video Streaming Rises 86% in one year. http://www.telecompetitor.com/adobe-smartphone-video-streaming-rises-86-in-one-year.

[4] AForge.NET. http://www.aforgenet.com/framework/features/motion_detection_2.0.html.

[5] Akamai releases q2 2015 state of the internet - security report. http://akamai.me/1qN434s.

[6] Apache, http server project. https://httpd.apache.org/.

[7] The bro network security monitor. https://www.bro.org/.

[8] Centos. https://www.centos.org.

[9] Cisco Visual Networking Index: Forecast and Methodology. http://bit.ly/LVhmuL.

[10] Cyber attacks likely to increase. http://pewrsr.ch/1qN4agg.

[11] D-WARD: DDoS network attack recognition and defense. http://www.lasr.cs.ucla.edu/ddos/.

[12] DARPA intrusion detection evaluation. http://bit.ly/1NtBr50.

[13] The DETER project. http://deter-project.org.

[14] DoS attacks get more complexâĂŞare networks prepared? http://defensesystems.com/articles/2013/12/19/dosattacks-complexity.aspx?admgarea=DS.

[15] emulab. http://www.emulab.net.

[16] EvalVid. http://bit.ly/UOPQ6v.

[17] First annual msu mpeg-4 avc/h.264 video codec comparison. http://compression.ru/video/codec_comparison/mpeg4_avc_h264_2004/mpeg4_avc_h264_2004_part2.h

[18] Intrusion detection FAQ: Statistical based approach to intrusion detection. https://www.sans.org/securityresources/idfaq/statistic_ids.php.

[19] Kali linux. https://www.kali.org/.

[20] LENA. http://bit.ly/UEleow.

[21] Mininet. http://mininet.org/.

[22] Monsoon Power Monitor. http://bit.ly/1lh8JX1.

[23] named-data/mini-ndn. https://github.com/named-data/mini-ndn.

[24] NDN testbed. https://named-data.net/ndn-testbed/.

[25] Netflix. http://www.techtimes.com/articles/7009/20140515/netflix-accounts-for-34-21-percent-of-downstream-traffic-in-u-s-report.htm.

[26] Netflix Now The Largest Single Source of Internet Traffic In North America. http://bit.ly/1xHkxHA.

[27] NFD - named data networking forwarding daemon. https://named-data.net/doc/NFD/current.

[28] NLSR - named data link state routing protocol. http://named-data.net/doc/NLSR/current.

[29] NS3. http://www.nsnam.org/.

[30] Slowloris HTTP DoS. http://ha.ckers.org/slowloris/.

[31] Snort. https://www.snort.org.

[32] Sockstress tools & source code. http://bit.ly/1SgI9Qd.

[33] What we learned from anonymous: DDoS is now 3DoS. https://devcentral.f5.com/articles/whatwelearnedfromanonymousddosisnow3dos.

[34] Strategies to reduce false positives and false negatives in nids. https://www.symantec.com/connect/articles/strategies-reduce-false-positives-and-false-negatives-nids, 2001.

[35] Snort faq. https://www.snort.org/faq/readme-sfportscan, 2004.

[36] The ddos that knocked spamhaus offline. http://blog.cloudflare.com/the-ddos-that-knocked-spamhaus-offline-and-ho, 2013.

[37] The expanding role of service providers in DDoS mitigation. https://resources.arbornetworks.com/i/481939-the-expanding-role-of-service-providers-in-ddos-mitigation?hubItemID=55526068, 2015. [Online; accessed 23-Jan-2017].

[38] IBM reat Intelligence QuX-Force Tharterly, 1Q 2015, 2015.

[39] The biggest data breaches in 2016, so far. https://www.identityforce.com/blog/2016-data-breaches, 2016. [Online; accessed 10-Jan-2017].

[40] How the dyn ddos attack unfolded. http://www.networkworld.com/article/3134057/security/how-the-dyn-ddos-attack-unfolded.html, 2016.

[41] jgamblin/mirai-source-code. https://github.com/jgamblin/Mirai-Source-Code, 2016.

[42] Krebsonsecurity hit with record ddos. https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/, 2016.

[43] Large DDoS attacks cause outages at twitter, spotify, and other sites. https://techcrunch.com/2016/10/21/many-sites-including-twitter-and-spotify-suffering-outage/, 2016.

[44] Lessons from the dyn ddos attack. https://www.schneier.com/blog/archives/2016/11/lessons_from_th_5.html, 2016.

