University of California

Los Angeles

End-to-End Detection of Third-Party Middlebox Interference

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Vahab Pournaghshband

2014

Abstract of the Dissertation

End-to-End Detection of Third-Party Middlebox Interference

by

Vahab Pournaghshband

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2014

Professor Peter Reiher, Co-chair

Professor Leonard Kleinrock, Co-chair

Currently, every packet sent on the Internet goes through numerous routers and intermediaries until it gets to the intended receiver. While routing the traffic, these intermediaries (referred to as middleboxes) are potentially capable of making significant changes to what happens to a traffic stream on the network. During the past decade, a wide variety of middleboxes have been proposed, implemented and deployed. Examples include traffic shapers, proxies, firewalls, and WAN optimizers. These middleboxes are becoming a common element of various types of networks, making their detection by end-hosts beneficial and in some cases crucial.

One class of intermediaries, defined as *payload-preserving middleboxes*, makes no changes to the content of the traffic, giving the appearance that nothing has been done to the stream other than routing it to the destination. This transparency property can make end-to-end detection of such intermediaries very challenging in most cases. The main contribution of this dissertation is to investigate the detectability of such middleboxes by seeking answers to this question: "If they pay attention, can the sender or receiver (or both if they cooperate) determine that something of this kind has been done?"

Another contribution of this dissertation is to view and analyze the universal set of all middlebox interferences on network traffic as a whole. We partition this set of middle-

box interferences into detectable and undetectable middleboxes. Within the detectable partition, we introduce the notion of normally detectable middleboxes, denoting that a certain type of middlebox is detectable under normal Internet conditions, with a specific degree of certainty. In this research we developed a general framework to detect network discriminators, which we have defined as middleboxes that delay and/or drop packets in a discriminatory fashion. We achieved this by modeling their interference on the network and proposing a unified solution to the detectability problem, rather than ad hoc approaches that are only applicable to a specific type of middlebox.

To illustrate the implementation feasibility of our generalization idea, we then used our results to detect a number of prevalent and important middleboxes: network compression, traffic prioritization, and traffic shaping/policing. In addition to the analytical approach that we took to solve our detection problem, our results are also supported by extensive network simulations and live Internet experiments using a real middlebox.

Sheila Greibach

Ying Nian Wu

Douglas Stott Parker

Leonard Kleinrock, Committee Co-chair

Peter Reiher, Committee Co-chair

University of California, Los Angeles

2014

Dedicated to my loving parents.

تقدیم به پدر و مادر عزیزم

تیرماه
۱۳۹۳

Table of Contents

List of Figures

List of Tables

Acknowledgments

I am eternally indebted to my loving parents. They unconditionally supported and helped me to become who I am now. I would like to dedicate to and thank specifically my mother, Tahereh, who in the past almost three decades of her life selflessly helped me and wanted me to succeed, perhaps even more than I ever did. *I owe you, mom.*

I am immensely grateful to my adviser, Peter Reiher, to be such an amazing mentor throughout my PhD. I cannot emphasize enough how fortunate I was to be his student. Through his constant guidance and advice I have learned to become a researcher in the field.

I had the highest privilege to receive occasional and yet invaluable advice and insights from Dr. Leonard Kleinrock. His advice and guidance made major contribution to the direction of my research.

My special thanks goes to Janice Martin-Wheeler, our lab assistant, who has patiently proofread and corrected my technical papers. I learned so much in the process and I will always be grateful for it.

This is also dedicated to my grandmother, who passed away before I even started college. She thought me, at my young age, the value and meaning of learning and teaching what you know to others. I never forget that she never stopped learning even until the day before she passed away.

Finally, many people, whom greatest friends who made various contributions to this project. My sincerest thanks go to them:

Alexander Afanasyev, Peter Peterson, Robert Chang, Erik Kline, Behnam Shahbazi, Matin Fouladinejad, and Mehdi Hosseini-Moghaddam.

This dissertation is dedicated to all of you.

Vita

2002	Supplemental Instruction Leader, Los Angeles Valley College
2006	Undergraduate Researcher, NASA Jet Propulsion Laboratory
2006	B.Sc., Electrical Engineering and Computer Science, University of California, Berkeley (with honors)
2007	Teaching Assistant, EECS Department, UC Berkeley. (Symbolic Programming and Discrete Mathematics)
2008	M.Sc., Computer Science, University of California, Berkeley
2008	Software Developer, PostPath Inc. (Cisco)
2008	Research Assistant, Laboratory for Advanced Systems Research. Computer Science Department, UCLA
2009	Teaching Assistant, Computer Science Department, UCLA (Introduction to CS I & II and Operating Systems Principles)
2010	Ph.D. Level Intern, Symantec Corporation
2014	Lecturer, Computer Science Department, UCLA
2014	Assistant Professor, Computer Science Department, California State University, Northridge

Publications

Pournaghshband, V., Afanasyev, A., and Reiher, P., “End-to-End Detection of Compression of Traffic Flows by Intermediaries,” *In Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, May 2014.

Pournaghshband, V., “Teaching the Security Mindset to CS 1 Students,” *In Proceedings of the 44th ACM technical symposium on Computer Science Education (SIGCSE)*, March 2013

Pournaghshband, V., Sarrafzadeh, M., and Reiher, P., “Securing Legacy Mobile Medical Devices,” *In Proceedings of the 3rd International Conference on Wireless Mobile Communication and Healthcare (MobiHealth)*, November 2012

Pournaghshband, V., Kleinrock, L., Reiher, P., and Afanasyev, A., “Controlling Applications by Managing Network Characteristics,” *In Proceedings of IEEE Communication and Information Systems Security Symposium (ICC)*, June 2012

Pournaghshband, V., and Natarajan, K., “A Robust Trust Model for Named-Data Networks,” *In Proceedings of International Conference on Security and Management (SAM)*, July 2011

Pournaghshband, V., “A New Watermarking Approach for Relational Data,” *In Proceedings of the 46th ACM Annual Southeast Regional Conference (ACM-SE)*, March 2008

1 Introduction

That fondness for science, ... that affability and condescension which is shown to the learned, that promptitude with which they are protected and supported in the elucidation of obscurities and in the removal of difficulties, has motivated me...

Khawrazmi (Algoritmi)

1.1 Thesis Statement

A distinctive and widely deployed class of third party middleboxes, for various reasons, influence traffic flows without modifying the end-host content, and are capable of leaving significant impacts on how the end-hosts communicate with each other. In this dissertation, we investigate how the two end-hosts, when working together, can detect network discriminators, a class of middleboxes that discriminate between network flows, by delaying or dropping packets within certain flows.

1.2 What Are Third-Party Middleboxes?

Abstractly, the Internet follows the end-to-end principle, with smart endpoints and a dumb network. However, the actual Internet is far more complex, with the emergence and rapidly growing prevalence of middleboxes deployed at various points in the network.

A middlebox, defined as "any intermediary device performing functions other than the normal, standard functions of an IP router on the datagram path between a source host and destination host [CB02]," manipulates traffic for purposes other than simple packet forwarding. In addition to routing the traffic, middleboxes are potentially capable of making serious changes to what happens to a traffic flow on the network. These changes have meaningful implications to the senders and receivers.

A wide variety of middleboxes have been proposed, implemented and deployed during the last decade [SHS12]. Today's enterprise networks rely on a wide spectrum of specialized applications of middleboxes. Middleboxes come in many forms such as proxies, firewalls, IDS, WAN optimizers, NATs, and application gateways and for various purposes including performance and security improvement and compliance. They are the integral part of today's Internet and play an important role in providing high levels of service for many applications. Recent papers have shed light on the deployment of these middleboxes [WQX11, SHS12] to show their prevalence. And a recent study [SRA12] shows that the number of different middleboxes in an enterprise network often exceeds the number of routers. Trends such as proliferation of smartphones and wireless video are set to further expand the range of middlebox applications [Rem].

Studying issues involving middleboxes has recently received special attentions from the research community. This is evident by a significant increase of publications related to middleboxes in just the past year. In fact, in December 2013, the first ACM HotMiddlebox workshop was held, which was the first workshop/conference primarily focusing on research issues related to middleboxes.

The focus of this dissertation is on detecting one class of third party middleboxes, *payload-preserving middleboxes*, that makes no changes to the content of the traffic, giving the appearance that nothing has been done to the stream other than routing it to the destination.

1.3 Why is Detecting Middleboxes Important?

Knowing the existence of such influence could be beneficial to the end-hosts. Sometimes the end-hosts would behave differently based on what they sense about what is happening to their traffic. In such cases, an accurate detection of what is happening to their traffic is the first step. Here, to illustrate this idea, we present a few scenarios from different categories.

Scenario I. Performance

The sender is about to send highly compressible data, making the compression worthwhile. The sender checks if an end-to-end compression or link compression on bottlenecks on the path is deployed. If he detects that compression is already in place, he would not compress the traffic stream, as double compression is redundant and costly. In a similar scenario, the sender might not encrypt if he detects that strong encryption is already in place by the gateways.

Scenario II. Security

The sender knows the receiver is using a wireless connection, but is unsure if that connection is secure. If he detects that the last link is unencrypted, he would either refrain from sending sensitive information or would apply end-to-end encryption to the channel. For example, Facebook, by default, does not provide end-to-end SSL encryption to its users, perhaps due to lack of available resources required to encrypt all users' contents for all users. Facebook servers, to use their resources effectively while protecting Facebook users' privacy, could first sense whether the user is using a secure wireless connection or not, and then apply end-to-end encryption only if the user needs that protection. Conversely, the receiver would mark the incoming data as untrusted if he detects the sender's wireless link is insecure.

Detecting the presence of a Man-in-the-Middle (MITM) is the first step to take any action against it or apply the necessary protection to their network flow to make potential attacks by the attacker ineffective. Therefore, it is crucial for the end-hosts to be able to detect the MITM's presence.

Scenario III. Politics

An Internet user in an oppressive country might detect Internet censorship imposed by his ISP and then chooses to use a proxy to bypass it. In a different scenario, an Internet user detects wire-tapping on his network and uses evading techniques such as Tor or VPN to make wire-tapping less effective.

Scenario IV. Debugging

Despite their growing importance in handling operational traffic, middleboxes are notoriously difficult and complex to manage [SHS12]. Hence, they are more prone to become a point of failure independent of network conditions. It is worth understanding if a problem with the middlebox (e.g., malfunction, misconfiguration, or overload) is causing the network to misbehave as a key step to debug the problem in hand.

The motivation behind detecting the influence of third parties by end-hosts is two-fold. The third parties could also benefit from such tools. Some make changes to the stream of traffic and want to assess the effectiveness of their changes by testing whether their changes are noticeable by end-hosts or not. This is mainly because they hope their changes to the network flow to be undetectable by the end-hosts. For instance, they can claim that performance is not heavily affected by security augmentation to the stream, if the end-hosts are unable to sense the change. Conversely, a stealth MITM attacker or an unauthorized eavesdropper aims to remain undetected by end-hosts throughout the attack.

1.4 Definitions

In this section, we define technical terminologies that we will be referring to throughout this document.

Definition 1. An *end-host* is one of the two communicating parties in the network. An end-host is capable of observing everything happening on its own machine (e.g., how its congestion control mechanism behaves).

Definition 2. A *network flow (or traffic stream)* between two end-hosts is uniquely identified by the 5-tuple of the source and destination addresses, port numbers, and the transport protocol type [RAJ11].

Definition 3. A *middlebox* is defined as any intermediary device performing functions other than the normal, standard functions of an IP router on the datagram path between a source host and destination host [CB02].

Definition 4. A *payload-preserving third party middlebox*[1] for a network flow between two end-hosts is an intermediary with the following characteristics:

1. The end-hosts receive exactly the same payload that the other party has sent. This means that the third party either does not modify the payload of any packets or undoes his changes before the packets get to the receiver[2].
2. The intention for influencing the network flow is more than just packet routing.
3. All packets in the network flow between the end-hosts go through the middlebox.

Following Definition 4, a third party middlebox is capable of doing (but is not limited to) the followings or a combination of them:

- Eavesdropping

[1] From now on in this document we refer to payload-preserving third party middleboxes as middleboxes.

[2] Note that middleboxes that drop packets are also included in this definition, whether the middlebox drops packets intentionally or unintentionally (e.g., as a result of saturated buffer queues).

- Delaying packets
- Dropping packets
- Compress/decompress payloads
- Encrypt/decrypt payloads
- Modifying link, IP, and/or transport layer headers
- Sending spoofed control packets to either parties
- Duplicating packets
- Combining packets
- Splitting packets

1.5 Middleboxes' Applications Taxonomy

In order to devise a non-technical taxonomy of this class of middleboxes, we observe the underlying principle behind why they exist along with their objectives rather than the technical characteristics of what they do to network streams. Figure 1.1 summarizes this taxonomy. In the remainder of this section we discuss each of the proposed categories.

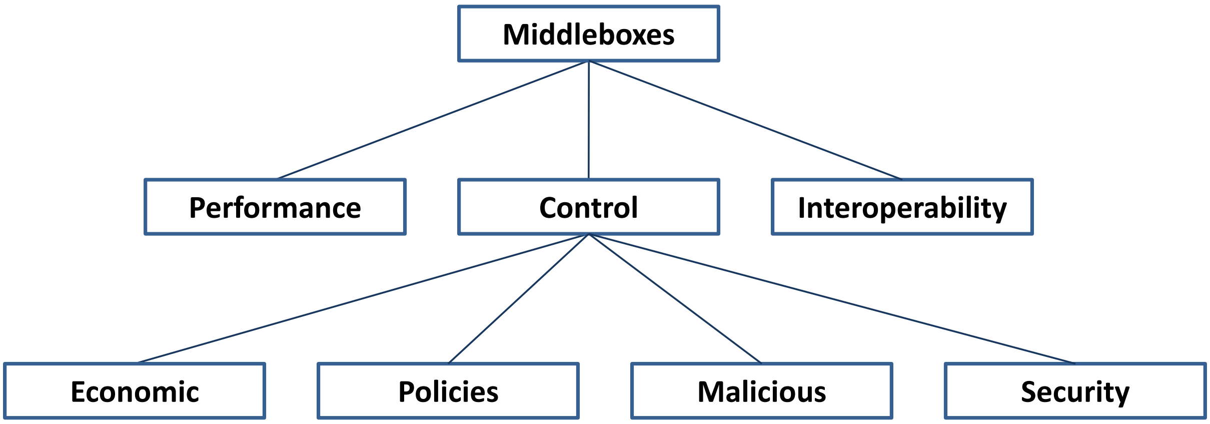


Figure 1.1: A non-technical taxonomy for middleboxes

1.5.1 Performance

Performance Enhancing Proxies (PEPs) [BKG01] are broad range of examples of such third parties for network performance improvement. PEPs are used in satellite communications (TCP Split [LSG04] and TCP Spoofing [IA01]), as well as in mobile network (ITCP [BB95], MTCP [YB94], and M-TCP [BS97]). PEPs are also used to cope with asymmetric links (ACK Filtering [BPK99] and ACK Decimation [R00]). These proxies are sometimes used to improve throughput of low bandwidth links by implementing various techniques such as compression or coding.

Transparent web proxies [Sha86] are another example of such third parties used to improve network performance.

1.5.2 Control

1.5.2.1 Economic

ISPs use a variety of tools (or a combination of such tools) for bandwidth management such as traffic shaping, policing, capping, throttling, classifying and conditioning. Another example is network address translators (NAT) which are widely used for the purpose of IP address management. Intermediaries also may impose proactive packet dropping to improve energy efficiency in wireless networks [CMN06].

1.5.2.2 Policies

Government or companies may employ various techniques to control the incoming and outgoing traffic as well as the traffic flows within their network. Internet censorship, firewalls, and wire-tapping are well-known examples of deployment of such third parties. These organizations are also capable of controlling network applications using various techniques such as network dissuasion [PKR12] and traffic classification. They can also specify regions of the Internet or countries they wish their network traffic to avoid [KR09].

1.5.2.3 Malicious

A man-in-the-middle (MITM) does not always modify the content of the packets. The attacker may simply eavesdrop on the traffic, and thus attack the confidentiality aspect of it. The attacker is also capable of causing service denial by performing either a high volume DoS attack or clever low volume DoS attacks (e.g., Shrew attacks [KK03]). Furthermore, Malicious selective packet dropping of critical network messages in wireless ad hoc networks can potentially paralyse the network by partitioning its topology [MSM09].

In a different class of attacks, called delay attacks, a malicious attacker in the middle deliberately delays the transmission of time synchronization messages to magnify the offset between the time of a malicious node and the actual time [SZC07].

1.5.2.4 Security

Practical examples of such influence by third parties are firewalls and VPNs (or more generally adding secrecy to the communication by encrypting the traffic flow). Packet marking (e.g., for IP traceback [VR10]) is another example of third party influences for security matters.

In our prior work [PSR12], we introduced a third party intermediary, Personal Security Device, which is a portable device to improve security for mobile medical systems. The device we developed requires no changes to either the medical device or its monitoring software. The personal security device is designed to seem transparent to both parties so it could offer protection for millions of existing devices.

1.5.3 Interoperability

Sometimes third parties modify a network flow or a subset of packets in a particular flow for interoperability purposes. Tunnelling for various reasons such as IPv6 and Mbone (Multicast backbone), and IP fragmentation are examples of this kind.

1.6 Dissertation Contributions

In this research we define and investigate the problem of detectability of a broad and widely deployed class of third party middleboxes that, for various reasons, influence traffic flows without modifying the end-host content. Payload-preserving middleboxes are capable of leaving potential serious impacts on how the end-hosts communicate, hence detecting their influence is beneficial and in some cases essential.

We have addressed this problem analytically by finding a common property between middleboxes and by devising a mathematical model of their interference with traffic flows. As an example, we present a loss-based approach to detect a general network discriminator, and use that to detect more specific middleboxes: traffic prioritization, network compression, and traffic shaping/policing.

In addition to analytical work, we implemented our approach in both simulation and using socket programming. We then validated our approach for every specific type of aforementioned middleboxes through extensive simulations (using `ns-3` [ns3]) and Internet experiments by testing our approach on a variety of cases and scenarios. In the simulation environment, we implemented new elements to simulate the middleboxes we attempt to detect. In the Internet experiment we used Click [KMC00], a software modular router, and an actual middlebox (Cisco Catalyst 3750 switch) in our lab at UCLA. The remote communicating parties were PlanetLab nodes, while the local party was located in our lab at UCLA.

1.7 Dissertation Organizations

Contributions of this dissertation and the organization of the remainder of the dissertation can be summarized as follows:

- We introduce the universal set of all payload-preserving third party middleboxes. We clearly define the many dimensions to the broader problem of detecting middleboxes, and the scope we focus on this dissertation (Chapter 2).

- We introduce network discriminators (that delay or drop packets within a particular network flow in a discriminatory fashion). We present an algorithm to detect this class of middleboxes, with thorough discussion on parameters and their values. Besides the general detection mechanism, our major contribution is a general framework in which any middlebox that can be analytically described as a *network discriminator*, with some condition on the magnitude of the discrimination, is also detectable (Chapter 3).

- We present a loss-based detection approach to detect network compression, traffic prioritization, and traffic shaping and policing by first analytically showing that they fit in the general framework. We then support our analytical work with network simulations and Internet experiments (Chapters 4, 5, and 6).

- We propose a detection mechanism for end-to-end detection of network compression by taking a purely delay-based approach. We show that using this approach network compression is detectable even without the cooperation of the receiver (Chapter 7).

- Based on lessons learned on preparing this dissertation, we recommend a set of steps that researchers who are interested in working on the problem of end-to-end detection of other middleboxes not addressed in this dissertation can take (Chapter 8).

2

The Universal Set

The whole is more than the sum of the parts.

Aristotle

Due to the fact that there are many different ways for constraining or more precisely defining the detection problem, the scope of the problem that we are presenting here is naturally very broad. In order to visualize an instance of this broad problem, we can think of the universal set of all different types and degrees of influence by third party middleboxes (Figure 2.1).

If we want to consider all aspects of this broad problem, we have to deal with many complexities. These types of problems are very complex because of their involvement with different dimensions. For sake of our discussion, we make this problem simple as much as we can by reducing it to only a single dimension that would concern only the network condition.

This is an abstract region within the universal set of influences ($D \subset U$) (Figure 2.1). Note that an influence in this region does not necessarily mean that it is "practically" detectable. That is why dotted line is used for the boundary between detectable region and the remaining part of the universal set ($U - D$). This remaining part is reserved for both unknown and fundamentally undetectable influences.