[45] Mirai iot botnet description and ddos attack mitigation. https://www.arbornetworks.com/blog/asert/mirai-iot-botnet-description-ddos-attack-mitigation/, 2016.

[46] Mirai: what you need to know about the botnet behind recent major ddos attacks. https://www.symantec.com/connect/blogs/mirai-what-you-need-know-about-botnet-behind-recent-major-ddos-attacks, 2016.

[47] Someone is learning how to take down the internet. https://www.schneier.com/blog/archives/2016/09/someone_is_lear.html, 2016.

[48] Top 7 types of network attacks. http://www.calyptix.com/top-threats/top-7-network-attack-types-2016/, 2016.

[49] White paper: Cisco VNI forecast and methodology, 2015-2020. White paper, Cisco, July 2016.

[50] Cyber-hunting at scale (chase). https://www.fbo.gov/index?s=opportunity&mode=form&id=a6b09e0661902c71a9c3205db0fff55d&tab=core&_cview=1, 2017.

[51] Ddos attack types: Glossary of terms. https://www.corero.com/resources/glossary.html, 2017.

[52] MAWI working group traffic archive. http://mawi.wide.ad.jp/mawi/, 2017.

[53] Nmap: the network mapper. https://nmap.org, 2017.

[54] Rule doc search. https://snort.org/rule-docs, 2017.

[55] Ryu SDN framework. https://osrg.github.io/ryu/, 2017.

[56] Sid 1-19559. https://www.snort.org/rule_docs/1-19559, 2017.

[57] Sid 3-16294. https://www.snort.org/rule_docs/3-16294, 2017.

[58] Snort users manual. https://www.snort.org/documents/snort-users-manual, 2017.

[59] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. Network flows: theory, algorithms, and applications. 1993.

[60] Monis Akhlaq, Faeiz Alserhani, Ahsan Subhan, Irfan Ullah Awan, John Mellor, and Pravin Mirchandani. High speed NIDS using dynamic cluster and comparator logic. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 575–581. IEEE, 2010.

[61] Saamer Akhshabi, Ali C. Begen, and Constantine Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, MMSys '11. ACM, 2011.

[62] Susanne Albers. Online algorithms: a survey. *Mathematical Programming*, 97(1-2):3–26, 2003.

[63] Mishari Almishari, Paolo Gasti, Naveen Nathan, and Gene Tsudik. Optimizing bi-directional low-latency communication in Named Data Networking. *SIGCOMM Comput. Commun. Rev.*, 44(1):13–19, December 2013.

[64] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. NP-hardness of euclidean sum-of-squares clustering. *Machine Learning*, 75(2):245–248, 2009.

[65] Faeiz Alserhani, Monis Akhlaq, Irfan U Awan, John Mellor, Andrea J Cullen, and Pravin Mirchandani. Evaluating intrusion detection systems in high speed networks. In *Information Assurance and Security, 2009. IAS'09. Fifth International Conference on*, volume 2, pages 454–459. IEEE, 2009.

[66] Ashok Anand, Athula Balachandran, Aditya Akella, Vyas Sekar, and Srinivasan Seshan. Enhancing video accessibility and availability using information-bound references. In *ACM CoNEXT*, 2013.

[67] Ghassane Aniba and Sonia Aissa. Packet delay modeling of truncated multi-process ARQ protocols for parallel communications. In *IEEE ICT*, 2010.

[68] Azeem Aqil, Ahmed OF Atya, Trent Jaeger, Srikanth V Krishnamurthy, Karl Levitt, Patrick D McDaniel, Jeff Rowe, and Ananthram Swami. Detection of stealthy tcp-based dos attacks. In *MILCOM*. IEEE, 2015.

[69] David Arthur, Bodo Manthey, and Heiko Röglin. Smoothed analysis of the k-means method. *J. ACM*, 58(5):19:1–19:31, October 2011.

[70] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[71] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technical report, Technical report Chalmers University of Technology, Goteborg, Sweden, 2000.

[72] Yossi Azar. On-line load balancing. In *Online Algorithms*, pages 178–195. Springer, 1998.

[73] Yossi Azar, Andrei Z Broder, and Anna R Karlin. On-line load balancing. *Theoretical Computer Science*, 130(1):73–84, 1994.

[74] Yossi Azar, Bala Kalyanasundaram, Serge Plotkin, Kirk R Pruhs, and Orli Waarts. On-line load balancing of temporary tasks. *Journal of Algorithms*, 22(1):93–110, 1997.

[75] Thomas Bäck. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. OUP, 1996.

[76] L. Badia and A.V. Guglielmi. A markov analysis of automatic repeat request for video traffic transmission. In *A World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on*, 2014.

[77] Maria-Florina F Balcan, Steven Ehrlich, and Yingyu Liang. Distributed k-means and k-median clustering on general topologies. In *Advances in Neural Information Processing Systems*, pages 1995–2003, 2013.

[78] Monowar H Bhuyan, Dhruba Kumar Bhattacharyya, and Jugal K Kalita. Network anomaly detection: methods, systems and tools. *IEEE Communications Surveys & Tutorials*, 16(1):303–336, 2014.

[79] C.S. Bontu and E. Illidge. DRX mechanism for power saving in LTE. *IEEE Comm. Magazine*, 2009.

[80] Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, October 1992.

[81] Safdar Hussain Bouk, Syed Hassan Ahmed, Dongkyun Kim, and Houbing Song. Named-data-networking-based its for smart cities. *IEEE Communications Magazine*, 55(1):105–111, 2017.

[82] A. C. Bovik. *The Essential Guide to Video Processing*. Academic Press, 2009.

[83] Lothar Braun, Cornelius Diekmann, Nils Kammenhuber, and Georg Carle. Adaptive load-aware sampling for network monitoring on multicore commodity hardware. In *IFIP Networking Conference, 2013*, pages 1–9. IEEE, 2013.

[84] S Terry Brugger and Jedidiah Chow. An assessment of the darpa ids evaluation dataset using snort. *UCDAVIS department of Computer Science*, 1(2007):22, 2007.

[85] Waleed Bulajoul, Anne James, and Mandeep Pannu. Network intrusion detection systems in high-speed traffic in computer networks. In *e-Business Engineering (ICEBE), 2013 IEEE 10th International Conference on*, pages 168–175. IEEE, 2013.

[86] Michael Buratowski. The DNC server breach: who did it and what does it mean? *Network Security*, 2016(10):5–7, 2016.

[87] G. Cermak, M. Pinson, and S. Wolf. The relationship among video quality, screen resolution, and bit rate. *Broadcasting, IEEE Transactions on*, 2011.

[88] Abhijnan Chakraborty, Vishnu Navda, Venkata N. Padmanabhan, and Ramachandran Ramjee. Coordinating cellular background transfers using loadsense. In *Proceedings of the 19th Annual International Conference on Mobile Computing & Networking*, MobiCom '13. ACM, 2013.

[89] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[90] Chia-Wei Chang, Guanyao Huang, Bill Lin, and Chen-Nee Chuah. LEISURE: Load-balanced network-wide traffic measurement and monitor placement. *IEEE Transactions on Parallel and Distributed Systems*, 26(4):1059–1070, 2015.

[91] Srilatha Chebrolu, Ajith Abraham, and Johnson P Thomas. Feature deduction and ensemble design of intrusion detection systems. *Computers & Security*, 24(4):295–307, 2005.

[92] Jiecao Chen, He Sun, David Woodruff, and Qin Zhang. Communication-optimal distributed clustering. In *Advances in Neural Information Processing Systems*, pages 3727–3735, 2016.

[93] Min Chen, Shiwen Mao, and Yunhao Liu. Big data: A survey. *Mobile Networks and Applications*, 19(2):171–209, 2014.

[94] Shihabur Rahman Chowdhury, Md Faizul Bari, Reaz Ahmed, and Raouf Boutaba. PayLess: A low cost network monitoring framework for software defined networks. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–9. IEEE, 2014.

[95] Richard G Clegg, Stuart Clayman, George Pavlou, Lefteris Mamatas, and Alex Galis. On the selection of management/monitoring nodes in highly dynamic networks. *IEEE Transactions on Computers*, 62(6):1207–1220, 2013.

[96] Louis Columbus. Roundup of internet of things forecasts and market estimates, 2016. `https://www.forbes.com/sites/louiscolumbus/2016/11/27/roundup-of-internet-of-things-forecasts-and-market-estimates-2016/#1b7ea093292d`, 2016.

[97] Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.

[98] Pilu Crescenzi, Giorgio Gambosi, Gaia Nicosia, Paolo Penna, and Walter Unger. Online load balancing made simple: Greedy strikes back. *Journal of Discrete Algorithms*, 5(1):162–175, 2007.