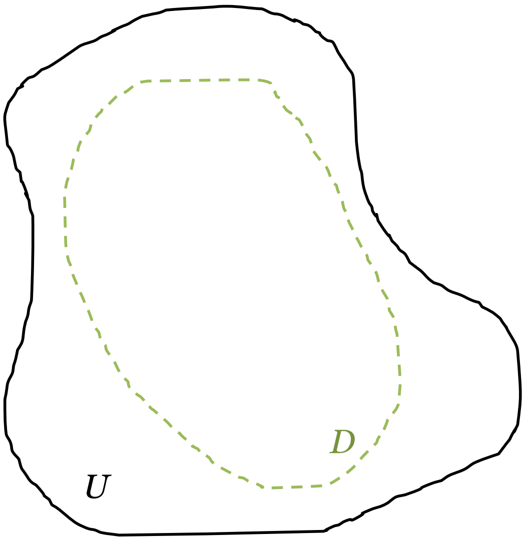


Figure 2.1: The detectable set, D, is a proper subset of the universal set of all influences, U.

We loosely define a detectable influence as follows: An influence is detectable if there exists an approach to detect the influence in a controlled and isolated environment when we remove as much variations as possible including but not limited to the following examples:

1. when the cross traffic and other Internet variabilities are absent.
2. when the end-hosts are running a limited OS sufficient enough to run our detection program with no extra overhead.
3. our code for detection mechanism is never swapped out or context switched.

This is an attempt to reduce all other variabilities to a point that we only experience those inherent to either the middlebox or our detection mechanism, and nothing else.

We acknowledge that this is a very constrained and arguably impractical environment to achieve. However, to show if an influence is detectable based on our definition we do not always have to go that far. For instance, if our detection mechanism works on a typical OS installed on the end-hosts, then we know it is certainly detectable in the limited specialized OS we defined in our definition. Also, we have to emphasize that the definition requires an actual middlebox. This is because we cannot be certain that the simulated middlebox in fact simulates all the important behavior of the middlebox.

2.1 Conditionally Detectable

We define *conditionally detectable* as a hypothetical subset of detectable set of influences. We use this definition solely for the purpose of illustrating our general idea (Figure 2.2).

A conditionally detectable middlebox's interference is an influence that is detectable under particular well-defined constraints over network conditions[1]. Looking at the figure, any influence inside that region ($CD \subset D \subset U$) is detectable under the defined network conditions.

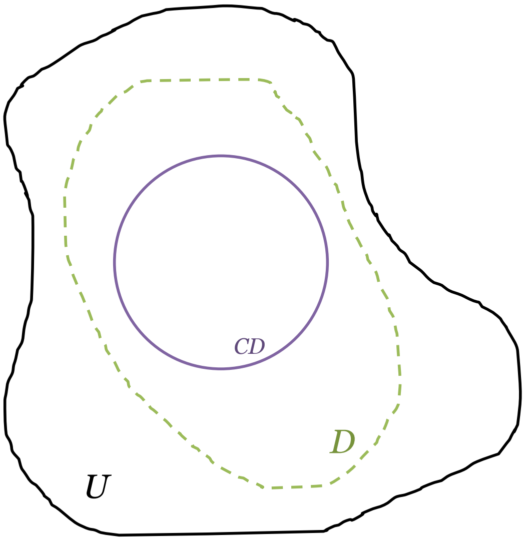


Figure 2.2: Conditionally detectable set of influences is contained in the detectable region.

Below we discuss two practical instances of conditionally detectable middleboxes:

- **Highly Detectable:**

  An influence is *highly detectable*, if it is detectable in almost all network conditions.

[1] The defined constraints also can be thought as goals that are required to be achieved, for someone who is attempting to devise a detection mechanism.

Therefore, it is always detectable because the effects of that influence is either extremely noticeable meaning it is detectable under any network condition (except perhaps when network is down,) or is not expected to be tampered with by normal intermediaries or influenced by network variations due its orthogonal nature.

An example of such influences is middleboxes that modify TCP header fields (e.g., TCP congestion window size). A simple detection mechanism can store the advertised TCP congestion window size in the payload (note that we are dealing with payload-preserving middleboxes, and hence by definition, the payloads are not altered.) The receiver can then compare the window size received in the header with the one stored in the payload to detect modification by a third party middlebox on the path. This effect is always detectable as long as the packet reaches the destination.

- **Normally detectable:**

  Cross traffic is usually present on the Internet and can sometimes significantly affect the network measurements including those performed for detection purposes. While the presence of cross traffic on the Internet is acknowledged, persistent cross traffic and network failures for long durations are not expected on the Internet. In addition, assuming that the core Internet is highly provisioned, we also do not expect high volume of traffic to cause network instability for a long period of time.

  We define *normally detectable* influences as those middleboxes' influences that are detectable under the aforementioned assumptions about the network condition.

  More specifically, an influence is normally detectable if it is detectable by performing a measurement at every hour throughout an entire day, resulting in a total of 24 measurements. We propose this pattern of measurements for normally detectable influences because it is known that the presence of cross traffic, on average, differs at certain times of the day and follows a particular pattern [SNC04]. Therefore, by performing measurements throughout the day, we expect to average out the effects of time-dependent cross traffic variation.

> The set of influences that their detectability is discussed in this dissertation is proven to be normally detectable.

In this section, we have defined state-based detectable influences that only concern the network conditions. As mentioned before this is a very complex topic as more dimensions could be added to the defined states. One might think of also including detection metrics (e.g., the detection and false positive rates) in the constraints as well. An example would be when a new detectable region is defined that imposes a tight upper bound on the false positive rate of the detection mechanism. This could be particularly interesting for detection in the security context where false positives are somewhat intolerable. Another possibility for increasing the dimension to the scope of the problem is the inherent characteristics of the detection measurement. For instance, one can think of imposing constraints on the number of measurements, or the level of intrusiveness by the approach (i.e., what percentage of the available bandwidth the measurement would use and for how long.) As another example, an influence is "passively" detectable if it is detectable using passive methods.

Finally, it is important to note that these states could overlap and be contained into another.

2.2 Theoretically and Practically Conditionally Detectable

In some cases, there are detectable regions that can be defined as two sets that one contains the other. This division is based on what is theoretically detectable (the super set, CD_T) and the inner region (CD_P) contains influences within that region (i.e., the super set) that are detectable in practice (Figure 2.3). To illustrate the idea, as an example we discuss the problem we address in this dissertation, which concerns the middleboxes that differentiate between packets of different flow by imposing some additional positive delay.

If the additional delay is greater than zero for every packet of the flow compared to the

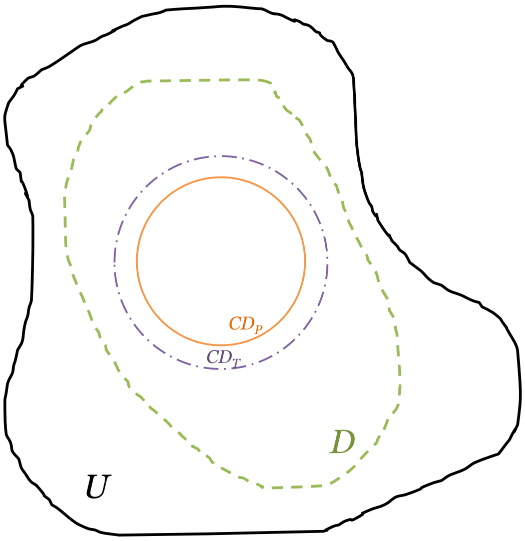


Figure 2.3: $CD_P \subseteq CD_T \subset D \subset U$.

other flow, it is considered discrimination, and in theory, this delay discrimination should be always detectable. However, in practice this is hardly the case. If the delay value is very small (e.g., 10 ns), then the end-hosts might not be able to observe the effects of the discrimination because (1) normal network delay variation is orders of magnitude higher than the value, (2) the existing tools do not have the sufficient accuracy or precision to capture such small values, and (3) resource and technological limitations on the end-hosts cause coarser granularity in the measurements (e.g., context switch effects). On the other hand, if the additional delay is relatively high (greater than some threshold, let's say Θ), then the end-hosts can detect the discrimination between flows by the third party (in the next chapter we present how).

In this example CD_T is the set of all influences that impose positive delay discriminations, while CD_P contains only those that are larger than Θ (Figure 2.5).

To detect network discrimination practically, one contribution of this dissertation is to come up with a suitable value for this threshold that serves as a lower bound, so that CD_P boundary is as close as possible to the boundaries of CD_T). In other words, we push the boundaries CD_P outward (Figure 2.6) so that it contains more influences (Figure 2.7). Widening the boundaries of CD_P to cover more areas of CD_T, however, is

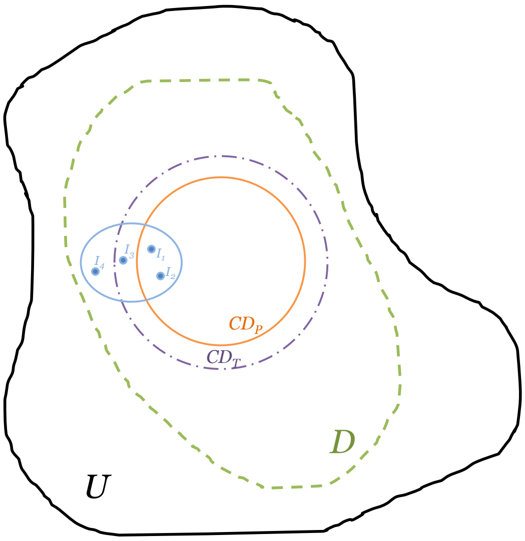


Figure 2.4: Specific influences spreading across the multiple regions.

more related to an area of complementary research to our work, which is to improve the accuracy of path property measurement tools.

Another contribution of this dissertation is to show that for some influences, through subtle formulations, we might be able to make the middlebox to behave in a way that its influence fall into the CD_P region (Figure 2.8), thus making it practically detectable.

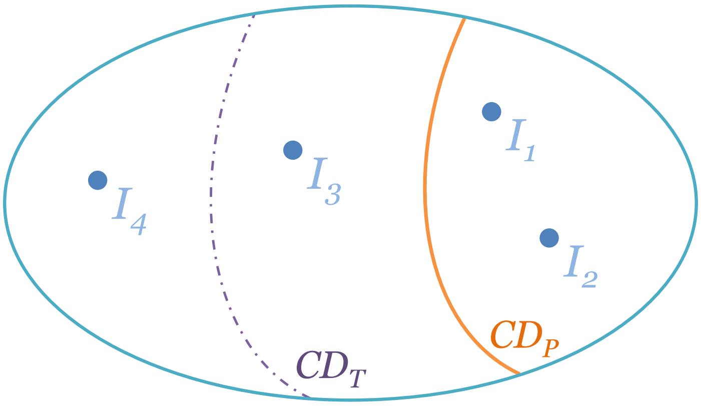


Figure 2.5: $CD_P : \delta > \Theta, CD_T : \delta > 0$, and influences beyond CD_T are influences that $\delta > 0$ have either high variations in delay (resulting in some negative values) or do not discriminate based on delay.

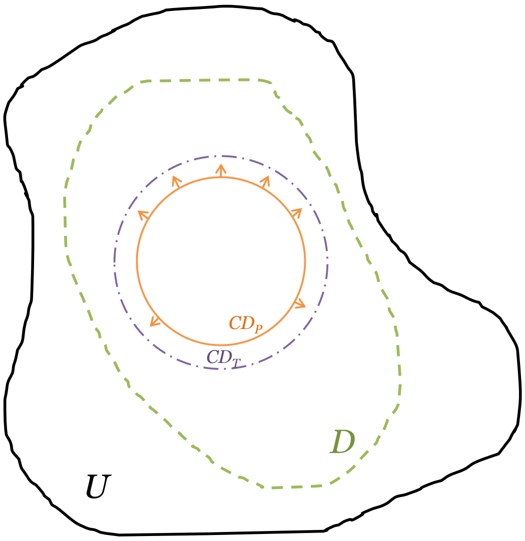


Figure 2.6: Pushing the boundaries of CD_P outward.

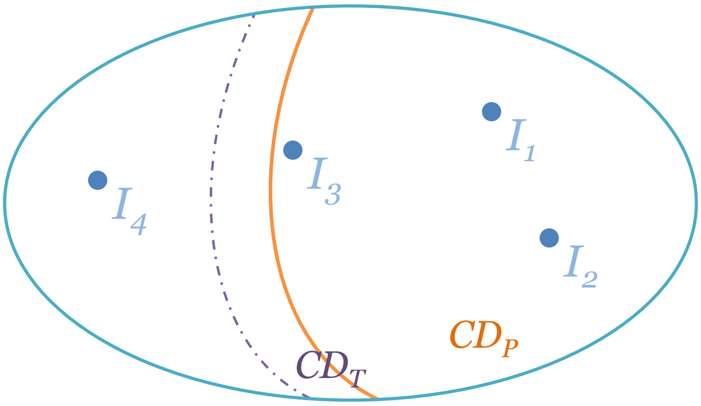


Figure 2.7: Widening the boundary of CD_P results in making influence I_3 practically detectable.

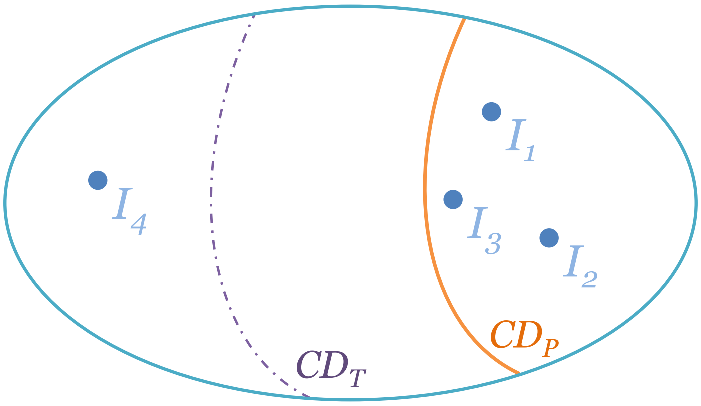


Figure 2.8: Moving the influence I_3 into the CD_P region results in making it practically detectable.

3 Detecting Network Flow Discrimination

Everything should be made as simple as possible, but not simpler.

Albert Einstein

The overall objective is to show that middleboxes (defined in this chapter as network discriminators) that influence traffic flows by delaying (or dropping) packets of different flows in a discriminatory fashion, are normally detectable[1]. In other words, given two network flows, one with property A, and the other with some property B, such middlebox imposes an additional processing delay on every single packet of the flow with property A greater than some positive constant, compared to packets of another flow with property B.

These middleboxes are not uncommon. Traffic prioritization, IP-level/link-layer compression, traffic shapers, and firewalls are just a few examples of such middleboxes that have been deployed by many ISPs worldwide.

As the first step to detect the influence of network discriminators, we describe their influence using a simple deterministic mathematical model. Utilizing this model, we then propose a loss-based detection mechanism to detect network discriminators. We then discuss thoroughly the parameters in our proposed detection mechanism and their theo-

[1] Note that normally detectable influences are defined in Section 2.1.

retical and practical values. Simulations and Internet experiments are used to evaluate our detection mechanism. Finally in this chapter, we present a general detection framework, in which any middlebox that could be described as a network discriminator, with some condition on the magnitude of the discrimination, is also detectable.

3.1 Notations

- $\mathcal{SND}$: The sender host.
- $\mathcal{RCV}$: The receiver host.
- P_A: A packet with a predefined property A and size ℓ.
- P_B: A packet with a predefined property B and size ℓ[2].
- $t_d(p)$: Departure time (time to transmit the last bit of packet p into the wire).
- $t_a(p)$[3]: Arrival time of packet p to the middlebox.
- $d_{MB}(p) := t_d(p) - t_a(p)$, total delay imposed by the middlebox on packet p.
- t_T: Transmission time is defined as network transmission time for transmitting the entire packet p into the MB's outbound link to the destination.
- c_p: The path capacity from $\mathcal{SND}$ to $\mathcal{RCV}$.

3.2 The Network Flow Discriminator

Definition 5. A pair of sets of packets $(\tilde{P_A}, \tilde{P_B})$ are said to be *delay-discriminatory* if:

- $\mid \tilde{P_A} \mid = \mid \tilde{P_B} \mid \neq 0.$

[2] Our analysis is based on the assumption that discrimination is not based on the packet size; however, some middleboxes *do* discriminate between flows based on packet sizes. For instance, many Cisco routers support `lt` command for classification based on packet size limit for QoS purposes. We consider detecting such interference by middleboxes out of the scope of this dissertation.

[3] $t_a(p)$ and $t_d(p)$ are based on relative clocks in a single system, hence synchronization is not required.

- $\exists P_{A_i} \in \tilde{P_A}, \exists P_{B_j} \in \tilde{P_B}$:

  $\delta_i := d_{MB}(P_{A_i}) - d_{MB}(P_{B_j}) \geq \delta_{min} > 0$

  $\wedge$

  $t_T(P_{A_i}) \geq t_T(P_{B_j})$

  $\wedge$

  $(\tilde{P_A} - \{P_{A_i}\}, \tilde{P_B} - \{P_{B_j}\})$ is also delay-discriminatory or is $(\emptyset, \emptyset)$.

It is important to note that this definition also covers middleboxes that drop packets in a discriminatory fashion such as traffic policers, network dissuasion [PKR12], and routers that employ Weighted RED (WRED) [FJ93, wre] for buffer management. In the case of discriminatory packet drops, $\delta_i = +\infty > 0$.

Definition 6. A pair of non-empty sets of packets ($\tilde{P_A}$, $\tilde{P_B}$) are said to be *pairwise-delay-discriminatory* if:

- $\forall i, j. P_{A_i} \in \tilde{P_A}, P_{B_j} \in \tilde{P_B}$:

  $d_{MB}(P_{A_i}) - d_{MB}(P_{B_j}) \geq \delta_{min} > 0$

  $\wedge$

  $t_T(P_{A_i}) \geq t_T(P_{B_j})$.

Theorem 1. If ($\tilde{P_A}$, $\tilde{P_B}$) is pairwise-delay-discriminatory, then ($\tilde{P_A}$, $\tilde{P_B}$) is also delay-discriminatory.

Proof. Choose any random mapping between P_{A_i}'s and P_{B_j}'s from $\tilde{P_A}$ and $\tilde{P_B}$. Any random mapping selected would satisfy the delay-discriminatory definition for ($\tilde{P_A}$, $\tilde{P_B}$) since by definition 6, $d_{MB}(P_{A_i}) - d_{MB}(P_{B_j}) \geq \delta_{min} > 0$ holds. □

Theorem 2. If ($\tilde{P_A}$, $\tilde{P_B}$) is pairwise-delay-discriminatory $\Rightarrow \tilde{P_A} \cap \tilde{P_B} = \emptyset$.

Proof. Proof by contradiction: We assume $\tilde{P_A} \cap \tilde{P_B} \neq \emptyset$. Then there exist a packet $\dot{P} \in \tilde{P_A} \cap \tilde{P_B}$. Since $d_{MB}(\dot{P}) - d_{MB}(\dot{P}) = 0 \not> 0$, then ($\tilde{P_A}$, $\tilde{P_B}$) is not pairwise-delay-discriminatory, by Definition 6. Therefore, our assumption was incorrect, hence $\tilde{P_A} \cap \tilde{P_B} = \emptyset$. □

Definition 7 (Network Flow Discriminator)**.** A middlebox is a *network flow discriminator* (or simply *discriminator*), if there exist a pair of non-empty sets of packets ($\tilde{P_A}$, $\tilde{P_B}$) that is *delay-discriminatory.*

3.3 Discriminator's Interference: A Mathematical Model

The past related work to solve the detectability problem of specific type of middleboxes had used experimental approaches. To our knowledge, our work is the first approach that looks into the problem of detectability of middleboxes employing analytical means.

The majority of interacting systems in the real world are too complicated to model in their entirety. Hence, the first level of compromise is to identify the most important parts of the system and find a right balance between greater precision (i.e., greater complexity) and simplicity. More specifically, the level of detail included in the model depends on our specific purpose (i.e., detecting the middlebox) for which the model will be used. For the purpose of detecting a certain class of middleboxes, a model that describes the influence is sufficient, and modelling the exact intrinsic behavior of these middleboxes is unnecessary.