[99] Saltarin de Arco, Jonnahtan Eduardo, Eirina Bourtsoulatze, Nikolaos Thomos, and Torsten Braun. Adaptive video streaming with network coding enabled named data networking. *IEEE Trans. on Multimedia*, 2017.

[100] Reinhard Diestel, Daniel Král, and Paul Seymour. Graph theory. *Oberwolfach Reports*, 13(1):51–86, 2016.

[101] Ning Ding, Daniel Wagner, Xiaomeng Chen, Abhinav Pathak, Y. Charlie Hu, and Andrew Rice. Characterizing and modeling the impact of wireless signal strength on smartphone battery drain. In *ACM SIGMETRICS*, 2013.

[102] J. Dohl and G. Fettweis. Energy aware evaluation of lte hybrid-arq and modulation/coding schemes. In *Communications (ICC), 2011 IEEE International Conference on*, pages 1–5, 2011.

[103] Nick Duffield, Carsten Lund, and Mikkel Thorup. Priority sampling for estimation of arbitrary subset sums. *Journal of the ACM (JACM)*, 54(6):32, 2007.

[104] William Dumouchel and Matthias Schonlau. A comparison of test statistics for computer intrusion detection based on principal components regression of transition probabilities. In *30th Symposium on the Interface: Computing Science and Statistics*, 1998.

[105] V. Durcekova, L. Schwartz, and N. Shahmehri. Sophisticated denial of service attacks aimed at application layer. In *Proc. ELEKTRO, 2012*, 2012.

[106] Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[107] W. Eddy. TCP SYN flooding attacks and common mitigations. In *RFC 4987*, Aug 2007.

[108] Mohammadreza Ektefa, Sara Memar, Fatimah Sidi, and Lilly Suriani Affendey. Intrusion detection using data mining techniques. In *Information Retrieval & Knowledge Management,(CAMP), 2010*. IEEE.

[109] Cristian Estan, Ken Keys, David Moore, and George Varghese. Building a better netflow. In *ACM SIGCOMM Computer Communication Review*, volume 34, pages 245–256. ACM, 2004.

[110] W. Fischer and K. Meier-Hellstern. The Markov-modulated Poisson process (MMPP) cookbook. Performance Evaluation. 18 (1992) 149âĂŞ171.

[111] Thomer M. Gil and Massimiliano Poletto. Multops: A data-structure for bandwidth attack detection. In *Proceedings of the 10th Conference on USENIX Security Symposium - Volume 10*, SSYM'01, 2001.

[112] Luis Campo Giralte, Cristina Conde, Isaac Martin de Diego, and Enrique Cabello. Detecting denial of service by modelling web-server behaviour. *Computers & Electrical Engineering*, 39(7):2252 – 2262, 2013.

[113] DavidE. Goldberg and JohnH. Holland. Genetic algorithms and machine learning. *Machine Learning*, 3:95–99, 1988.

[114] Himanshu Gupta, Vishnu Navda, Samir Das, and Vishal Chowdhary. Efficient gathering of correlated data in sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 4(1):4, 2008.

[115] Martin Gütlein, Eibe Frank, Mark Hall, and Andreas Karwath. Large-scale attribute selection using wrappers. In *IEEE CIDM*, 2009.

[116] Mark A Hall. *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato, 1999.

[117] Bo Han, Vijay Gopalakrishnan, Lusheng Ji, and Seungjoon Lee. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97, 2015.

[118] Jiawei Han, Micheline Kamber, and Jian Pei. *Data mining: concepts and techniques: concepts and techniques*. Elsevier, 2011.

[119] Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of Computing*, pages 291–300. ACM, 2004.

[120] Zhihai He, Yongfang Liang, Lulin Chen, I Ahmad, and Dapeng Wu. Power-rate-distortion analysis for wireless video communication under energy constraints. *IEEE Transactions on Circuits and Systems for Video Technology*, 2005.

[121] H. Heffes and D. Lucantoni. A markov modulated characterization of packetized voice and data traffic and related statistical multiplexer performance. *IEEE J.Sel. A. Commun.*, 4(6):856–868, September 2006.

[122] Victor Heorhiadi, Michael K Reiter, and Vyas Sekar. New opportunities for load balancing in network-wide intrusion detection systems. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 361–372. ACM, 2012.

[123] Victor Heorhiadi, Michael K Reiter, and Vyas Sekar. Simplifying software-defined network optimization using sol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 223–237. USENIX Association, 2016.