In the previous section, we fundamentally described the network discriminators' interference using a deterministic model (rather than a stochastic model) to describe their interference on network flows. One of the key contributions of this work is this analytical approximation of the interference of network discriminators.

3.4 Assumptions

In our analysis in this chapter we make the following assumptions:

$$d_{np}(P_A) = d_{np}(P_B) \tag{3.1}$$

We assume that the routing nodal processing delay[4], d_{np}, is constant. In practice, we do not expect these values to be exactly equal, but in our measurements their difference in value is negligible.

We also assume that the transmission delay is linear with respect to packet size. The assumption that transmission delay is linear with respect to packet size may not be always true if, for example, a router manages its buffers in such a way that a 128 byte packet is copied proportionally faster than a 129 byte packet. However, this effect is usually small so it can be ignored.

We further assume that the network consists of a series of store-and-forward nodes; each of them equipped with a FIFO queue and has a constant service rate. The assumption that routers are store-and-forward (they receive the last bit of the packet before forwarding the first bit) is almost always true in the Internet [LB00].

Finally, we assume that other traffics do not cause the measurement packets to queue. In the Internet, this assumption is almost always false. The assumption arises from the deterministic nature of the model we introduce in this chapter. We do not know when the other packets are sent and of what size they are, so we cannot model them deterministically. This is a simplifying assumption to avoid unnecessary complexity in presenting fundamental aspects of our analytical presentation.

3.5 The Intrinsic Virtual Link of Network Discriminators

Figure 3.1, depicts the simple topology of the network between sender and receiver. X is the path capacity from the sender to MB[5], and similarly Z is the path capacity from MB to the receiver.

To investigate the detectability of the middleboxes, we look at the problem in a rather

[4] In a network based on packet switching, nodal processing delay is the time it takes routers to process the packet header. During processing of a packet, routers may check for bit-level errors in the packet that occurred during transmission as well as determining where the packet's next destination is. Processing delays in high-speed routers are typically on the order of microseconds or less. [RWW04]

[5] In practice, X is the minimum of the path capacity and the maximum sending rate can be achieved by the sender.

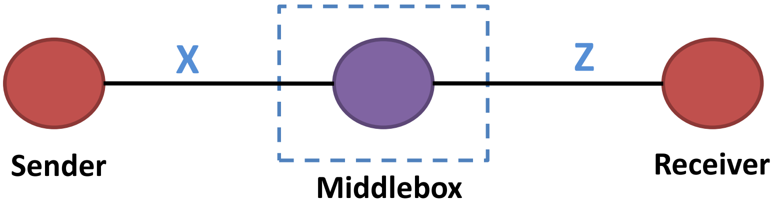


Figure 3.1: A middlebox that imposes additional delay on the path between sender and the receiver

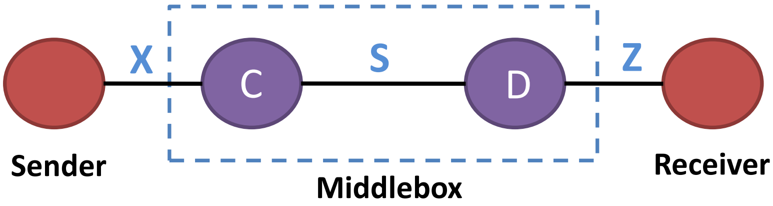


Figure 3.2: Virtual decomposition of the discriminator node into two nodes and a virtual link

different perspective. We imagine that the imposed delay by the discriminator has the same effect as if there was a link between two nodes C and D in the topology with some bandwidth S (Figure 3.2). Keep in mind that, since C and D are not real routers, C has a receive buffer but not a transmission queue, and D has a transmission queue but does not have a receive buffer.

For the purpose of illustrating our idea here, we make some simplification assumptions. First, we remind that we assume $|P_A| = |P_B| = \ell$. Secondly, MB adds some only "constant" delays D_A to packets of P_A, and D_B to packets of P_B, where $D_A > D_B$. Because we assume D_A and D_B are constant, then

$$\delta_{min} = \delta_i = D_A - D_B \tag{3.2}$$

For arriving packets with property A, the capacity of a virtual link S_A is[6]

[6] The factor "8" here is to match the units since bandwidth is expressed as bits per seconds and packet size is often expressed as number of bytes.

	Scenarios	Expected loss rate
1	$S < X \leq Z$	$1 - \frac{S}{X}$
2	$S < Z \leq X$	$1 - \frac{S}{X}$
3	$X < S < Z$	0
4	$X < Z < S$	0
5	$Z < S < X$	$1 - \frac{Z}{X}$
6	$Z < X < S$	$1 - \frac{Z}{X}$

Table 3.1: Expected loss rate for various cases when there is a delay D_A and its corresponding virtual link S

$$S_A = \frac{\ell * 8}{D_A} \tag{3.3}$$

And for arriving packets with property B, it is $S_B = \dfrac{\ell * 8}{D_B}$.

3.5.1 Detectability Under Different Network Characteristics

In this section, we examine detectability in various scenarios by comparing S_A (or S_B), X, and Z. We examine these scenarios where D_B is so small that $S_B > max(X, Z)$. For this reason, in examining this set of scenarios we ignore S_B in our analysis (since it does not affect the end-to-end network performance in this case), and we refer to S_A as just S. Table 3.5.1 enumerates all different scenarios in this condition.

In scenarios 1 and 2 in Table 3.5.1 , D_A is so large that makes S the narrow link on the path. If data is sent at the maximum rate, X, then the expected loss rate is $1 - \dfrac{S}{X}$, after the queue of MB is saturated.

In this chapter we show that scenarios 1 and 2 are detectable under certain conditions. The detection mechanism is based on the difference between the estimated loss measured for flows between the end-hosts with properties A and B independently.

3.5.2 The Role of c_p in Detecting Discriminators

The path capacity, c_p, between the sender and the receiver is $min(X, Z)$ and is estimatable by the end-hosts using existing tools. We assume that the sender (who already has an estimation of c_p) is sending two different sequence of packets independently, one with property A and the other with property B, each at the rate of c_p. Since $S_B > max(X, Z) = c_p$ and the sending rate of packets is c_p, no packets should be lost by MB. This is because the MB's queue will never saturate in the absence of cross traffic. This simplifies calculating the difference between the estimated loss ratios for A and B, to only that of A.

Analytically, there are essentially two cases that need to be examined to find the loss ratio for flow with the property A:

- $c_p = X$: The loss ratio difference in this case is $1 - \dfrac{S}{X}$ which is $1 - \dfrac{S}{c_p}$.
- $c_p = Z$: The loss ratio difference in this case is also $1 - \dfrac{S}{c_p}$, since the sender is assumed to send data at the rate of c_p and $c_p < X$.

This implies that the loss difference is $1 - \dfrac{S}{c_p}$. This value considers only the loss imposed by MB and overlooks the normal loss on the Internet. However, if the loss imposed by MB is the dominant one of the two, then we can detect discrimination in the networks by comparing the observed loss ratios. This suggests that if there exists a threshold, say τ, that distinguishes normal Internet loss and induced loss of this kind, then for a network discriminator to be detectable, it might be sufficient to show $1 - \dfrac{S}{c_p} \geq \tau$ holds, or equivalently:

$$S \leq (1 - \tau)c_p \tag{3.4}$$

Later in this chapter (Section 3.11.7) we show that a suitable value for this threshold is 20%, which leads to $S \leq \frac{4}{5}c_p$.

3.6 Queueing Theory: The D/D/1//Q Model

3.6.1 Notations

- Q: The queue size of MB (in packets).
- T_{start}: The time to start sending the detection probes.
- T_{end}: The time that the last detection probe is arrived to MB.
- r: The arrival rate of packets to the MB system.
- S: The service rate of packets at the MB system.
- n_I: The minimum number of packets needed to be sent to saturate the queue of MB.
- $\Delta_{A \cdot B}$: The time for the saturated queue to dissipate.

3.6.2 Queueing Theory Application

We assume the system is in the steady-state condition[7]. Our objective is to make the system unstable for a short time for the purpose of detection. We achieve that by making the arrival rate to the packets to the system larger than the service rate of the system.

To illustrate the basic idea behind our approach, we assume that the only traffic entering the middlebox (system) is our detection probes which are sent at a constant rate. Assuming that the service rate is constant (i.e., the internal variability of the middlebox is negligible and all packets are fast processed[8]), then both the arrival and service processes are deterministic. We assume the arrival rate to the middlebox is the sending rate, r, which is set by the sender. The service rate, S, is technically equivalent to the departure rate of the packets from the middlebox.

[7] Steady-state condition: after sufficient time has elapsed, the state of the system becomes essentially independent of the initial state and the elapsed time.

[8] In modern routers usually the first packet in a fresh flow is a cache miss, thus it is not fast processed. However, the subsequent packets are normally fast processed.

This simplified system is a basic system introduced in queuing theory “D/D/1//Q”. Note that since $r > S$ the system is unstable, and hence accumulates packets in its finite queue of size Q. Eventually, the maximum capacity is reached and packets will be dropped. That is the time that, as we will see in later sections in this chapter, the sender should start sending the detection probe packets.

The expected behavior of this system is illustrated in Figure 3.3. At T_{start}, the queue of the middlebox is saturated.

$$T_{start} = \frac{Q}{r - S} \tag{3.5}$$

This suggests that at least n_I packets are needed to be sent until this state is reached (queue is fully saturated), which is:

$$n_I = r \times T_{start} \tag{3.6}$$

Or (from 3.5),

$$n_I = \frac{rQ}{r - S} \tag{3.7}$$

T_{end} is when the last detection probe arrives at the system. $\Delta_{A \cdot B}$, is then the time for the queue to completely dissipate. To avoid interfering experiments resulting in distortion of data caused by the leftover effects from the prior experiment, we have to make sure that the next experiment does not run before this time. To calculate $\Delta_{A \cdot B}$ from the graph in Figure 3.3,

$$\begin{gathered} tan\theta = \frac{1}{S} = \frac{\Delta_{A \cdot B}}{Q} \\ \Delta_{A \cdot B} = \frac{Q}{S} \end{gathered} \tag{3.8}$$

And finally, L is the average number of packets in the queue as it accumulates before saturation:

$$L = \frac{1}{T_{start}} \int_{t=0}^{t=T_{start}} (rt - St)dt \tag{3.9}$$

3.6.3 On Exactness and Approximation

In the previous section, we presented the exact values and relationship between parameters of our system. In practice, this level of accuracy is not needed and in most cases not even obtainable. For instance, one could only estimate the queue size of an intermediate router rather than inferring the exact queue size. As we will discuss in later sections of this chapter, we will see that for our detection mechanism to work, these values only serve as lower bounds. For instance, $\Delta_{A \cdot B} \geq \frac{Q}{S}$ is required to hold to ensure independence between experiments. How much larger $\Delta_{A \cdot B}$ would be does not affect the detection accuracy.

3.7 A Loss-Based Approach to Detect Discriminators

3.7.1 Measuring the effects of the discriminator's interference

The first step to detect discriminators is to examine observable measurement metrics that are directly influenced by the middlebox's behavior. There are several candidate metrics to infer discriminatory delay, including packet loss, delay, or out-of-order packets. In some cases, any or all three metrics could be used to detect a particular discriminator. For instance, consider a strict priority queueing (SPQ) of two priorities, where the high priority queue is always served first. Low priority packets will experience larger loss rates and longer queueing delays than the high priority packets. Besides, a low priority packet may arrive to the middlebox earlier than a high priority packet but leave it after the high priority packet, while the opposite will never happen. This leads to reordering events that are clearly asymmetric.

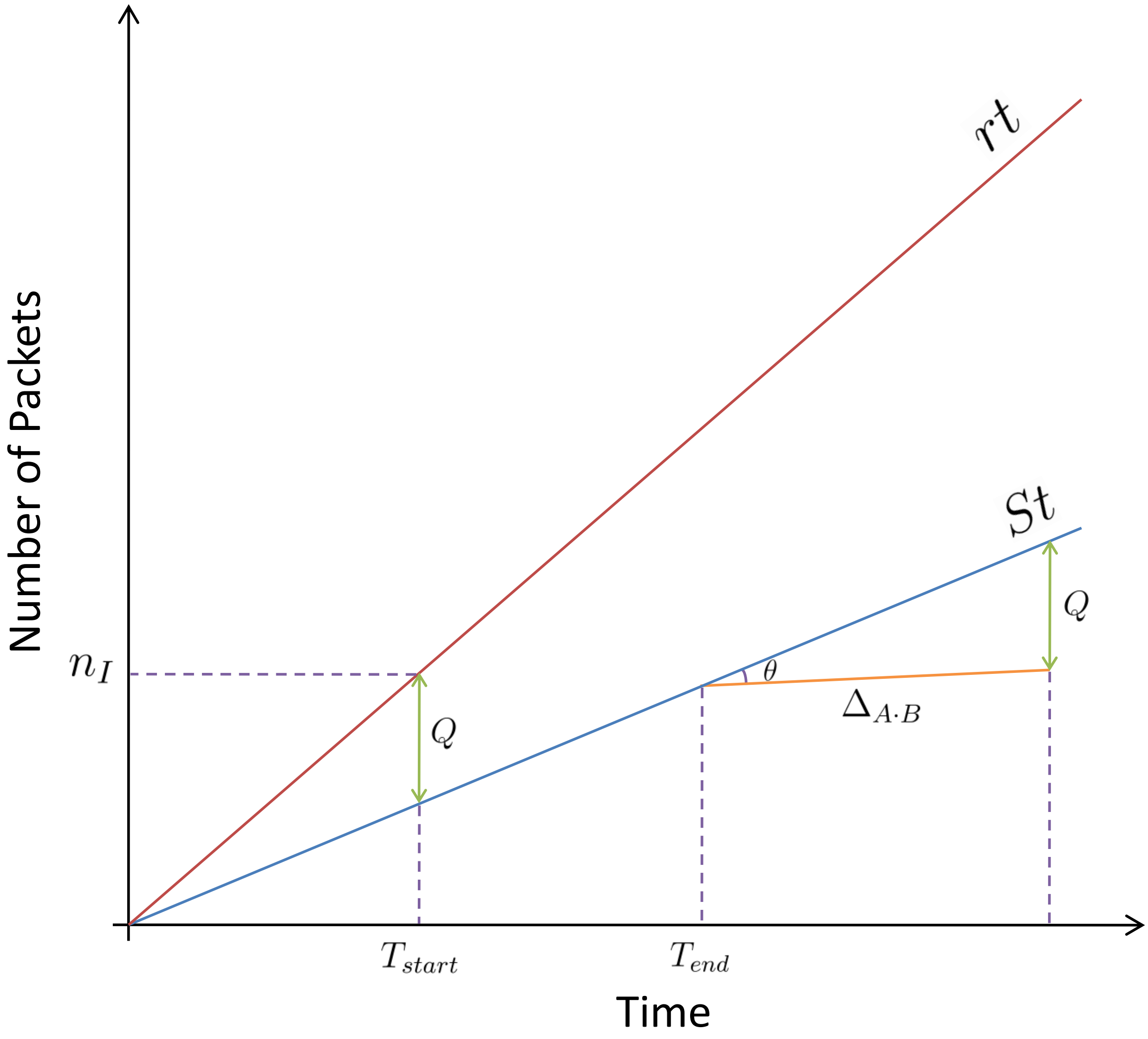


Figure 3.3: The D/D/1//Q Model

3.7.2 Packet loss as the inference metric

While, perhaps any or all three basic end-to-end performance metrics, loss, delay and out-of-order, are candidates for inference metrics, we have chosen to use packet loss as our inference metric.

The main reason behind our selection is mainly because the loss difference can be observed for the majority of discriminators (especially QoS mechanisms) while the other two cannot. Using delay and reordering metrics we cannot detect certain router QoS mechanisms simply because those mechanisms do not generate different delays at all. For instance, traffic policing does not either generate any packet reordering nor delay differences. However, any kinds of router QoS mechanisms will ultimately generate loss rates differences because that is the purpose of configuring such mechanisms. Two major disadvantages of using delay as an inference metric are (1) they often require synchronization of clocks between the end-hosts and (2) normal delay variation on the Internet may heavily affect the measurements.

On the other hand, using the loss metric has its disadvantages. The probe overhead of packet loss metric is larger than the other two, which is the major drawback of loss-based approaches. Obviously, loss rates difference will not become evident until the associated link (or a sublink for a traffic class) is saturated and begin to drop packets. This simple observation defines the basis of our loss-based inference approach: In order to reveal network discriminator's interference, one needs to saturate the path available bandwidth for a given class to produce loss rates difference among different classes. On the other hand, packet reordering and delay differences can be observed as soon as queue begins to build up.

In summary, although the loss-based method has larger probe overhead, it can detect a broader set of discriminators. However, there are cases that a delay-based approach works better. As we will see in Chapter 7, a delay-based compression can detect more network scenarios than a loss-based detection approach (which is presented in Chapter 4) can detect.

3.8 The Significance of δ_{min}

Our `ns-3` results confirm the significance of δ_{min} on the detection accuracy (Figure 3.4). For our simulation, we implemented a new module called "delay-discriminator" that simulates additional processing delay imposed by an intermediary on the path (refer to Appendix A for implementation details). The intuition behind large delays imposed by middleboxes is that the receive queue of the middlebox is expected to overflow (and hence drop packets) since it cannot process incoming packets fast enough.

As seen in both Figure 3.4 and 3.5, for middleboxes with $\delta_{min} \geq 1ms$, regardless of the packet probe size, they result in a very high loss ratio that makes the middlebox in question detectable. This suggests that in order to devise a detection mechanism to detect a particular network discriminator, one condition is to formulate the behavior of the middlebox such that it ensures $\delta_{min} \geq 1ms$.

One objective of this research is to find a suitable value for Θ such that if $\delta_{min} \geq \Theta$ holds (under all or certain network conditions) for some particular middlebox influence, then that middlebox is detectable. Achieving the minimum possible value for Θ is another future work direction.

However, as seen in Figure 3.6, $1ms$ difference in delay does not lead to detect discriminators for all cases. For instance, in the case of a middlebox that imposes $3ms$ delay for one class and $4ms$ for another class of traffic, the loss ratio for both cases are so high that the difference between the loss ratio is very low, hence not easily detectable using this method.

Therefore, this suggests that, in addition to the $\delta_{min} \geq \Theta$ requirement, there should be an upper bound for $d_{MB}(P_{B_i})$'s:

$$\forall i > 0, d_{MB}(P_{B_i}) < \theta \tag{3.10}$$

We chose θ empirically in our analysis, simulations, and experiments to be 10 μ-sec. Future work should improve this value.

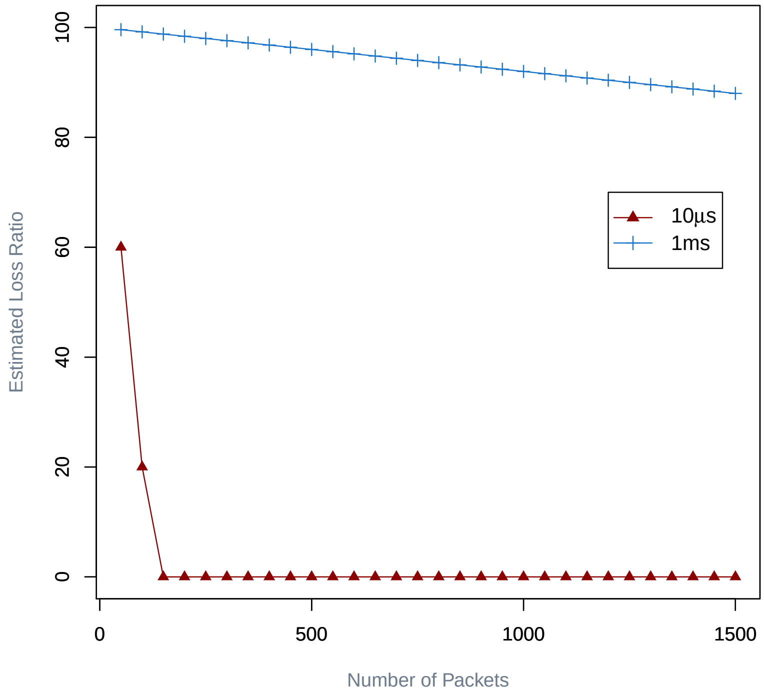


Figure 3.4: Loss ratio difference between when $\delta_i = 1\mu$s and $\delta_i = 1$ms in the simulated environment ($\ell = 100, c_p = 100$Mbps).