[124] M.A Hoque, M. Siekkinen, J.K. Nurminen, and M. Aalto. Dissecting mobile video services: An energy consumption perspective. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2013.

[125] Chengchen Hu, Sheng Wang, Jia Tian, Bin Liu, Yu Cheng, and Yan Chen. Accurate and efficient traffic monitoring using adaptive non-linear sampling method. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 26–30. IEEE, 2008.

[126] J. Hwang, K. K. Ramakrishnan, and T. Wood. NetVM: High performance and flexible networking using virtualization on commodity platforms. *IEEE Transactions on Network and Service Management*, March 2015.

[127] O.C. Ibe. Markov Process for Stochastic Modelling. Academic Press, 2008.

[128] Félix Iglesias and Tanja Zseby. Analysis of network traffic features for anomaly detection. *Machine Learning*, 2015.

[129] F. Qian J. Huang, A. Gerber, S. Sen Z. M. Mao, and O. Spatscheck. A close examination of performance and power characteristics of 4G LTE networks. ACM MobiSys 2012.

[130] Ramaprabhu Janakiraman, Marcel Waldvogel, and Qi Zhang. Indra: A peer-to-peer approach to network intrusion detection and prevention. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on*, pages 226–231. IEEE, 2003.

[131] AR. Jensen, M. Lauridsen, P. Mogensen, T.B. SÃÿrensen, and P. Jensen. LTE UE power consumption model: For system level energy and performance optimization. In *IEEE VTC*, 2012.

[132] Junchen Jiang, Vyas Sekar, and Hui Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12. ACM, 2012.

[133] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[134] Audun Josang, Javier Diaz, and Maria Rifqi. Cumulative and averaging fusion of beliefs. *Inf. Fusion*, 11(2):192–200, 2010.

[135] N Kamiyama and T Mori. Simple and accurate identification of high-rate flows by packet sampling. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*.

[136] Junaid Khalid, Aaron Gember-Jacobson, Roney Michael, Anubhavnidhi Abhashkumar, and Aditya Akella. Paving the way for NFV: Simplifying middlebox modifications using StateAlyzr. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 239–253. USENIX Association, 2016.

[137] George Khalil. Open source ids high performance shootout. White paper, SANS Institute, February 2015.

[138] Marius Kloft, Ulf Brefeld, Patrick Düessel, Christian Gehl, and Pavel Laskov. Automatic feature selection for anomaly detection. In *Proceedings of the 1st ACM Workshop on AISec*. ACM, 2008.

[139] T. Kolding, J. Wigard, and Lars Dalsgaard. Balancing power saving and single user experience with discontinuous reception in LTE. In *IEEE ISWCS*, 2008.

[140] Alexander Kott, Ananthram Swami, and Bruce J West. The internet of battle things. *Computer*, 49(12):70–75, 2016.

[141] Balachander Krishnamurthy, Subhabrata Sen, Yin Zhang, and Yan Chen. Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 234–247. ACM, 2003.

[142] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. In *International Conference on Machine Learning*, pages 957–966, 2015.

[143] Anukool Lakhina, Mark Crovella, and Christophe Diot. Diagnosing network-wide traffic anomalies. In *ACM SIGCOMM Computer Communication Review*, volume 34, pages 219–230. ACM, 2004.

[144] Jongdeog Lee, Md Tanvir Al Amin, and Tarek Abdelzaher. Espresso: A data naming service for self-summarizing transport. In *2017 14th Annual IEEE Int. Conf. on Sensing, Commun., and Networking (SECON)*, pages 1–9, 2017.

[145] Jongdeog Lee, Akash Kapoor, Md Tanvir Al Amin, Zhehao Wang, Zeyuan Zhang, Radhika Goyal, and Tarek Abdelzaher. InfoMax: An information maximizing transport layer protocol for named data networks. In *IEEE 2015 24th Int. Conf. on Computer Commun. and Networks (ICCCN)*, pages 1–10, 2015.

[146] Ye Li, Martin Reisslein, and Chaitali Chakrabarti. Energy-efficient video transmission over a wireless link. *Vehicular Technology, IEEE Transactions on*, 58(3):1229–1244, 2009.

[147] Chong Liu, Kui Wu, and Jian Pei. An energy-efficient data collection framework for wireless sensor networks by exploiting spatiotemporal correlation. *IEEE transactions on parallel and distributed systems*, 18(7), 2007.

[148] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. One sketch to rule them all: Rethinking network flow monitoring with UnivMon. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 101–114. ACM, 2016.