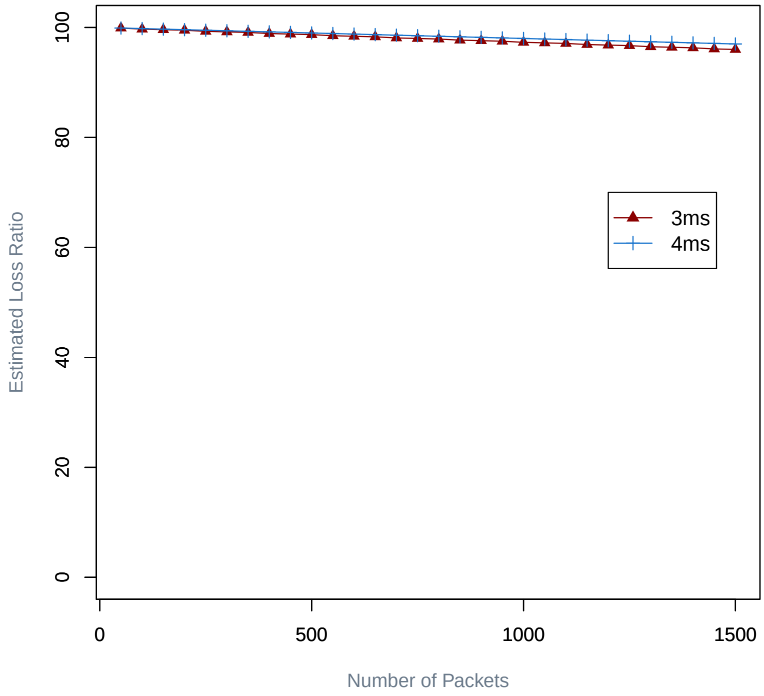


Figure 3.5: Loss ratio difference between when $\delta_i = 3$ms and $\delta_i = 4$ms in the simulated environment ($\ell = 100, c_p = 100$Mbps).

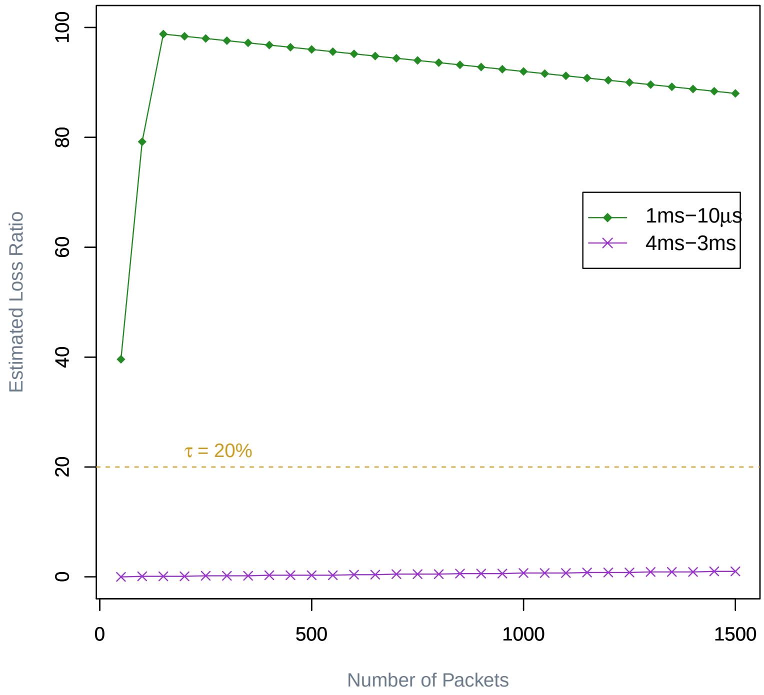


Figure 3.6: Detectability between scenarios ($\delta_i = 1\mu$s and $\delta_i = 1$ms) vs. ($\delta_i = 3$ms and $\delta_i = 4$ms) in the simulated environment ($\ell = 100, c_p = 100$Mbps).

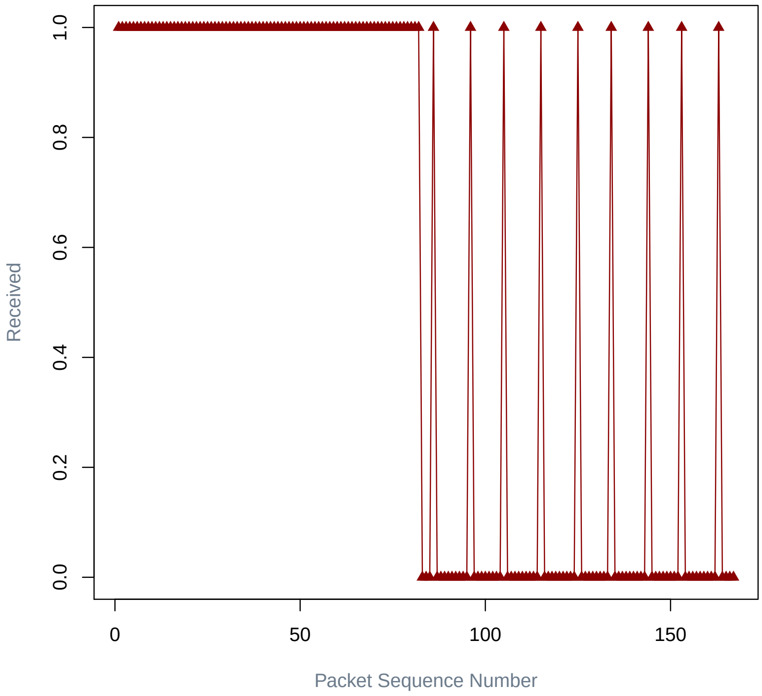


Figure 3.7: Observed loss pattern after queue saturation in the simulated environment ($\ell = 100, c_p = 100$Mbps).

An interesting observation is that the smaller the packet size, the higher the loss ratio for 10 μ-sec which makes it more detectable. This observation is summarized in a complete discussion presented later in this chapter.

3.9 The Significance of $\tilde{P_A}$ and $\tilde{P_B}$

As discussed in the previous section, the larger the δ_{min}, the more indicative is the presence of the middlebox. This suggests that, based on the equation in definition 5, we should define our sets $\tilde{P_A}$ and $\tilde{P_B}$ somehow that we get the maximum possible difference which would lead to the largest possible δ_{min} we can achieve. While ideally $\tilde{P_A}$ and $\tilde{P_B}$ should represent extreme cases, that level of extreme might not be entirely necessary to achieve high detection accuracy in all cases.

3.10 Detection Methodology Overview

In short the sender sends back-to-back packets (or probes) of packets with property A, pause for some time, and sends a train of packets with property B. The receiver keeps note of the packets that have been received or lost in the process. This process is repeated 24 times. At then end of the last round of measurements, the end-hosts, using the collected measurement data makes the detection decision.

In the next section we discuss the parameters used in this mechanism thoroughly, before moving to the formal presentation of our detection mechanism to detect network discriminators, which is followed immediately after the discussion on parameters.

3.11 Detection Methodology Parameters

3.11.1 Initial Packet Train,$(\mathcal{P}_{(A,B)_j})_{j=1}^{n_I}$

The detection probes are prelude by another train of packets, called the initial packet train, to form a longer train together. The sole objective of this packet train is to saturate the receive buffer queue of the middlebox, only if it is at all possible (i.e., if $min(r, X) \geq Z$). There are two characteristics of the initial packet train that requires careful consideration: the packets' content and the number of packets in the packet train. Below we discuss each:

Packet Content:

In the case of general traffic discriminator, the content of the packets should have the property A if preludes P_A probes, and property B if preludes P_B probes. However, this is not always the case to detect all discriminators. For instance, in detecting traffic prioritization (Chapter 5), packets for both initial packet trains (A and B) should have the property B.

Number of Packets, n_I:

Our goal here is to find a lower bound for n_I. Attempting to find such lower bound is valuable, since it means less packets need to be sent. Sending the minimum probe packets required for detection is desirable, since the length of the packet train contributes to intrusiveness of the detection approach. We emphasize that we only require an approximation (higher) and finding an exact value is not really needed here.

As discussed earlier, the main objective of initial packet train is to saturate the queue of the middlebox. If the sender cannot saturate that queue in either of the cases A and B, then the interference of the middlebox is not observable using loss as the metric, since it does not degrade that aspect of the network performance for the path between the sender and receiver.

Theorem 3. $n_I \geq \dfrac{8r}{r - c_p} \times Q$.

Proof. From our findings in Section 3.6.2 (earlier in this chapter) we know,

$$rt - St \geq Q \times 8\ell \tag{3.11}$$

Then we solve for t and we get $t \geq \dfrac{Q \times 8\ell}{r - S}$ which is technically T_{start}, as defined in Section 3.6.2.

$$T_{start} \geq \frac{Q \times 8\ell}{r - S} \tag{3.12}$$

Recall from Equation 3.5:

$$n_I \geq \frac{r \times T_{start}}{\ell} \tag{3.13}$$

Replacing T_{start} with our findings in Equation 3.6, we get:

$$n_I \geq \frac{8r}{r - S} \times Q \tag{3.14}$$

And because $S \leq \frac{4}{5}c_p$ (as derived earlier in this chapter), then $\dfrac{1}{r - c_p} \geq \dfrac{1}{r - S}$, and thus we have

$$n_I \geq \frac{8r}{r - c_p} \times Q \tag{3.15}$$

□

This theorem gives us a lower bound for n_I. An observation is that $r - c_p$ is in the denominator, so desirably r should be much larger than c_p but at the same time if r itself is too large then because it is in the numerator it can lead to other problems, requiring n_I to be large.

3.11.2 Detection Probes Sequence, $(\mathcal{P}_{A_j})_{j=1}^{M}$

$(\mathcal{P}_{A_j})_{j=1}^{M} = (\mathcal{P}_{A_1}, \mathcal{P}_{A_2}, ..., \mathcal{P}_{A_M})$ is a finite sequence of M ordered elements. The sequence of packet probes is constructed based on the Definition 3.2 such that $\mathcal{P}_{A_i} - \mathcal{P}_{B_i} \geq \delta_{min}$.

In the case of detecting network discriminator, since $M = n_I + n_P$: $(\mathcal{P}_{A_j})_{j=1}^{(n_I+n_P)} = (\mathcal{P}_{A_1}, \mathcal{P}_{A_2}, ..., \mathcal{P}_{A_{n_I}}, \mathcal{P}_{A_{n_I+1}}, ..., \mathcal{P}_{A_{(n_I+n_P)}})$. The sender essentially sends exactly these constructed packets exactly in the order that is presented.

3.11.3 Inter-Departure Time, λ

We set the inter-departure times to zero which means that the packets are sent back-to-back. This results in the sending rate, r, to be at maximum possible, unless it is rate-limited by the `tc` command.

3.11.4 Measurement Rounds, N_{rounds} and Δ_{rounds}

In the previous chapter, we discussed that our objective is to show that network discriminator is normally detectable. Accordingly, $N_{rounds} = 24$ and $\Delta_{rounds} = 1$ hour, meaning that the detection measurement takes place every hour for the span of 24 hours.

3.11.5 Inter-Measurement Time, $\Delta_{A \cdot B}$

As discussed in an earlier section, $\Delta_{A \cdot B}$ should be set to a value that independence between experiments is ensured. That is it should be large enough for the queue to dissipate from probes of the previous experiment. It can be set to a constant time in the order of tens of seconds. We chose rather empirically to be 30 seconds.

3.11.6 Estimating Loss Ratio, $\widehat{p}$

The task of performing statistical analysis for loss measurements on the Internet is known to be challenging, because Internet packet loss ratios are typically small and performing such analysis for small probabilities is always challenging. There have been extensive work in literature that present different models for model the loss on the Internet and many of them used active probing to measure loss rate on the Internet.

Similar to, we model the loss process on an Internet path as a continuous time binary stochastic process $I(t)$, where $I(t) = 1$ if at time t an arriving packet would be dropped, and $I(t) = 0$ otherwise.

For estimates to be valid, generally we must assume that this process is wide-sense stationary[9], which means that its mean, variance, and autocovariance are all constant with respect to t, for $\forall t.t \geq T_{start}$. This is true in our case, since $\mathcal{SND}$ is sending the probes at a constant rate and we assume our induced loss rate dominates the normal loss observed on the Internet.

The mean of I is just the loss ratio, which we denote as p, and can consequently be written as $p = \mathbb{E}[I(t)]$. Since $\mathcal{SND}$ sends n_P probes at times $t_1, \ldots, t_{n_P}$, it results in samples, $I_1, \ldots I_{n_P}$, thus the standard estimator for the loss ratio is simply:

$$\widehat{p} = \frac{1}{n_P} \sum_{i=1}^{n_P} I_i \tag{3.16}$$

Dshalalow et al. [Dsh95] showed that this estimate is guaranteed to converge to the true loss ratio , $p = \mathbb{E}[I(t)]$, when the loss process is stationary and ergodic[10]. It is important to note that we do not include the packets in the initial packet train $((\mathcal{P}_I)_{j=1}^{n_I})$ in estimating the loss ratio.

As future work, for our loss ratio estimation to be robust against the influence of cross traffic in our measurements, we will compare the expected loss pattern to the observed one, which, in addition to estimating the loss ratio, requires calculating the confidence interval, variance, and covariance for $I(t)$.

[9] A stationary process is a stochastic process whose joint probability distribution does not change when shifted in time. Consequently, parameters such as the mean and variance, if they are present, also do not change over time and do not follow any trends.

[10] A stochastic process is said to be ergodic if its statistical properties (such as its mean and variance) can be deduced from a single, sufficiently long sample (realization) of the process.

3.11.7 Detection Loss Ratio Threshold, τ

As mentioned earlier in this chapter, our proposed detection approach is loss-based. That means that the difference between the measured and estimated loss for the two flows in question is what determines the presence of a discriminator. This requires a pre-determined loss ratio threshold, τ, to compare the loss ratio difference to it. In this section we thoroughly show that a static threshold exists and discuss what a suitable value for this threshold is.

First we define the loss-based detectability function formally.

Definition 8 (Loss-Based Detectability Function)**.** The loss-based detectability function, $\mathscr{F}_l : \mathbf{A} \to \{0, 1\}$, is a Boolean function as defined:

$$\mathscr{F}_l(\mathbf{A} = \mathbf{a}) = \begin{cases} 1 & \Delta\widehat{p} \geq \tau \\ 0 & \Delta\widehat{p} < \tau \end{cases} \tag{3.17}$$

where $\mathbf{a}$ is the parameter values, and $\widehat{p}$ is the estimated loss ratio as described in Section 3.11.6. $\mathscr{F}_l$ returns 1 if the middlebox is detected, and 0 if not detected (or if it is unidentifiable).

Coming up with a correct loss ratio threshold, τ requires careful consideration since a major factor in accuracy of our detection mechanism is this value. In order to come up with a correct loss ratio threshold, we need to understand the root causes of loss on the Internet, and what is considered normal loss, and in contrast, what is considered unusually high loss rate.

Packet loss occurs when the buffer in a switch, router, or an end-host fills, when there are routing instabilities (e.g., as a result of routing misconfiguration), or when there are software/hardware failures at the end-hosts.

In 1998, Andren et al. [AHV98] reported loss percentage between 0.1% and 31% on the Internet based on their measurements, without providing any information about what traffic they used or the distribution of the observed loss ratio.

In 2001, Zhang et al. [ZD01] reported the following:

"We took two main sets of data, one during winter 1999-2000 (W1), and one during winter 2000-2001 (W2) ... Packet loss in the datasets was in general low. Over all of W1 measurements, 0.87% of the packets were lost, and for W2 measurements, 0.60%. However, as is common with Internet behavior, we find a wide range: 11-15% of the traces experienced no loss; 47-52% had some loss, but at a rate of 0.1% or less; 21-24% had loss rates of 0.1-1.0%; 12-15% had loss rates of 1.0- 10%; and 0.5-1% had loss rates exceeding 10%."

And lastly, Nguyen et al. in 2013 [NR13], after obtaining 5346 stationary traces between PlanetLab nodes and Web servers, reported that only 23 traces indicated loss between 0.01% and 20% and no traces was recorded with loss beyond 20%.

Based on previous work results and measurements mentioned in this section, we choose $\tau = 20\%$.

3.12 Detection Methodology Phases

Now we discuss the different phases of the experiment. The first three phases (pre-probing, probing, and post-probing) happen at every single round, and in that order. These phases are repeated N_{rounds} rounds, each Δ_{rounds} time apart. The final phase, the detection phase, happens only once and after all the N_{rounds} rounds of the experiment are completed.

3.12.1 Pre-Probing Phase

At pre-probing phase, the sender constructs $(\mathcal{P}_{A_j})_{j=1}^{M}$ and $(\mathcal{P}_{B_j})_{j=1}^{M}$. In order to do that, it might need to estimate a few parameters, such as n_I, n_P, and c_P (or perhaps more, depending on the nature of the discriminator we are trying to detect). The sender and receiver communicate through a pre-probing TCP connection to coordinate the probing phase of experiment parameters needed to be run.

3.12.2 Probing Phase

At the probing phase $\mathcal{SND}$ sends $(\mathcal{P}_{A_j})_{j=1}^{(n_I+n_P)}$, pauses for $\Delta_{A \cdot B}$, to ensure that the buffer queues are emptied of any packets from our last experiment, in order to avoid interfering experiments. Then, $\mathcal{SND}$ sends $(\mathcal{P}_{B_j})_{j=1}^{(n_I+n_P)}$.

3.12.3 Post-Probing Phase

At post-probing phase, $\mathcal{RCV}$, after some measurement data refinement, estimates the perceived loss ratio for each set of probes $(\mathcal{P}_{A_j})_{j=n_{I+1}}^{M}$ and $(\mathcal{P}_{B_j})_{j=n_{I+1}}^{M}$ and then stores the difference for final calculation which happen after the experiment is over, at the detection phase. This information maybe shared with $\mathcal{SND}$ by sending a summary of the received packets $(((\mathcal{P}_{A_j})_{j=1}^{M})', ((\mathcal{P}_{B_j})_{j=1}^{M})')$ via a post-probing TCP connection.

3.12.4 Detection Phase

This is the final phase. After running all the N_{rounds}, then we use a median of the stored values and compare it with τ to make its detection decision in this round. We find median as opposed to calculating the mean, to implicitly disregard the experiment runs that were highly affected by cross traffic or other network variations at the time the probing was taking place. More complex statistical outlier removal approach can replace the simple finding median approach, but our results presented later in this dissertation (see Section 4.4) show that this simple approach is sufficient for our purposes here.

3.13 Detection Algorithm

We summarize our detection mechanism proposed in this chapter in Algorithm 1.

Algorithm 1 Detecting Network Discriminators

detectDiscriminator($n_p, \tau, N_{rounds}, \Delta_{rounds}, \Delta_{A \cdot B}$)

```
for $i = 1$ to $N_{rounds}$ do
    # Pre-Probing Phase
    $c_p \leftarrow$ estimatePathCapacity()
    $n_l \leftarrow$ estimateInitialPacketTrainLength()
    $(\mathcal{P}_{A_j})_{j=1}^{M} \leftarrow$ constructAProbeSequence($n_l, n_p, c_p$)
    $(\mathcal{P}_{B_j})_{j=1}^{M} \leftarrow$ constructBProbeSequence($n_l, n_p, c_p$)

    # Probing Phase
    send$(\mathcal{P}_{A_j})_{j=1}^{M}$
    pause for $\Delta_{A \cdot B}$
    send$(\mathcal{P}_{B_j})_{j=1}^{M}$

    # Post-Probing Phase
    $\hat{p}_A^i \leftarrow$ estimateLossRatio($n_l, n_p, ((\mathcal{P}_{A_j})_{j=1}^{M})'$)
    $\hat{p}_B^i \leftarrow$ estimateLossRatio($n_l, n_p, ((\mathcal{P}_{B_j})_{j=1}^{M})'$)

    pause for $\Delta_{rounds}$
end for

# Detection Phase
if $\text{median}_{i \in \{1,2,\ldots,24\}}(\hat{p}_A^i) - \text{median}_{i \in \{1,2,\ldots,24\}}(\hat{p}_B^i) \geq \tau$ then
    return true
else
    return false
end if
```

3.14 Detection Framework

In order to show that a particular middlebox, MB, that influences network flows by imposing discriminatory delay or drops in a discriminatory fashion, is normally detectable, we believe it is sufficient to show:

1. MB is a network discriminator. In other words, we can construct (hence, there exist) a pair of sets ($\tilde{P_A}$,$\tilde{P_B}$) that is delay-discriminatory.
2. $\delta_{min} \geq \Theta$.
3. $\forall i, d_{MB}(P_B^i) \leq \theta$

As discussed before, Θ is a function of ℓ and c_p. ℓ is set by the end-hosts and can be set to a fixed value either 100 or 1000. Therefore, c_p is the only factor that needs to be estimated. As we could see in the plots, the higher the path capacity, the smaller δ_{min} can be to be detectable. Figure 3.8 depicts the relationship between δ_{min}, c_p, and ℓ for normally detectable influences, which is the region in the right of the curve:

$$\delta_{min} \geq \frac{10\ell}{c_p} \tag{3.18}$$

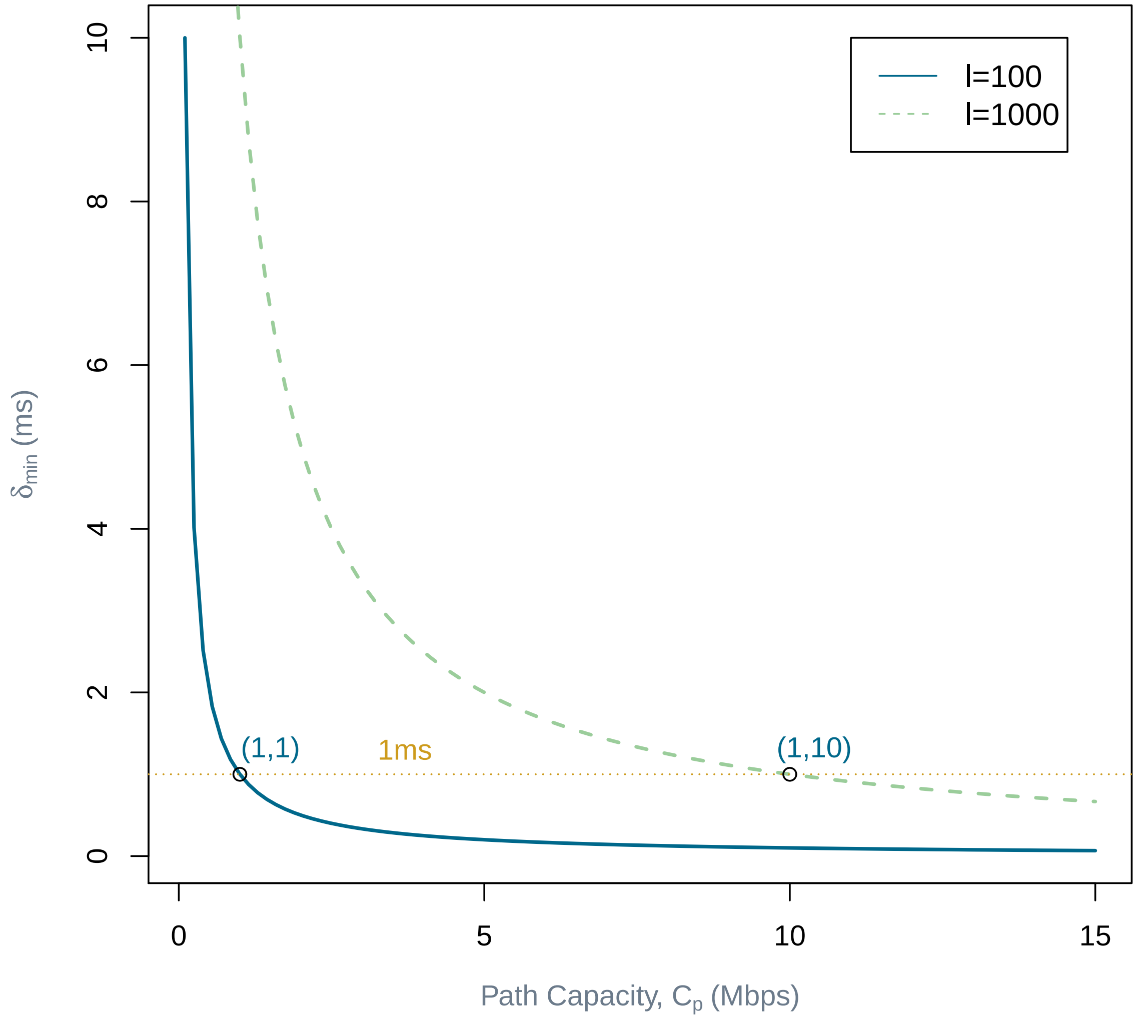


Figure 3.8: $\delta_{min} \geq \dfrac{10\ell}{c_p}$.

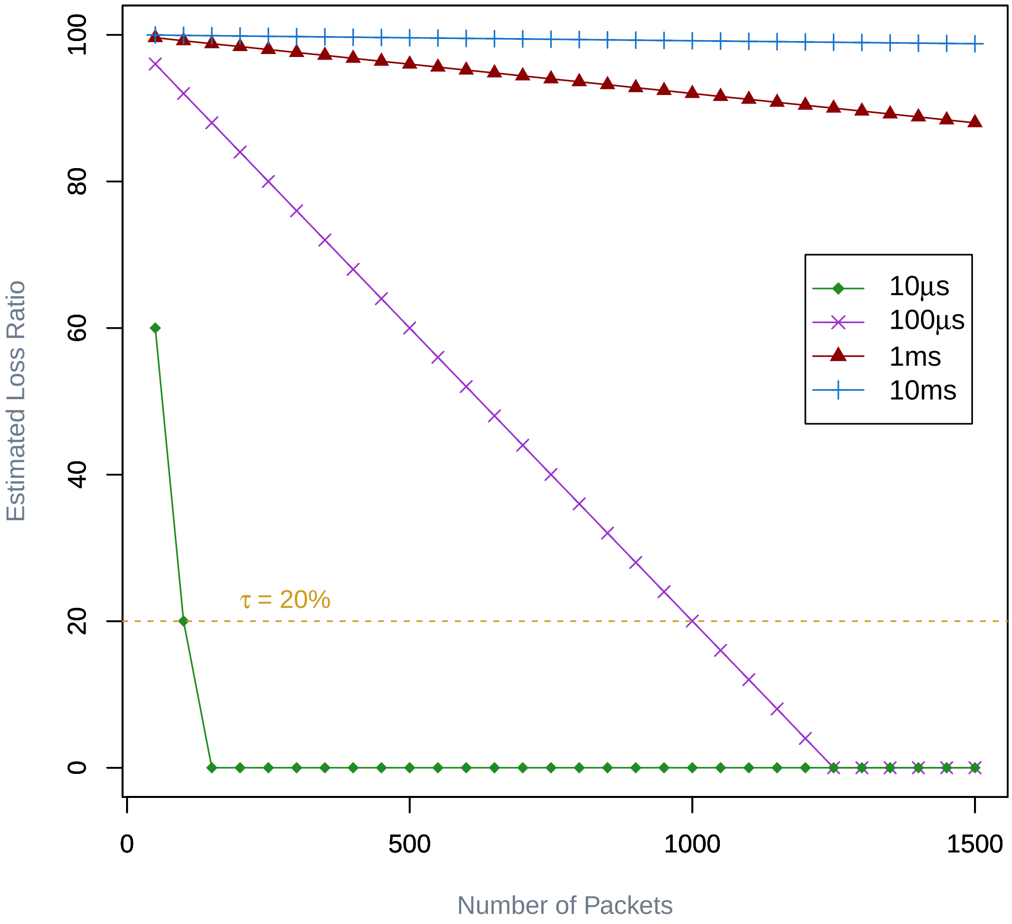


Figure 3.9: The effect of packet size and path capacity on detectability in the simulated environment ($c_p = 100$Mbps).

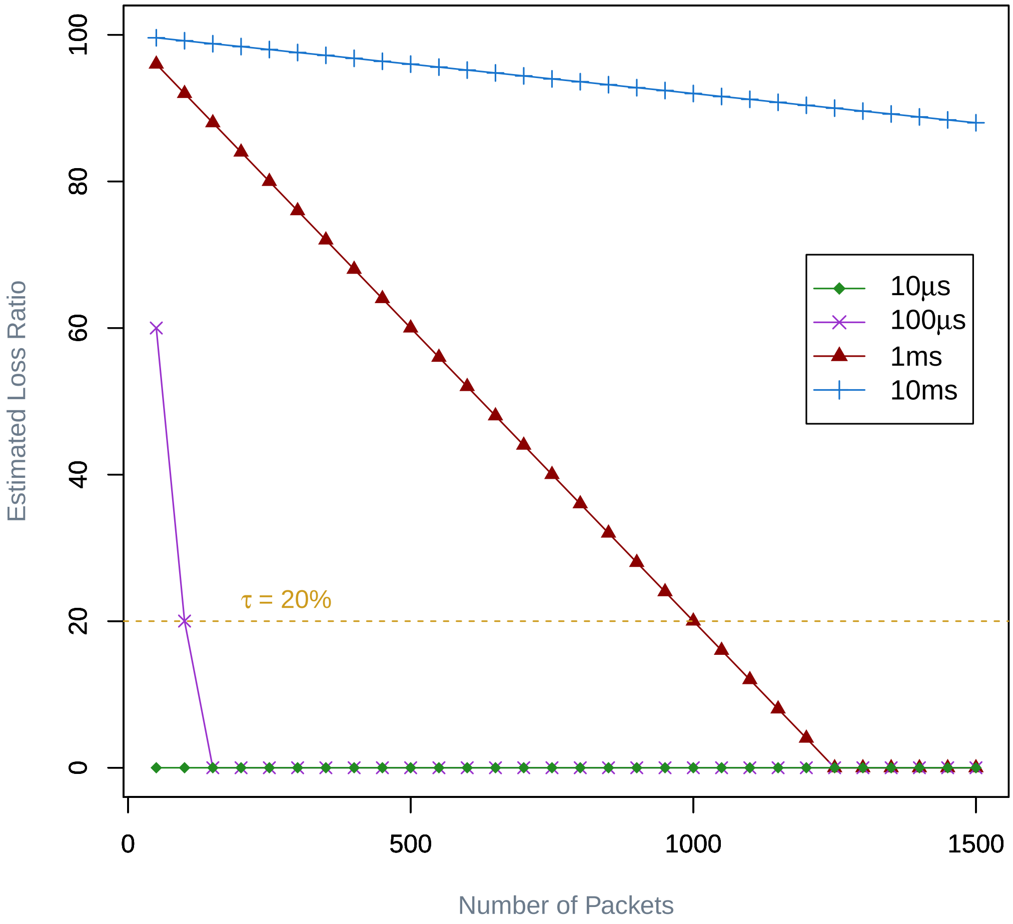


Figure 3.10: The effect of packet size and path capacity on detectability in the simulated environment ($c_p = 10$Mbps).

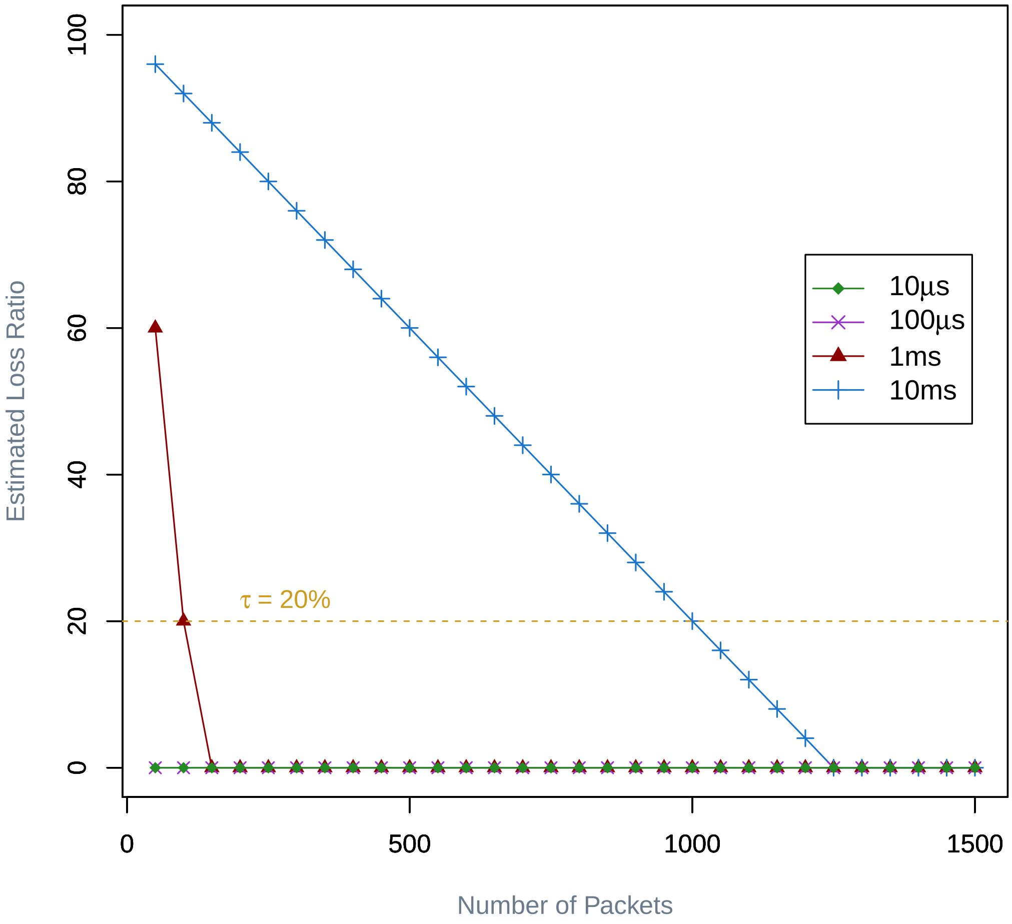


Figure 3.11: The effect of packet size and path capacity on detectability in the simulated environment (c_p = 1Mbps).

4 Detecting Network Compression: A Loss-Based Approach

Compression is desirable on slow connections,
but will only slow down things on fast networks.

The OpenSSH manual

4.1 Introduction and Motivation

An example of payload-preserving middleboxes is that of network compression which happens at intermediate nodes, rather transparently to the end-hosts. In this chapter we investigate the feasibility of detecting network compression on the path, using a loss-based approach. In Chapter 7, we provide a delay-based approach to detect network compression.

One way to increase network throughput is to compress data that is being transmitted. Network compression may happen at different network layers and in different forms:

Application layer. Compression at the application layer is widely used, specially for applications that use highly compressible data such as VoIP and video streaming. At the application layer, both compression and decompression happen at the end-hosts.

TCP/IP header. Often header compression is possible when there is significant redundancy between header fields; within the headers of a single packet, but in particular between consecutive packets belonging to the same flow. This is mainly achieved by first sending header field information that is expected to remain static for most of the lifetime of the packet flow. Since these methods are used to avoid sending unchanged header fields for a network flow, no data compression algorithm is actually applied here [CJ99].

Early TCP/IP header compressions such as CTP [Jac90] and IPHC [ECB99] were designed for slow serial links of 32 kbps or less to produce a significant performance impact [141]. More recent header compressions have since been developed, such as ROHC [JPS07].

IP payload. IPComp [SMP01] is the de facto method in this case. LZS [FM98], Deflate [Per98], and ITUT v.44 [BH01] are the well-known compression algorithms that work with IPComp. IPComp is generally used with IPsec. IP payload compression is something of a niche optimization. It is necessary because IP-level security converts IP payloads to random bitstreams, defeating commonly deployed link-layer compression mechanisms that are faced with payloads which have no redundant information that can be more compactly represented. However, many IP payloads are already compressed (images, audio, video, or zipped files being FTPed), or are already encrypted above the IP layer (e.g., SSL). These payloads will typically not compress further, limiting the benefit of this optimization. In general, application-level compression can often outperform IPComp because of the opportunity to use compression dictionaries based on knowledge of the specific data being compressed. This makes the mechanism less useful, and hence reduces the need for IPComp [Daw].

Link layer. Commonly used link compression schemes include the Stacker [FS96] and the Predictor [Ran96][141]. The Stacker compression algorithm is based on the Lempel-Ziv

[FM98] compression algorithm. The Predictor compression scheme works by predicting the next sequence of characters in a data stream using an index to look up a sequence in a compression dictionary. There is no information on how widely link-layer compression is used in practice.

Except for application-layer compression, compression happens at intermediate nodes, often without the knowledge of end-users. For example, in January 2013, a researcher discovered that Nokia had been applying compression to its users' data without their knowledge [nok13]. In this case, the intermediary was surreptitiously decrypting and re-encrypting user packets in order to effectively apply compression. Users surely would have preferred to know that this was happening, both because of the security risk and because it would render their own application-level compression unnecessary. However, performing compression and decompression requires many resources at the intermediate nodes, and the resulting overhead can overload the intermediary's queue, causing delay and packet losses. Further, not all commercial routers come with compression capabilities [HP]. Thus, some intermediaries apply compression, some do not, and generally they do not tell end-users whether they do. While managing resources effectively at end-hosts is not as crucial as it is at routers, it is still beneficial—particularly for mobile devices where resources are limited. Wasting these resources on redundant compression is undesirable. End-hosts can benefit from recognizing when compression has already being applied on a network connection.

Ideally, end-hosts and intermediaries should coordinate their compression decision, but practical problems make that ideal unlikely. Therefore, since the end-hosts have the greatest interest in proper compression choices for their data, they could detect if intermediate compression is present and adjust their behavior accordingly. An end-to-end approach to detect compression by third party middleboxes can help to save end-hosts' resources by not compressing data when intermediaries are already doing so. The savings are even greater in mobile applications.

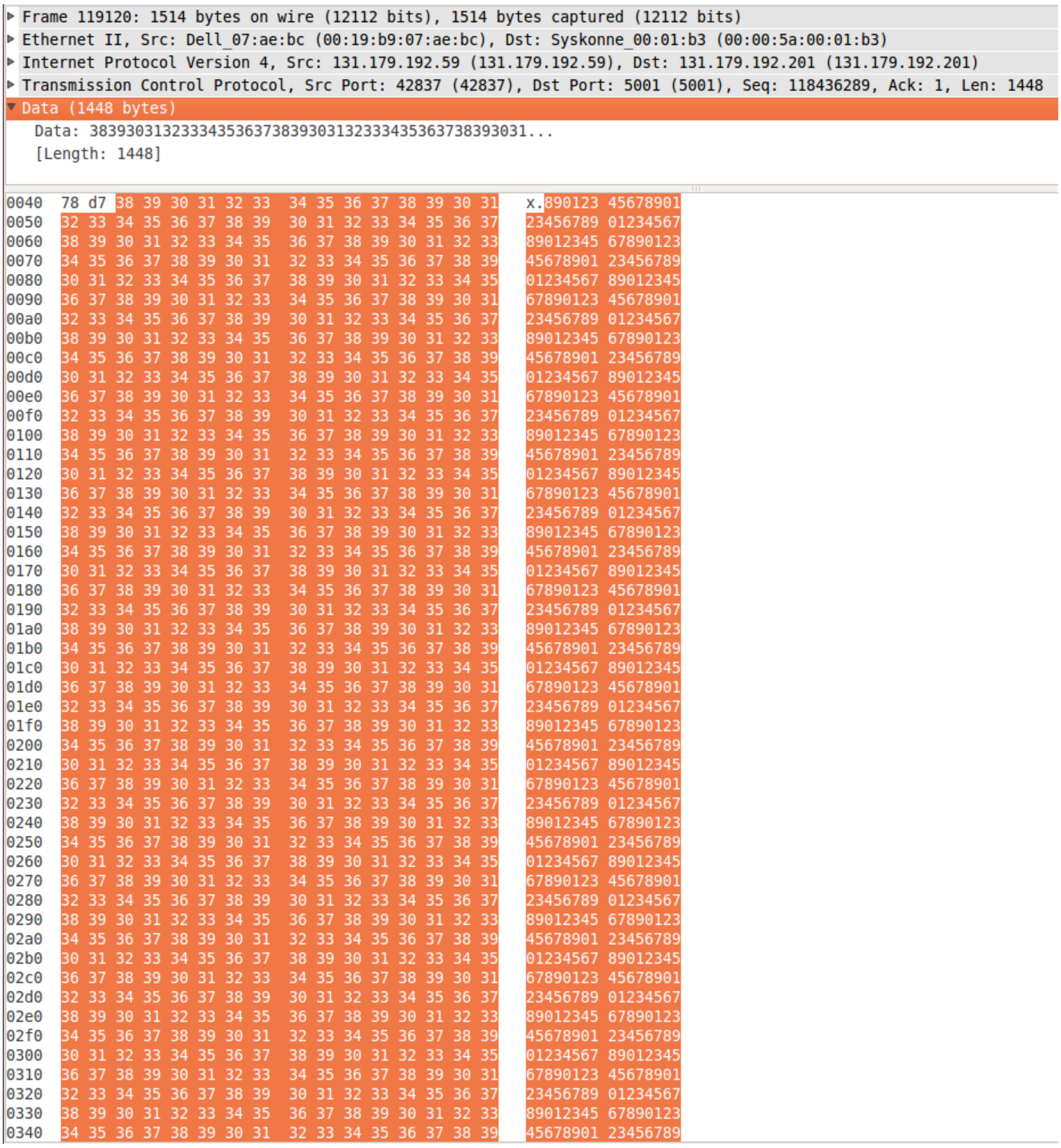
The approach requires no special network support. However, it is not designed to detect compressions that are not based on entropy of data, such as dictionary-based or TCP/IP packet header compression.