[149] Stuart Lloyd. Least squares quantization in PCM. *IEEE Trans. on Inf. Theory*, 28(2):129–137, 1982.

[150] Xiaoan Lu, Yao Wang, and E. Erkip. Power efficient h.263 video transmission over wireless channels. In *International Conference on Image Processing*, 2002.

[151] Yajie Ma, Yike Guo, Xiangchuan Tian, and Moustafa Ghanem. Distributed clustering-based aggregation algorithm for spatial correlated sensor networks. *IEEE Sensors Journal*, 11(3):641–648, 2011.

[152] Jianning Mai, Chen-Nee Chuah, Ashwin Sridharan, Tao Ye, and Hui Zang. Is sampled data sufficient for anomaly detection? In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 165–176. ACM, 2006.

[153] B. Makki, T. Svensson, and M. Zorzi. Finite block-length analysis of the incremental redundancy harq. *Wireless Communications Letters, IEEE*, 3(5):529–532, 2014.

[154] Bill Marczak, Nicholas Weaver, Jakub Dalek, Roya Ensafi, David Fifield, Sarah McKune, Arn Rey, John Scott-Railton, Ron Deibert, and Vern Paxson. An analysis of chinaâĂŹs âĂIJgreat cannonâĂİ. In *5th USENIX Workshop on Free and Open Communications on the Internet (FOCI 15)*, 2015.

[155] Bernard Marr. Big data overload: Why most companies can't deal with the data explosion. `https://www.forbes.com/sites/bernardmarr/2016/04/28/big-data-overload-most-companies-cant-deal-with-the-data-explosion/#33cde7b06b0d`, 2016.

[156] Kazuhisa Matsuzono, Hitoshi Asaeda, and Thierry Turletti. Low latency low loss streaming using in-network coding and caching. In *IEEE INFOCOM*, 2017.

[157] Nimrod Megiddo. Optimal flows in networks with multiple sources and sinks. *Mathematical Programming*, 7(1):97–107, 1974.

[158] J. Mirkovic, G. Prier, and P. Reiher. Attacking DDoS at the source. In *In Proc. 10th IEEE International Conference on Network Protocols*, 2002.

[159] Shivajit Mohapatra, Radu Cornea, Nikil Dutt, Alex Nicolau, and Nalini Venkatasubramanian. Integrated power management for video streaming to mobile handheld devices. In *ACM Multimedia*, 2003.

[160] Srinivas Mukkamala and Andrew Sung. Feature selection for intrusion detection with neural networks and support vector machines. *Transportation Research Record: Journal of the Transportation Research Board*, (1822):33–39, 2003.

[161] Christopher Müller, Stefan Lederer, and Christian Timmerer. An evaluation of dynamic adaptive streaming over http in vehicular environments. In *Proceedings of the 4th Workshop on Mobile Video*, MoVid '12. ACM, 2012.

[162] Gerhard Münz, Sa Li, and Georg Carle. Traffic anomaly detection using k-means clustering. In *GI/ITG Workshop MMBnet*, 2007.

[163] Gerhard MÃijnz, Nico Weber, and Georg Carle. Signature detection in sampled packets. In *in Proc. of IEEE Workshop on Monitoring, Attack Detection and Mitigation (MonAM 2007)*, 2007.

[164] W.W.Y. Ng, R.K.C. Chang, and D.S. Yeung. Dimensionality reduction for denial of service detection problems using rbfnn output sensitivity. In *Int'l Conf on Machine Learning and Cybernetics,*, 2003.

[165] A Pande, V. Ramamurthi, and P. Mohapatra. Quality-oriented video delivery over lte using adaptive modulation and coding. In *IEEE GLOBECOM*, 2011.

[166] Nicholas Pappas. Network IDS & IPS deployment strategies. White paper, SANS Institute, April 2008.

[167] Samuel Patton, William Yurcik, and David Doss. An achillesâĂŹ heel in signature-based ids: Squealing false positives in snort. In *Proceedings of RAID*, volume 2001, 2001.

[168] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23):2435–2463, 1999.

[169] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[170] John Platt et al. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998.

[171] Alex Poylisher, Yitzchak M Gottlieb, Constantin Serban, Jeyull Lee, Farooq Sultan, Ritu Chadha, C Jason Chiang, Keith Whittaker, John Nguyen, and Chris Scilla. Building an operation support system for a fast reconfigurable network experimentation testbed. In *MILCOM*. IEEE, 2012.