While IPComp is not widely used, there is no evidence related to how commonly link-layer compression is deployed. Knowing this is essential before performing further research in this direction, which suggests an investigation of the prevalence of link-layer compression as a next step. As our future work, we plan to use our findings here and in Chapter 7 in a longitudinal study of the prevalence of this type of compression in the Internet.

End-to-end detection of compression by intermediaries is also valuable for bandwidth availability and capacity estimation. Bandwidth availability and path capacity are among the most important characteristics of Internet paths. IP-level and link-layer compression directly influence the estimation of these path properties, since compression has a considerable effect on the assessment of both capacity and bandwidth. Not taking such effects into consideration will lead to bandwidth or capacity under- or overestimation. Currently, there is no common approach that active capacity or bandwidth measurement tools take for include in the content of their probes. For instance, two popular tools Iperf [ipe] and Pathrate [DRM01a] use rather different content. Iperf uses repeating "0123456789" sequence of characters (Figure 4.1), which is highly compressible, whereas pathrate that uses `srandom()` to fill up the payload, leading to high entropy data. Here is an excerpt from the pathrate source code, that fills the payloads of their measurement probes:

```
/* Create random packet payload to deal with links that do payload
   compression */
   srandom(getpid());
    for (i=0; i<MAX_PACK_SZ-1; i++)
      random_data[i]=(char)(random()&0x000000ff);
```

As the first and an important step to devise a loss-based approach to detect network compression, in the subsequent sections we attempt to achieve the objective described in



```
▶ Frame 119120: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits)
▶ Ethernet II, Src: Dell_07:ae:bc (00:19:b9:07:ae:bc), Dst: Syskonne_00:01:b3 (00:00:5a:00:01:b3)
▶ Internet Protocol Version 4, Src: 131.179.192.59 (131.179.192.59), Dst: 131.179.192.201 (131.179.192.201)
▶ Transmission Control Protocol, Src Port: 42837 (42837), Dst Port: 5001 (5001), Seq: 118436289, Ack: 1, Len: 1448
▼ Data (1448 bytes)
    Data: 383930313233343536373839303132333435363738393031...
    [Length: 1448]

0040  78 d7 38 39 30 31 32 33  34 35 36 37 38 39 30 31   x.890123 45678901
0050  32 33 34 35 36 37 38 39  30 31 32 33 34 35 36 37   23456789 01234567
0060  38 39 30 31 32 33 34 35  36 37 38 39 30 31 32 33   89012345 67890123
0070  34 35 36 37 38 39 30 31  32 33 34 35 36 37 38 39   45678901 23456789
0080  30 31 32 33 34 35 36 37  38 39 30 31 32 33 34 35   01234567 89012345
0090  36 37 38 39 30 31 32 33  34 35 36 37 38 39 30 31   67890123 45678901
00a0  32 33 34 35 36 37 38 39  30 31 32 33 34 35 36 37   23456789 01234567
00b0  38 39 30 31 32 33 34 35  36 37 38 39 30 31 32 33   89012345 67890123
00c0  34 35 36 37 38 39 30 31  32 33 34 35 36 37 38 39   45678901 23456789
00d0  30 31 32 33 34 35 36 37  38 39 30 31 32 33 34 35   01234567 89012345
00e0  36 37 38 39 30 31 32 33  34 35 36 37 38 39 30 31   67890123 45678901
00f0  32 33 34 35 36 37 38 39  30 31 32 33 34 35 36 37   23456789 01234567
0100  38 39 30 31 32 33 34 35  36 37 38 39 30 31 32 33   89012345 67890123
0110  34 35 36 37 38 39 30 31  32 33 34 35 36 37 38 39   45678901 23456789
0120  30 31 32 33 34 35 36 37  38 39 30 31 32 33 34 35   01234567 89012345
0130  36 37 38 39 30 31 32 33  34 35 36 37 38 39 30 31   67890123 45678901
0140  32 33 34 35 36 37 38 39  30 31 32 33 34 35 36 37   23456789 01234567
0150  38 39 30 31 32 33 34 35  36 37 38 39 30 31 32 33   89012345 67890123
0160  34 35 36 37 38 39 30 31  32 33 34 35 36 37 38 39   45678901 23456789
0170  30 31 32 33 34 35 36 37  38 39 30 31 32 33 34 35   01234567 89012345
0180  36 37 38 39 30 31 32 33  34 35 36 37 38 39 30 31   67890123 45678901
0190  32 33 34 35 36 37 38 39  30 31 32 33 34 35 36 37   23456789 01234567
01a0  38 39 30 31 32 33 34 35  36 37 38 39 30 31 32 33   89012345 67890123
01b0  34 35 36 37 38 39 30 31  32 33 34 35 36 37 38 39   45678901 23456789
01c0  30 31 32 33 34 35 36 37  38 39 30 31 32 33 34 35   01234567 89012345
01d0  36 37 38 39 30 31 32 33  34 35 36 37 38 39 30 31   67890123 45678901
01e0  32 33 34 35 36 37 38 39  30 31 32 33 34 35 36 37   23456789 01234567
01f0  38 39 30 31 32 33 34 35  36 37 38 39 30 31 32 33   89012345 67890123
0200  34 35 36 37 38 39 30 31  32 33 34 35 36 37 38 39   45678901 23456789
0210  30 31 32 33 34 35 36 37  38 39 30 31 32 33 34 35   01234567 89012345
0220  36 37 38 39 30 31 32 33  34 35 36 37 38 39 30 31   67890123 45678901
0230  32 33 34 35 36 37 38 39  30 31 32 33 34 35 36 37   23456789 01234567
0240  38 39 30 31 32 33 34 35  36 37 38 39 30 31 32 33   89012345 67890123
0250  34 35 36 37 38 39 30 31  32 33 34 35 36 37 38 39   45678901 23456789
0260  30 31 32 33 34 35 36 37  38 39 30 31 32 33 34 35   01234567 89012345
0270  36 37 38 39 30 31 32 33  34 35 36 37 38 39 30 31   67890123 45678901
0280  32 33 34 35 36 37 38 39  30 31 32 33 34 35 36 37   23456789 01234567
0290  38 39 30 31 32 33 34 35  36 37 38 39 30 31 32 33   89012345 67890123
02a0  34 35 36 37 38 39 30 31  32 33 34 35 36 37 38 39   45678901 23456789
02b0  30 31 32 33 34 35 36 37  38 39 30 31 32 33 34 35   01234567 89012345
02c0  36 37 38 39 30 31 32 33  34 35 36 37 38 39 30 31   67890123 45678901
02d0  32 33 34 35 36 37 38 39  30 31 32 33 34 35 36 37   23456789 01234567
02e0  38 39 30 31 32 33 34 35  36 37 38 39 30 31 32 33   89012345 67890123
02f0  34 35 36 37 38 39 30 31  32 33 34 35 36 37 38 39   45678901 23456789
0300  30 31 32 33 34 35 36 37  38 39 30 31 32 33 34 35   01234567 89012345
0310  36 37 38 39 30 31 32 33  34 35 36 37 38 39 30 31   67890123 45678901
0320  32 33 34 35 36 37 38 39  30 31 32 33 34 35 36 37   23456789 01234567
0330  38 39 30 31 32 33 34 35  36 37 38 39 30 31 32 33   89012345 67890123
0340  34 35 36 37 38 39 30 31  32 33 34 35 36 37 38 39   45678901 23456789
```

Figure 4.1: Iperf bandwidth estimation tool payload content captured by Wireshark

Chapter 3.

4.2 Notations

In this section we present a list of all notations that will be used in this chapter. Some symbols will be defined later in the chapter. Note that we modified the definition for $t_d(p)$ from the original definition in Section 3.1. This new definition affects $d_{MB}(p)$. We introduce a new terminology, *perceived link capacity*, as the link capacity sensed by the end-hosts.

- $\mathcal{SND}$: The sender host.
- $\mathcal{RCV}$: The receiver host.
- P_L: The packet with a low entropy payload and size ℓ.
- P_H: The packet with a high entropy payload and size ℓ.
- P'_L: The resulting packet after after compression applied to P_L.
- P'_H: The resulting packet after after compression applied to P_H.
- $\alpha := \dfrac{|P'_L|}{|P'_H|}$, compression efficiency ratio.
- Y_{Le}: The *perceived link capacity* when a packet train of P_L (low entropy data) are sent.
- Y_{He}: The *perceived link capacity* when a packet train of P_H (high entropy data) are sent.
- $t_d(p)$: Departure time (time to transmit the last bit of packet p into the wire).
- $t_a(p)$[1]: Arrival time of packet p to the middlebox.
- $d_{MB}(p) := t_d(p) - t_a(p)$, total delay imposed by the middlebox on packet p.

[1] $t_a(p)$ and $t_d(p)$ are based on relative clocks.

- t_T: Transmission time is defined as network transmission time for transmitting the entire packet p into the MB's outbound link to the destination.

- TxQ_i: Transmission queue of the i-th link on the path.

- c: The link capacity of the compression link (which is the path capacity if the compression link is the bottleneck).

- c_p: The path capacity from $\mathcal{SND}$ to $\mathcal{RCV}$.

- $d_c(p)$: The additional processing delay to compress the packet p.

4.2.1 Construction of $\tilde{P_A}$ and $\tilde{P_B}$

In our detection experiment described in this chapter we will use multiple reads from `/dev/urandom` to ensure that we capture high entropy data the majority of the time[2]. And as for low entropy, we used payloads consisting of all zero bytes. However, in our analytical work we must ensure $\tilde{P_A} \cap \tilde{P_B} \neq \emptyset$. That means that we must avoid even the slim probability $\Pr[\mathcal{P}_{L_i} = \mathcal{P}_{H_i}] = \frac{1}{2^{\ell * 8}}$. As a matter of fact, it is not sufficient to avoid that case. Since we are interested in capturing the effects of extreme cases, we also need to avoid a more general undesirable effect and that is if the entropy of $\mathcal{P}_{H_i}$ is not sufficiently high. For this reason, assuming that we could calculate the entropy of payloads using a method `calculateEntropy()`[3], the sender runs the following code until it achieves the entropy desired to be used for P_H in the detection[4]:

[2] To avoid the effects of dictionary-based compression, a different read from `/dev/urandom` for every packet is required.

[3] This function returns the data compression ratio achieved by Lempel-Ziv algorithm. One might argue that Shannon bound is a better selection since it is compression algorithm-agnostic. For our purposes here, this level of compression is sufficient, specially that Stacker, one of the two widely used link-layer compression mechanisms uses the LZ compression [141]. Note that Shannon entropy gives the optimal compression rate that can be approached but not improved and is defined as $H = \sum_{i=1}^{n} p_i \log_2 P_i$, where p_i is the probability that any given character (byte) in the file is character i.

[4] `/dev/urandom` is non-blocking and uses the internal pool to produce pseudo-random bits. Therefore, it is guaranteed to return every time it is called, but at the cost of perhaps lower entropy data [ura]. To alleviate the problem of P_H not containing sufficiently enough entropy that was addressed earlier, we run the function repeatedly until data sufficient entropy is obtained.

Algorithm 2 Initializing P_H payload with high entropy byte stream

```
repeat
    $P_H \leftarrow$ read from /dev/urandom
until $calculateEntropy(P_H) < \gamma$
```

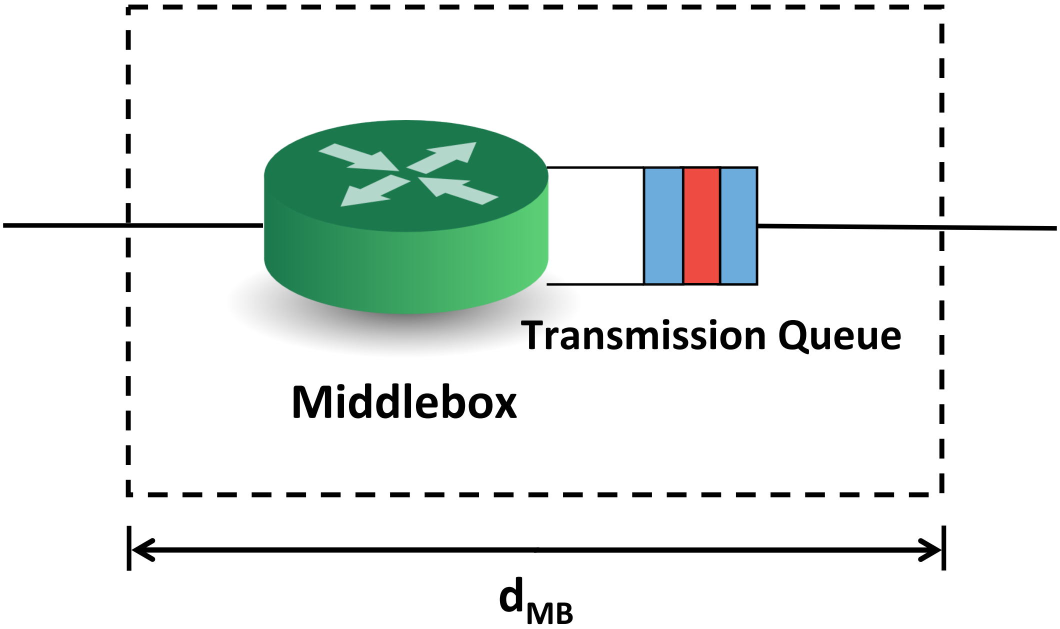


Figure 4.2: Modified definition of $d_{MB}(p)$ for detecting network compression

In that case we have two singletons:

$\tilde{P_A} = \{P_H\}$

$\tilde{P_B} = \{P_L\}$

4.2.2 Assumptions

In our analysis in this chapter we make the following assumptions:

$$d_c(P_H) \geq d_c(P_L) \tag{4.1}$$

This is generally true because it takes longer to compress random data than low entropy data in most compression methods. For simplicity in our analysis, however, we assume they are equal.

$$|P'_L| \leq |P'_H| \tag{4.2}$$

This indicates a generally accepted statement that a good compression should be able to compress very low entropy data to a length no worse than data with a relatively high entropy.

$$|P'_H| = \ell \tag{4.3}$$

This is either true or $|P'_H|$ is very close to ℓ. However, for simplicity of our analysis presented in the next section, we assume that this is true. Given that some compression algorithms add more to the data to be able to decompress, it is possible for $|P'_H|$ to be even slightly greater than ℓ in some cases. For instance, DEFLATE method that uses LZ77 requires to know the original size of the compressed data before it can decompress the data. In packet-by-packet compression, this adds a header to pass that information to the decompressor. To investigate the validity of this assumption, we experimented the size of the compressed packet size of constructed high entropy packets from `/dev/urandom` using LZ for 10000 times. Our observation was that in nearly all cases our assumption of $|P'_H| \approx \ell$ is true.

$$c = c_p \tag{4.4}$$

It is generally accepted that employing link-layer compression on non-bottleneck links is a poor network management practice, since it degrades network performance. For this reason, we expect that scenarios that compression is placed on the bottleneck are common in practice, making handling other scenarios where $c \neq c_p$ less important.

Furthermore, we assume that the network consists of a series of store-and-forward nodes; each of them is equipped with a FIFO queue and has a constant service rate. We also assume that the packet delay results from propagation delay, service time and variable queuing delay. Lastly, we assume, in the absence of compression, packets of the same size are not treated differently based on the entropy of their payload.

4.2.3 Network Compression *is* a Discriminator

Suppose that P'_H and P'_L are the packets P_H and P_L after compression. We, then, define the compression efficiency ratio, $\alpha := \frac{|P'_L|}{|P'_H|}$. Technically α is the state of the art for the compression algorithm in place. Taking into consideration the assumption 4.2: $0 < \alpha \leq 1$. For example, in our simulations presented in Section 4.3, where we used LZ compression, the size of the packets P'_L and P'_H are 56 and 1100 respectively, where $\ell = 1100$. In this case, $\alpha = \frac{56}{1100} \approx 0.05$.

$$\begin{aligned} d_{MB}(P_H) - d_{MB}(P_L) = \\ \{d_c(P_H) - d_c(P_L)\} + \{t_T(P'_H) - t_T(P'_L\}) \geq \\ (From(4.1)) \\ t_T(P'_H) - t_T(P'_L) \geq \delta_{min} > 0 \end{aligned} \tag{4.5}$$

We can apply the general framework introduced in Section 3.14 to devise a detection mechanism only if the additional delay imposed is considered as the nodal additional processing delay and not a combination of the processing and the transmission time. That is why we modified the definition of d_{MB} so that the transmission time difference between P_L and P_H is considered as the additional processing delay of the packet.

Our objective is to calculate δ_i and then show that $\forall i.\delta_i \geq \delta_{min} \geq \Theta$. Let's define P^i_H and P^i_L as the i-th packet in the transmission queue as depicted in Figure 4.3. We assume in the absence of external network variations, the position of P^i_L in the queue is no worse than P^i_H, if there were to be the i-th packet in the probing train.

Lemma 4. $t_T(P'_H) = \frac{\ell}{c}$ and $t_T(P'_L) = \frac{\ell}{c} * \alpha$.

Proof. From Equation 4.3:

$$t_T(P'_H) = \frac{|P'_H|}{c} = \frac{\ell}{c} \tag{4.6}$$

Also, from Equation 4.3:

$$t_T(P'_L) = \frac{|P'_L|}{c} = \frac{|P'_L|}{c} * \frac{|P'_H|}{|P'_H|} = \frac{\ell}{c} * \frac{|P'_L|}{|P'_H|} = \frac{\ell}{c} * \alpha \tag{4.7}$$

□

Theorem 5. $\exists K > 0. \forall i \geq K. d_{MB}(P_H^i) - d_{MB}(P_L^i) \geq \Theta$.

Proof. Let's define $T(i)$ to be truth statement $d_{MB}(P_H^i) - d_{MB}(P_L^i) \geq \Theta$. We now prove by induction that this statement is true given that it holds for some value K.

- $\exists K \geq 1$. $T(K)$ is true.
- $T(i) \Rightarrow T(i+1)$.

We first start with the inductive step: Based on the assumption that the position of P_L^i in the queue is no worse than P_H^i (if there were to be the i-th packet in the probing train), and from Equation 4.5 we have:

$$\begin{gathered} d_{MB}(P_H^i) - d_{MB}(P_L^i) = \sum_{j=1}^{i-1} t_d(P_H^j) - \sum_{j=1}^{i-1} t_d(P_L^j) \\ \text{Given all } P_H^j\text{'s have the same length, and all } P_L^j\text{'s have the same size} \\ = \sum_{j=1}^{i-1} t_T(P'_H) - \sum_{j=1}^{i-1} t_T(P'_L) = t_T(P'_H)(i-1) - t_T(P'_L)(i-1) \end{gathered} \tag{4.8}$$

Applying Lemma 4, we get,

$$d_{MB}(P_H^i) - d_{MB}(P_L^i) \geq \frac{\ell}{c}(i-1)(1-\alpha) \tag{4.9}$$

Therefore, to show $d_{MB}(P_H^i) - d_{MB}(P_L^i) \geq \Theta$ is true, it is sufficient to show that $\frac{\ell}{c}(i-1)(1-\alpha) \geq \Theta$.