[172] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. Simple-fying middlebox policy enforcement using sdn. *ACM SIGCOMM computer communication review*, 43(4):27–38, 2013.

[173] Anirudh Ramachandran, Srinivasan Seetharaman, Nick Feamster, and Vijay Vazirani. Fast monitoring of traffic subpopulations. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, pages 257–270. ACM, 2008.

[174] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first Instructional Conf. on Machine Learning*, volume 242, pages 133–142, 2003.

[175] Martin Roesch. Snort-lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX conference on System administration*, pages 229–238. USENIX Association, 1999.

[176] Lior Rokach and Oded Maimon. Clustering methods. In *Data mining and knowledge discovery handbook*, pages 321–352. Springer, 2005.

[177] Lior Rokach and Oded Maimon. Web mining. In *Data mining and knowledge discovery handbook*, pages 321–352. Springer, 2005.

[178] K Salah and A Kahtani. Performance evaluation comparison of snort NIDS under linux and windows server. *Journal of Network and Computer Applications*, 33(1):6–15, 2010.

[179] T. Schierl, T. Stockhammer, and T. Wiegand. Mobile video transmission using scalable video coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 2007.

[180] Tobias Schnabel, Igor Labutov, David M Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. In *EMNLP*, pages 298–307, 2015.

[181] Aaron Schulman, Vishnu Navda, Ramachandran Ramjee, Neil Spring, Pralhad Deshpande, Calvin Grunewald, Venkata N. Padmanabhan, and Kamal Jain. Bartendr: A practical approach to energy-aware cellular data scheduling. In *ACM Mobicom*, 2010.

[182] Vyas Sekar, Michael K Reiter, Walter Willinger, Hui Zhang, Ramana Rao Kompella, and David G Andersen. CSAMP: A system for network-wide flow monitoring. In *NSDI*, volume 8, pages 233–246, 2008.

[183] Vyas Sekar, Michael K Reiter, and Hui Zhang. Revisiting the case for a minimalist approach for network flow monitoring. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 328–341. ACM, 2010.

[184] Vyas Sekar, Michael K. Reiter, and Hui Zhang. Revisiting the case for a minimalist approach for network flow monitoring. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 328–341, New York, NY, USA, 2010. ACM.

[185] Kari Sentz and Scott Ferson. *Combination of evidence in Dempster-Shafer theory*. Sandia National Laboratories, 2002.

[186] Stefania Sesia, Issam Toufik, and Matthew Baker. *LTE - The UMTS Long Term Evolution: From Theory to Practice*. Wiley, 2 edition, September 2011.

[187] Glenn Shafer et al. *A mathematical theory of evidence*. Princeton University Press, 1976.

[188] Praveen Kumar Shanmugam, Naveen Dasa Subramanyam, Joe Breen, Corey Roach, and Jacobus Van der Merwe. DEIDtect: towards distributed elastic intrusion detection. In *Proceedings of the 2014 ACM SIGCOMM workshop on Distributed cloud computing*, pages 17–24. ACM, 2014.

[189] Matti Siekkinen, Mohammad Ashraful Hoque, Jukka K. Nurminen, and Mika Aalto. Streaming over 3G and LTE: How to save smartphone energy in radio access network-friendly way. In *Proceedings of the 5th Workshop on Mobile Video*, 2013.

[190] C. Singhal, S. De, R. Trestian, and G.-M. Muntean. Joint optimization of user-experience and energy-efficiency in wireless multimedia broadcast. *Mobile Computing, IEEE Transactions on*, 2014.

[191] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring ISP topologies with rocketfuel. *ACM SIGCOMM Computer Communication Review*, 32(4):133–145, 2002.

[192] Gary Stein, Bing Chen, Annie S Wu, and Kien A Hua. Decision tree classifier for network intrusion detection with ga-based feature selection. In *Proceedings of the 43rd annual Southeast regional conference-Volume 2*, pages 136–141. ACM, 2005.

[193] Thomas Stockhammer. Dynamic adaptive streaming over http –: Standards and design principles. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, MMSys '11. ACM, 2011.

[194] Kyoungwon Suh, Yang Guo, Jim Kurose, and Don Towsley. Locating network monitors: complexity, heuristics, and coverage. *Computer Communications*, 29(10):1564–1577, 2006.

[195] Andrew H Sung and Srinivas Mukkamala. The feature selection and intrusion detection problems. In *Advances in Computer Science-ASIAN 2004. Higher-Level Decision Making*, pages 468–482. Springer, 2005.