If $T(i)$ is true, then: $t_T(P_H^i) - t_T(P_L^i) = \frac{\ell}{c}(i-1)(1-\alpha) \geq \Theta$

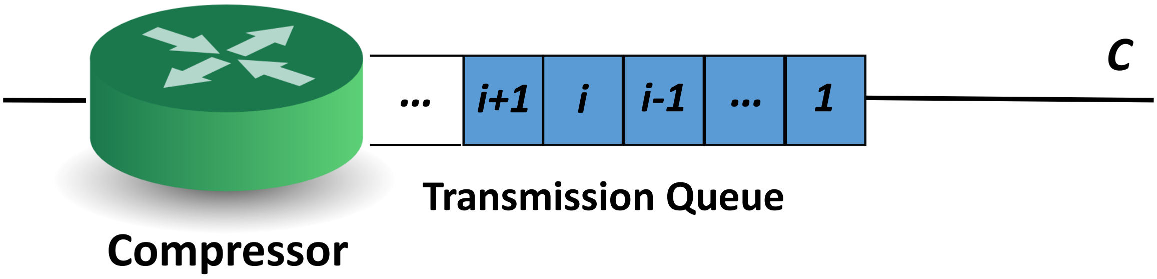


Figure 4.3: i-th packet in the queue

Since $i \geq 1$ then: $\frac{\ell}{c}(i)(1-\alpha) \geq \frac{\ell}{c}(i-1)(1-\alpha)$

And hence: $d_{MB}(P_H^{i+1}) - d_{MB}(P_L^{i+1}) = \frac{\ell}{c}(i)(1-\alpha) \geq \Theta$.

Therefore, $T(i+1)$ is also true. This proves $T(i) \Rightarrow T(i+1)$.

As for the base step of our induction: We need to find $K > 0$ in which $T(K)$ is true:

$$T(K) = \frac{\ell}{c}(K-1)(1-\alpha) \geq \Theta$$

$$K \geq K-1 \geq \frac{c \times \Theta}{\ell \times (1-\alpha)}$$

From Section 3.14, we replace Θ with $\frac{10 \times \ell}{c_p}$: $K \geq \frac{10 \times c \times \ell}{\ell \times c_p \times (1-\alpha)}$

Then from Equation 4.4 ($c = c_p$): $K \geq \frac{10}{1-\alpha}$

So what this suggest is that our $min(K)$ is a function of α. A good compression algorithm should lead to small values for α (remember that $\alpha = 0.05$ when LZ compression algorithm was applied to low and high entropy data with the size of 1100 bytes, so in this case $K \geq \lceil \frac{10}{1-0.05} \rceil = 11$). If we bound α with 0.5, then $K \geq 20$. Essentially, this suggests that we must disregard in our analysis the first $K-1$ packets in the packet train used for detection.

Given this analysis on existence of some K in which $T(K)$ holds, our induction proof

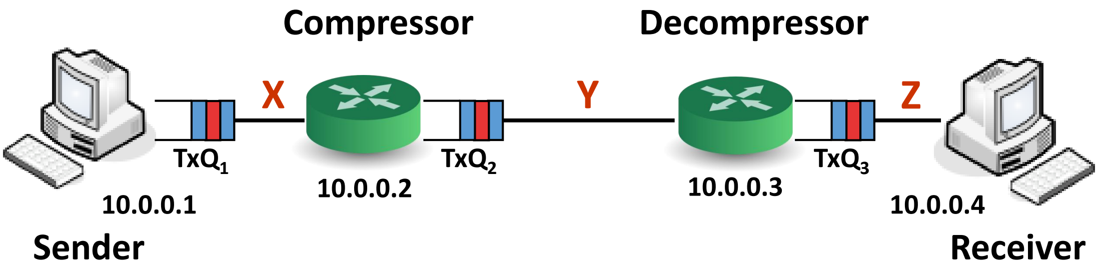


Figure 4.4: Topology used for simulations in the presence of compression link on the path

is complete. □

4.2.4 Detectability Under Different Network Characteristics

In this section we examine detectability under all possible scenarios where compression is applied to the bottleneck. As illustrated in Figure 4.4, X is the path capacity from the sender to the compression node and Z is the path capacity from the decompression node to the receiver. Y is the capacity of the compression link. The perceived bandwidths Y_{He} and Y_{Le} (as defined in Section 4.2) can be described by the link capacity, c, and α.

$$\begin{aligned} Y_{He} &= c \\ Y_{Le} &= \frac{c}{\alpha} \end{aligned} \tag{4.10}$$

Table 4.2.4, enumerates various cases when compression is applied and the expected loss rate for each case. Note that this is of importance since our approach is loss-based.

We omit scenarios that the compression link is not the bottleneck (i.e., $min(X, Z) < c$), as it is explained earlier (see Section 4.2.2). In all of the presented scenarios when we send high entropy the expected loss ratios are $TxQ_3 = 0$ and $TxQ_2 = 1 - \frac{c}{X}$, hence, $\widehat{p}_H = 1 - \frac{c}{X}$ in arithmetic in Table 4.2.4. It is only when we use low entropy that the loss ratios differ.

If the values in the most right column ($\Delta p \geq \tau = 20\%$), then the presence of compression is detectable. In examples scenarios we used for simulation (Table 4.2.4), compression in all of those scenarios were detected since the estimated loss ratios for all are greater

	All Possible	TxQ_2	TxQ_3	Expected loss rate	Expected loss rate	Total expected	Δp
	Scenarios	Overflows?	Overflow?	in TxQ_2 (p_L)	in TxQ_3 (p_L)	loss rate (p_L)	$p_H - p_L$
1	$c < \frac{c}{\alpha} < X < Z$	X	–	$1 - \frac{c}{\alpha X}$	0	$1 - \frac{c}{\alpha X}$	$\frac{c}{X}(\frac{1-\alpha}{\alpha})$
2	$c < \frac{c}{\alpha} < Z < X$	X	–	$1 - \frac{c}{\alpha X}$	0	$1 - \frac{c}{\alpha X}$	$\frac{c}{X}(\frac{1-\alpha}{\alpha})$
3	$c < X < \frac{c}{\alpha} < Z$	–	–	0	0	0	$1 - \frac{c}{X}$
4	$c < Z < \frac{c}{\alpha} < X$	X	X	$1 - \frac{c}{\alpha X}$	$1 - \frac{\alpha Z}{c}$	$1 - \frac{Z}{X}$	$\frac{Z-c}{X}$
5	$c < X < Z < \frac{c}{\alpha}$	–	–	0	0	0	$1 - \frac{c}{X}$
6	$c < Z < X < \frac{c}{\alpha}$	–	X	0	$1 - \frac{Z}{X}$	$1 - \frac{Z}{X}$	$\frac{Z-c}{X}$

Table 4.1: Expected loss rate for various cases when compression is applied

	Scenarios	(X, Y, Z)	$p_H - p_L$ (%)	$\widehat{p}_H - \widehat{p}_L$ (%)
1	$c < \frac{c}{\alpha} < X < Z$	$(30, 1, 40)$	63.3	56.5
2	$c < \frac{c}{\alpha} < Z < X$	$(40, 1, 30)$	47.5	42.4
3	$c < X < \frac{c}{\alpha} < Z$	$(10, 1, 30)$	90.0	90.1
4	$c < Z < \frac{c}{\alpha} < X$	$(30, 1, 10)$	30.0	30.1
5	$c < X < Z < \frac{c}{\alpha}$	$(5, 1, 10)$	80.0	80.2
6	$c < Z < X < \frac{c}{\alpha}$	$(10, 1, 5)$	40.0	40.1

Table 4.2: Expected loss rate difference ($p_H - p_L$) from analytical results and estimated loss rate difference ($\widehat{p}_H - \widehat{p}_L$) from simulation

than 20%.

What if compression link is not the bottleneck?

While we based our analysis on the assumption that the compression is applied to the bottleneck, in the scenario where $Z < c < X$, link-layer compression might still be detectable to some extent. This is because we expect that compression would degrade the network performance, leading to higher loss rate for the low entropy packet train compared to that of high entropy. We examined the validity of this behavior by running a single simulation scenario where $(X, c, Z) = (100, 20, 10)$, where the difference of loss ratio $\Delta\widehat{p} = -9.5$ (which is relatively low for such an extreme case where X is much higher than c and Z). And in the case of $X < c < Z$, the cooperative sender and receiver can switch roles and reverse the direction of the detection probes to make this scenario to be formulated as $Z < c < X$, which was discussed earlier. Lastly, we cannot detect compression if $c > max(X, Z)$ since compression does not affect the network performance.

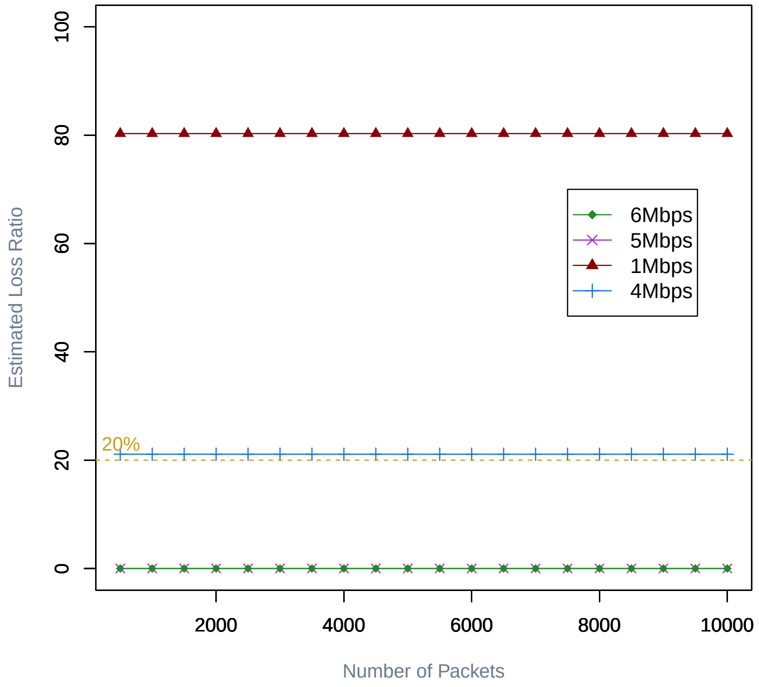


Figure 4.5: The effect of packet capacity (and number of packets) on compression detection in the simulated environment when the first and last links are 5Mbps ($\ell = 1100$).

4.3 Simulations

4.3.1 The effects of number of probes (n_P) and path capacity (c_P)

Figure 4.5 demonstrates simulation scenarios when a wide range of values number of packets are used. Also, keeping the first and last link fixed (at 5Mbps), it covers scenarios where the capacity of the compression link (the middle link) is varied.

As illustrated in Figure 4.5, the packet capacity greatly affects the detection. Specifically, when it is the bottleneck. When it is not the bottleneck or shares the same capacity with the other links (i.e., not the only bottleneck on the path), no loss attributed to the compression link is detected, hence the perceived loss ratio in those scenarios are zero. Therefore, our approach only detects, *effective* compression.

Another observation from this experiment is that, number of packets really does not affect the . Its role, however, are expected to become apparent in the case the presence of cross traffic.

4.3.2 The effects of probe size (ℓ)

Figure 4.6 illustrates the effect of the probe size on detection. An apparent observation is that the size of the packet does not affect the detection if it is at least 300 bytes long.

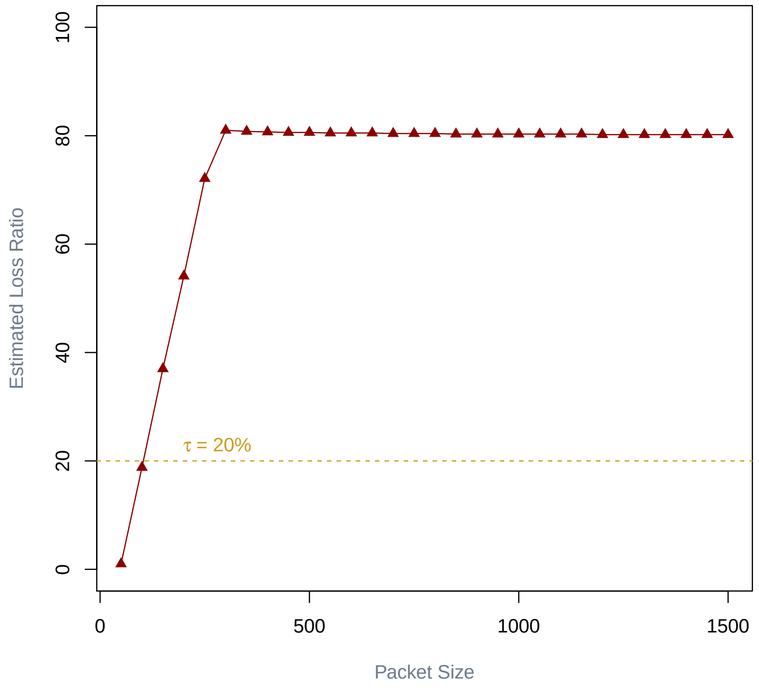


Figure 4.6: The effect of the packet size, ℓ, on detecting compression in the simulated environment $(X = 5Mbps, c = 1Mbps, Y = 5Mbps)$.

4.3.3 Estimated initial packet train length ($\widehat{n_I}$)

Table 4.3.3 shows the estimated n_I in different scenarios. It shows that consistently for the low entropy packets, more packets are needed to saturate the compression link. In scenarios (3) and (5), the low entropy packets never saturate the queue, hence by definition of the initial packet train which its purpose is to saturate the queue, n_I is undefined (i.e., $n_I = -1$). This is consistent with our results in Table 4.3.3, where we discussed that TXQ_2 never overflows in those two scenarios by the low entropy packets,

	Scenarios	(X, Y, Z)	$\widehat{n_I}$ for P_H's	$\widehat{n_I}$ for P_L's
1	$c < \frac{c}{\alpha} < X < Z$	$(30, 1, 40)$	78	191
2	$c < \frac{c}{\alpha} < Z < X$	$(40, 1, 30)$	77	137
3	$c < X < \frac{c}{\alpha} < Z$	$(10, 1, 30)$	84	-1
4	$c < Z < \frac{c}{\alpha} < X$	$(30, 1, 10)$	78	166
5	$c < X < Z < \frac{c}{\alpha}$	$(5, 1, 10)$	94	-1
6	$c < Z < X < \frac{c}{\alpha}$	$(10, 1, 5)$	84	151

Table 4.3: Estimated values of n_I for different scenarios from the simulation (-1 means the transmission queue was never saturated).

if our detection probes are the only traffic entering the compressor.

4.4 Internet Experiments

A set of three geographically distributed PlanetLab nodes connected via the open Internet were selected for our experiment (Table 7.2). We define an experiment scenario uniquely as two elements: (1) a remote PlanetLab node, and (2) whether we applied link-layer compression or not in the experiment. For every scenario, we performed 24 individual experiments, within a span of 24 hours, running only one experiment every hour. The reason behind running the experiment with this particular pattern is discussed in Chapter 2. Our results is summarized in Table by illustrating the normal distribution parameters in each scenario. The results confirm the existence of a significant gap in the perceived loss ratio difference between low and high entropy packet trains, in the the presence network compression for all scenarios tested, while in the absence of compression, both packet trains seem to be experiencing the same loss rate.

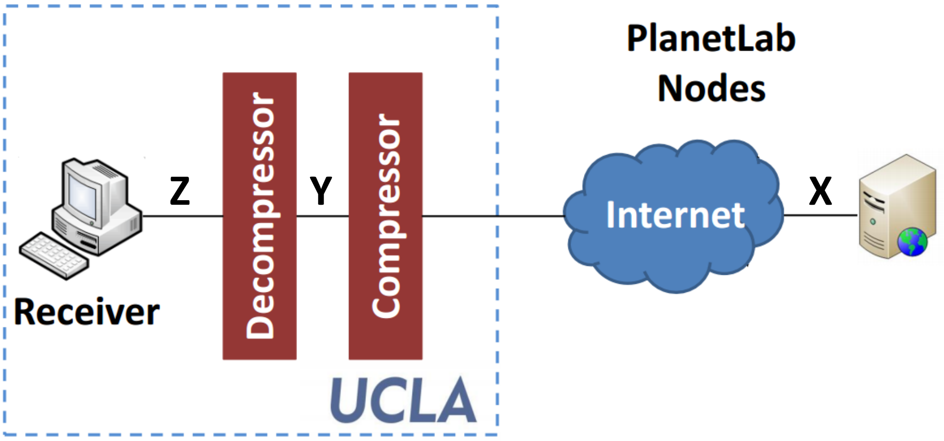


Figure 4.7: Environment used in our experiments ($Y = 1$Mbps).

				Compression		**No Compression**	
				$\widehat{p}_H - \widehat{p}_L$		$\widehat{p}_H - \widehat{p}_L$	
Location	**IP Address**	**Est RTT**	$n_I : \ (\mu, \sigma)$	(μ, σ)	*median*	(μ, σ)	*median*
California	128.111.52.63	4 ms	$(154, 11)$	$(82.1, 3.4)$	82.1	$(5.3, 4.1)$	4.9
Russia	82.179.176.42	206 ms	$(149, 16)$	$(37.2, 11.8)$	37.9	$(3.1, 9.8)$	2.5
Australia	130.194.252.8	181 ms	$(152, 9)$	$(82.3, 3.8)$	82.7	$(-1.3, 5.2)$	-0.6

Table 4.4: Summary of the results from the Internet experiments.

5

Detecting Traffic Prioritization A Loss-Based Approach

5.1 Introduction and Motivation

One of the first responses to the disadvantages of FIFO in a congested environment is strict priority queuing (SPQ), also known as just priority queuing (PQ). Strict priority queuing assumes that types of traffic can be differentiated and treated preferentially. Separate FIFO queues are created for each defined priority level and the arriving traffic is sorted into its proper queue as it arrives. Thus the first task of configuring strict priority queuing is to determine the traffic classifications. Typically between two and five levels of priority are defined (e.g., high, medium, normal, and low), although there is no theoretical limit to how many levels can be defined. More queues, however, means more complexity in running the algorithm.

At the service side of the queue, the processing rule is simple: higher priority FIFO queues are always processed to completion before lower priority queues are processed. Figure 5.1 shows how traffic randomly entering a router with two classes of defined priorities (high and low) and leaving the router according to its sorted priority. In a three-queue system, if the two highest priority queues had no packets buffered, then the lowest priority queue would be serviced. The moment a higher priority packet arrived in its FIFO queue, however, servicing of the lower priority packets would be interrupted in favour of

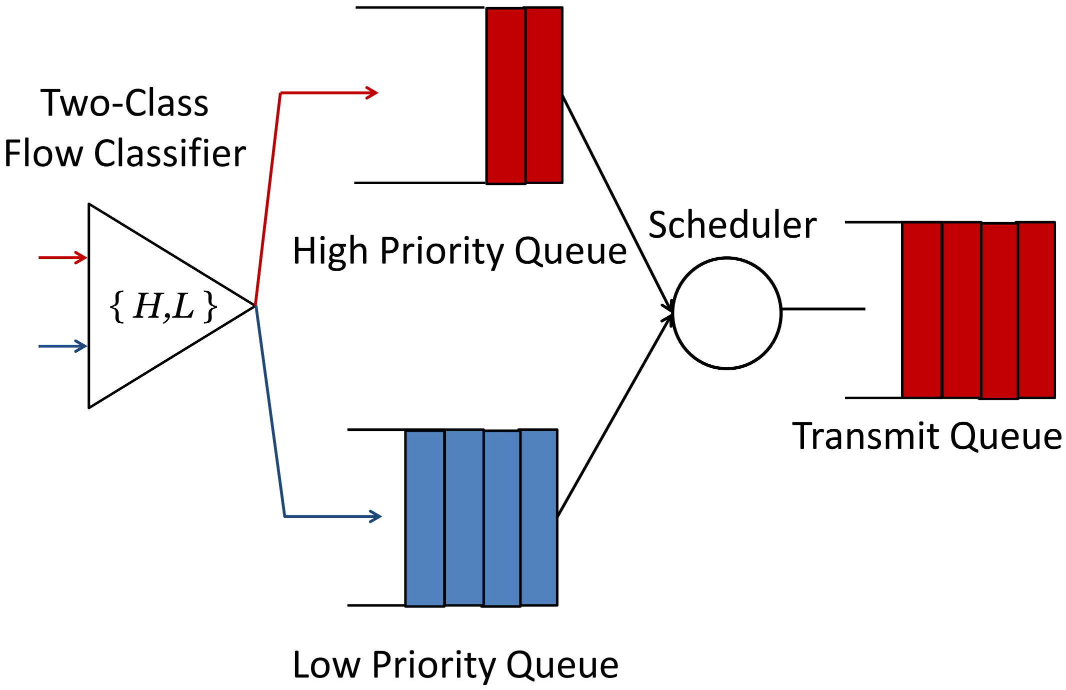


Figure 5.1: Strict Priority Queuing (with two queues high and low).

the higher priority queue. In fact, this is one of the most distinct advantages of strict priority queuing: no other queuing strategy gives high priority traffic better treatment.