[196] Nirupama Talele, Jason Teutsch, Robert Erbacher, and Trent Jaeger. Monitor placement for large-scale systems. In *Proceedings of the 19th ACM symposium on Access control models and technologies*, pages 29–40. ACM, 2014.

[197] Chi-Ho Tsang, Sam Kwong, and Hanli Wang. Genetic-fuzzy rule mining approach and evaluation of feature selection techniques for anomaly intrusion detection. *Pattern Recognition*, 40(9):2373–2391, 2007.

[198] Matthias Vallentin, Robin Sommer, Jason Lee, Craig Leres, Vern Paxson, and Brian Tierney. The NIDS cluster: Scalable, stateful network intrusion detection on commodity hardware. In *International Workshop on Recent Advances in Intrusion Detection*, pages 107–126. Springer, 2007.

[199] Bobby Vandalore, Wu chi Feng, Raj Jain, and Sonia Fahmy. A survey of application layer techniques for adaptive streaming of multimedia. *Real-Time Imaging*, 7(3):221 – 235, 2001.

[200] Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[201] Haining Wang, Danlu Zhang, and K.G. Shin. Detecting SYN flooding attacks. In *In Proc. IEEE INFOCOM 2002.*, 2002.

[202] Haining Wang, Danlu Zhang, and K.G. Shin. Change-point monitoring for the detection of DoS attacks. *IEEE Transactions on Dependable and Secure Computing*, 1(4):193–208, Oct 2004.

[203] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

[204] Ian H Witten and Eibe Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005.

[205] Yu Xiao, R.S. Kalyanaraman, and A Yla-Jaaski. Energy consumption of mobile YouTube: Quantitative measurement and analysis. In *NGMAST*, 2008.

[206] Georgios Xilouris, E Trouva, F Lobillo, JM Soares, J Carapinha, Michael J McGrath, George Gardikis, P Paglierani, Evangelos Pallis, L Zuccaro, et al. T-nova: A marketplace for virtualized network functions. In *Networks and Communications (EuCNC), 2014 European Conference on*, pages 1–5. IEEE, 2014.

[207] Jingteng Xue and Chang Wen Chen. Mobile video perception: New insights and adaptation strategies. *Selected Topics in Signal Processing, IEEE Journal of*, 8(3):390–401, 2014.

[208] Lili Yang and George Michailidis. Sampled based estimation of network traffic flow characteristics. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1775–1783. IEEE, 2007.

[209] T. Yatagai, T. Isohara, and Iwao Sasase. Detection of http-get flood attack based on analysis of page access behavior. In *Proc. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing. PacRim 2007*, 2007.

[210] Sun Yi, Seyed Kaveh Fayazbakhsh, Yang Guo, Vyas Sekar, Yun Jin, Dali Kaafar, and Steve Uhlig. Trace-Driven Analysis of ICN Caching Algorithms on Video-on-Demand Workloads. In *ACM CoNEXT*, December 2014.

[211] Ya-Ju Yu, Pi-Cheng Hsiu, and Ai-Chun Pang. Energy-efficient video multicast in 4g wireless systems. *Mobile Computing, IEEE Transactions on*, Oct 2012.

[212] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P. Dick, Zhuoqing Morley Mao, and Lei Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *CODES/ISSS*, 2010.

[213] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Patrick Crowley, Christos Papadopoulos, Lan Wang, Beichuan Zhang, et al. Named data networking. *ACM SIGCOMM Computer Communication Review*, 44(3):66–73, 2014.

[214] Wei Zhang, Guyue Liu, Wenhui Zhang, Neel Shah, Phillip Lopreiato, Gregoire Todeschi, K.K. Ramakrishnan, and Timothy Wood. OpenNetVM: A platform for high performance network service chains. In *Proceedings of the 2016 Workshop on Hot*

Topics in Middleboxes and Network Function Virtualization, HotMIddlebox '16, pages 26–31, New York, NY, USA, 2016. ACM.

[215] Lei Zhou, Haibo Xu, Hui Tian, Youjun Gao, Lei Du, and Lan Chen. Performance analysis of power saving mechanism with adjustable DRX cycles in 3GPP LTE. In *IEEE VTC 2008 (Fall)*, 2008.

[216] Zhenkai Zhu and Alexander Afanasyev. Let's Chronosync: Decentralized dataset state synchronization in named data networking. In *2013 21st IEEE Int. Conf. on Network Protocols (ICNP)*, pages 1–10, 2013.