The idea that all Internet traffic should be treated equally is known as network neutrality and recently there is significant interest about this concept. In other words, no matter who uploads or downloads data, or what kind of data is involved, networks should treat all of those packets in the same manner. To do otherwise, advocates argue, would amount to data discrimination [Hon08]. In addition, users are concerned about decreasing the network performance due to prioritization or deprioritization of specific applications such as P2P file sharing or “over-the-top” services like Skype with respect to others. For instance, there is already evidence that some ISPs are discriminating against BitTorrent traffic by rate limiting or blocking such flows [DMH, KD10a].

5.2 Notations

- $\mathcal{SND}$: The sender host.

- $\mathcal{RCV}$: The receiver host.

- P_H: The packet that is being prioritized.

- P_L: The packet that is being deprioritized.

- TxQ_{MB}: Transmission queue of the middlebox link.

- $t_d(p)$: Departure time (time to transmit the last bit of packet p into the wire).

- $t_a(p)$[1]: Arrival time of packet p to the middlebox.

- $d_{MB}(p) := t_d(p) - t_a(p)$, total delay imposed by the middlebox on packet p.

- t_T: Transmission time is defined as network transmission time for transmitting the entire packet p into the MB's outbound link to the destination.

- c_p: The path capacity from $\mathcal{SND}$ to $\mathcal{RCV}$.

- t_c: Time to classify the packet and place them in the designated queue.

- t_s: Time to schedule a packet (i.e., to remove the packet from the buffer and place it into the transmission queue.

- r: The sending rate of the probe packets as perceived by the middlebox. In other words, it is the minimum of the sending rate at the sender and the capacity of the narrow link between the sender and the middlebox.

5.3 Construction of $\tilde{P_A}$ and $\tilde{P_B}$

$\tilde{P_B}$ should include packets of all fixed size ℓ that are being treated with relative higher priority (e.g., is being prioritized vs. normal traffic) by the ISP. On the other hand, $\tilde{P_B}$ consists of packets with size ℓ that are treated with relative lower priority (e.g., is being de-prioritized vs. normal traffic). That is how we construct the following two singletons:

[1] $t_a(p)$ and $t_d(p)$ are based on relative clocks in a single system, hence synchronization is not required.

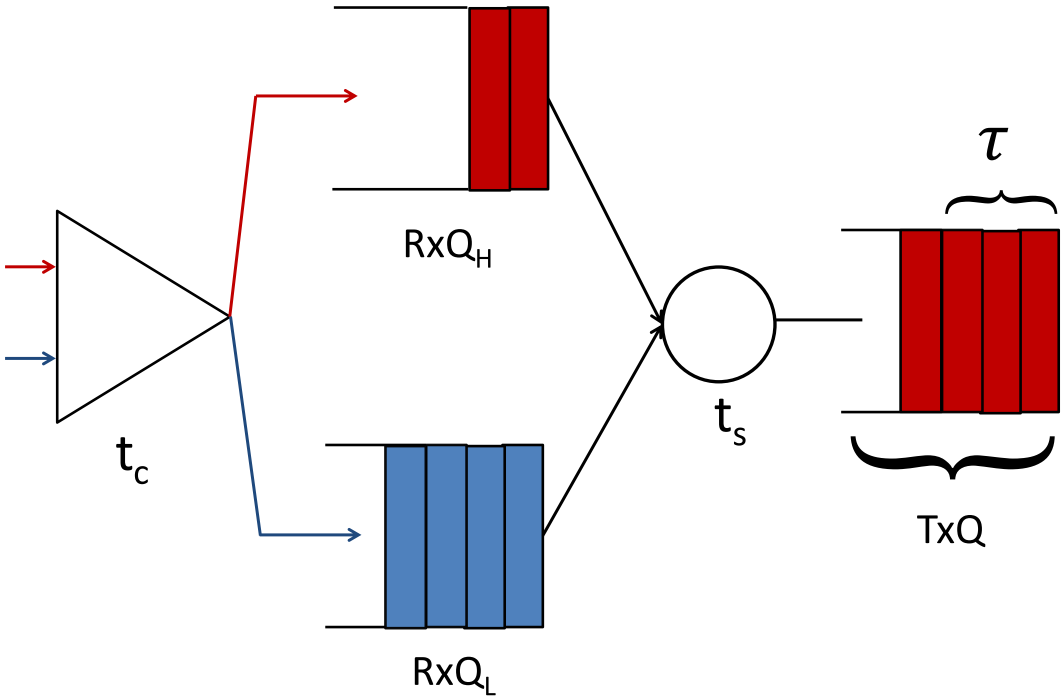


Figure 5.2: Internal details of SPQ

$\tilde{P_A} = \{P_L\}$, consists of packets with lower priority.

$\tilde{P_B} = \{P_H\}$, consists of packets with higher priority.

5.4 Detectability Under Different Network Characteristics

As illustrated in Figure 5.3, X is the path capacity from the sender to the SPQ node and Z is the path capacity from the SPQ node to the receiver. Knowing that r cannot be greater than X, Table 5.4, enumerates various cases.

We only consider the case that SPQ is on the last link to the receiver.

5.5 Assumptions

We assume that the middlebox consists of two FIFO tail-drop queues, one for high priority traffic and one with low priority traffic. This is a simplified version of more complex middleboxes that have more than two queues and use dropping mechanisms other than

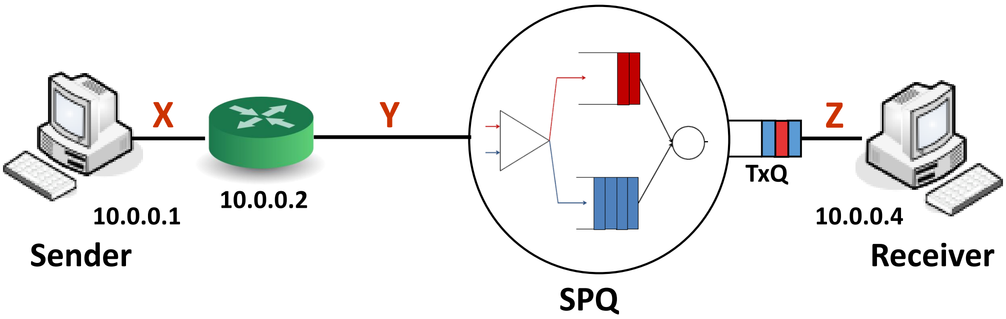


Figure 5.3: Topology used for simulations in the presence of SPQ on the path

	All Possible Scenarios
1	$Z < r \leq X$
2	$r \leq Z \leq X$
3	$r \leq X \leq Z$

Table 5.1: Different scenarios when SPQ is on the path.

tail-drop (e.g., WRED). Our analysis however can easily be extended to include those middleboxes and this assumption is not limiting the generality.

In the last section (Section 5.4), we discussed various scenarios based on the network path properties. In our analysis we only consider the following scenario:

$$c_p = Z < r \leq X \tag{5.1}$$

5.6 Traffic Prioritizer *is* a Discriminator

To show that the traffic prioritizer is a discriminator, we indeed create a scenario that in that case any SPQ that works as discussed earlier in this chapter, would behave like a discriminator such that for our definition of P_A^i and P_L^i, $\delta_{min} > \Theta$.

We define P_H^i and P_L^i as i-th packets in the high priority packet train and the low priority packet train (detection probe packets). In order to ensure that, we define the notion of packet separation train that is a set of packets with a specific priority level, placed between our detection probe packets. The packet separation train consists of N' packets of the opposite priority that is sent between each packet of the same probe. Also, the initial packet train to saturate the queue is of the high priority packet for both type of high and low priority trains. Based on the assumption that $Z = c_p$, then the initial packet train is able to saturate the high priority queue of the middlebox, hence any incoming packet either high or low would either be dropped or queued for sometime and will not immediately leave the middlebox.

Theorem 6. $\exists K > 0, \forall i \geq K, d_{MB}(P_H^i) - d_{MB}(P_L^i) \geq \Theta$.

Proof. Let's define $T(i)$ to be truth statement $d_{MB}(P_H^i) - d_{MB}(P_L^i) \geq \Theta$. We now prove by induction that this statement is true given that it holds for some value K.

- $\exists K \geq 1$. $T(K)$ is true.
- $T(i) \Rightarrow T(i+1)$.

We first start with the inductive step: We reiterate that we assume that the high priority queue is already saturated before we send the P_L^i's and P_H^i's.

We further assume that N' is set to a value such that:

$$\frac{r}{N'+1} < Z < r \leq X \tag{5.2}$$

This means that because now the high priority packets are sent at a rate of $\frac{r}{N'+1}$ which is smaller than the path capacity, then high priority packets in the detection probe cannot saturate the queue any longer. On the other hand, low priority packets theoretically cannot leave the low priority queue until all the high priority packets in the separation train are gone. And since the separation train is being sent at the rate of r, low priority packet probes will never have the chance to leave the low priority queue since the high priority queue is constantly non-empty.

Thus, $d_{MB}(P_L^i) = +\infty$ while $d_{MB}(P_H^i) \neq +\infty$ for sufficiently large values for i. Hence $d_{MB}(P_L^i) - d_{MB}(P_H^i) \geq \Theta$.

In this proof, $T(K)$ where our base step works is equal to the size of the low priority queue. Since the queue size of routers is not unbounded, then such value for K must exist.

□

5.7 Internet Experiments

A set of three geographically distributed PlanetLab nodes connected via the open Internet were selected for our experiment (Table 5.2). We define an experiment scenario uniquely as two elements: (1) a remote PlanetLab node, and (2) whether we applied SPQ or not in the experiment. For every scenario, we performed 24 individual experiments, within a span of 24 hours, running only one experiment every hour. The reason behind running the experiment with this particular pattern is discussed in Chapter 2. Our results is summarized in Table by illustrating the normal distribution parameters in each scenario. The results confirm the existence of a significant gap in the loss ratio difference between low and high priority packet trains, in the the presence SPQ, while in the absence of strict priority queueing, both packet trains seem to be experiencing the same loss rate.

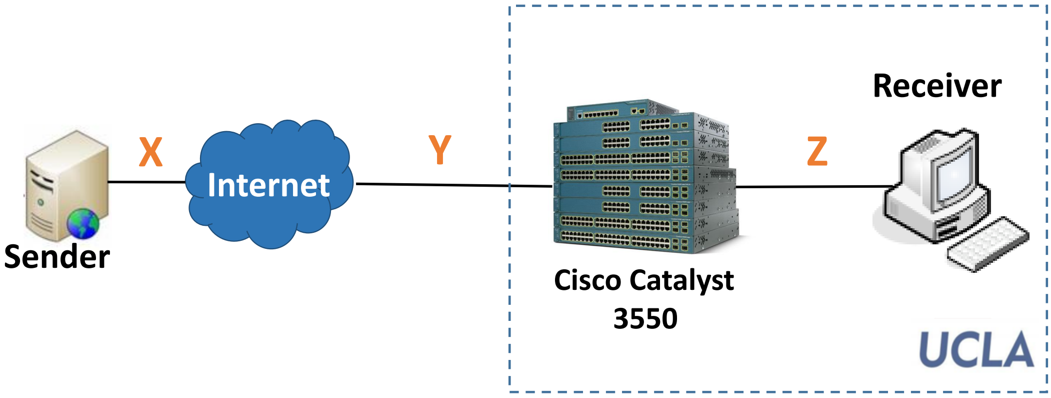


Figure 5.4: Environment used in our Internet experiment ($Z = 10$Mbps).

				SPQ		No SPQ	
				$\widehat{p}_H - \widehat{p}_L$		$\widehat{p}_H - \widehat{p}_L$	
Location	**IP Address**	**Est RTT**	$n_I:\ (\mu, \sigma)$	(μ, σ)	*median*	(μ, σ)	*median*
California	128.111.52.63	4 ms	$(154, 11)$	$(82.1, 3.4)$	82.1	$(5.3, 4.1)$	4.9
Russia	82.179.176.42	206 ms	$(149, 16)$	$(37.2, 11.8)$	37.9	$(3.1, 9.8)$	2.5
Australia	130.194.252.8	181 ms	$(152, 9)$	$(82.3, 3.8)$	82.7	$(-1.3, 5.2)$	-0.6

Table 5.2: Summary of the results from the Internet experiments.

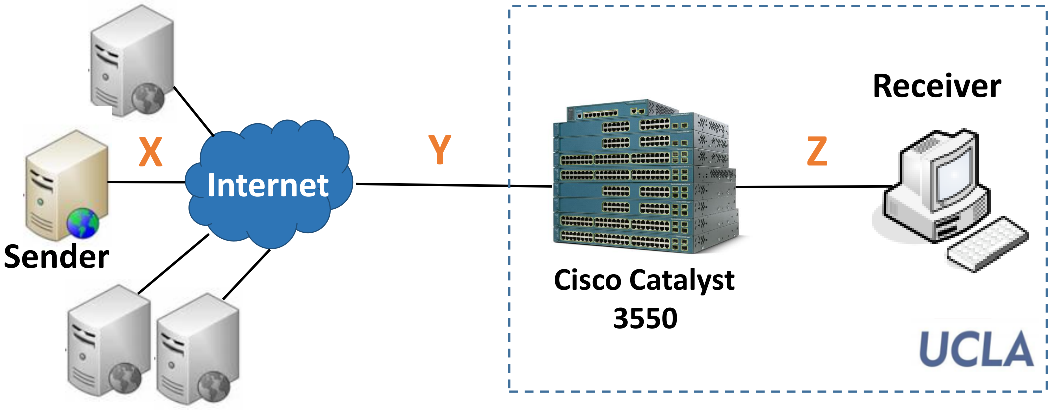


Figure 5.5: Environment used in our Internet experiment with the help of contributors to throttle the last link ($Z = 100$Mbps).

6 Detecting Traffic Shaping and Policing: A Loss-Based Approach

6.1 Introduction and Motivation

In this chapter we will present a loss-based approach to detect traffic shaping and policing. We will achieve this by formulating the detection of traffic shaping and policing as a discriminator Our proposed approach detects traffic differentiation between any two traffic flows that is to be compared. That is we do not claim to detect two types of traffic that are both being shaped or they are both not being shaped by the shaper. We only can detect when one flow is being shaped while the other is not.

The idea that all Internet traffic should be treated equally is known as network neutrality and recently there is significant interest about this concept. In other words, no matter who uploads or downloads data, or what kind of data is involved, networks should treat all of those packets in the same manner [TMF09]. One of the responses to the question of why would a residential ISP deploy traffic shaping is relevant to the network neutrality debate. An ISP want to limit the service rate provided to the aggregate traffic produced or consumed by a customer, or to limit the service rate consumed by a certain application (e.g. BitTorrent). Furthermore, an ISP want to allow a user to exceed the service rate that he/she has paid for, for a limited burst size. In that case the user pays for ρ bps, with the additional service capacity $C - \rho$ marketed as a free service enhancement. This is, for instance, how Comcast advertises their PowerBoost traffic shaping mechanism. Moreover, certain ISPs prefer to describe their service rates as upper bounds for what the user will actually get, e.g., a downstream rate of at most 6Mbps. In that case, a shaper can be used to enforce the upper bound of the service rate[KD10a].

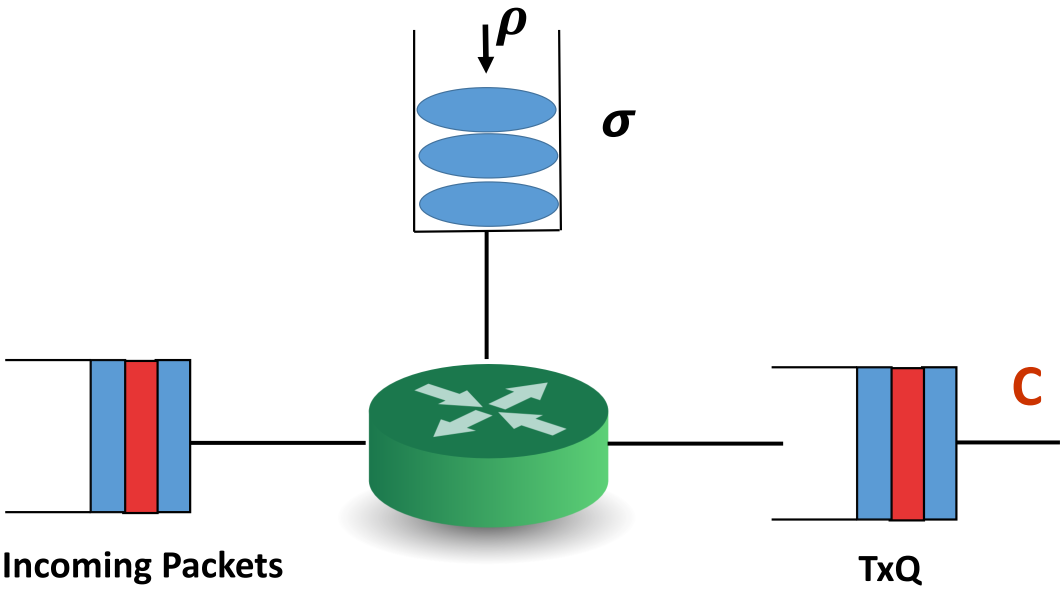


Figure 6.1: How traffic shaping works (token leaky bucket algorithm)

The increasing penetration of broadband access technologies, such as DSL, DOCSIS and WiMAX, provides users with a wide range of upstream and downstream service rates. Broadband users need to know whether they actually get the service rates they pay for. On the other hand, ISPs now have an extensive toolbox of traffic management mechanisms they can apply to their customers' traffic: application classifiers, schedulers, active queue managers etc.

Traffic shaping is a way to help optimize performance and improve latency by controlling the amount of data that flows into and out of the network. It will increase usable bandwidth for some kinds of packets by delaying some other kinds of packets. Many access ISPs are deploying traffic shaping to limit the peak network traffic on their transit links, and thereby reduce their wide-area bandwidth costs.

In more details, a traffic shaper is a single-input single-output packet forwarding module that behaves as follows: Consider a link of capacity C bps, associated with a "token bucket" of size σ tokens. Whenever the bucket is not full, tokens are generated at a rate ρ tokens per second, with $\rho < C$. The link can transmit an arriving packet of size L bits only if the token bucket has at least L tokens - upon the transmission of the packet, the shaper consumes L tokens from the bucket.

Hence, if we start with a full token bucket of size σ tokens, and with a large burst of packets of size L bits each (suppose that σ is an integer multiple of L for simplicity), the link will be able to transmit k of those packets at the rate of the capacity C, with $k = \frac{\frac{\sigma}{L}}{1-\frac{\rho}{C}}$.

6.2 Objective

The difference between a traffic shaper and a traffic policer is that the former has a buffer to hold packets that arrive when the token bucket is empty, and a policer simply drops such "non-conforming" packets. In other words, a shaper[1] delays packets that exceed the traffic shaping profile (σ, ρ), while a policer drops them. Policers can cause excessive packet losses and so shapers are more common in practice. This suggests that both shaper and policer behave in exactly the same way when the receive queue of the shaper is full, hence incoming packets will be dropped. In our detection mechanism we will focus only on this scenario, hence our approach can be applied to detecting both types of middleboxes aforementioned.

Recall from Chapter 3, in order to show that a particular middlebox, MB, that influences network flows by imposing discriminatory delay, is normally detectable, it is sufficient to show:

1. Middlebox, MB, is a network discriminator. In other words, we can construct (hence, there exist) a pair of sets $(\tilde{P_A},\tilde{P_B})$ that is delay discriminatory.

[1] From now on in this chapter when we mention the shaper we refer to the middlebox that does either traffic shaping or traffic policing.

2. $\delta_{min} \geq \Theta$.

In the following sections in this chapter we attempt to achieve this.

6.3 Notations

- $t_d(p)$: Departure time (time to transmit the last bit of packet p into the wire).
- $t_a(p)$[2]: Arrival time of packet p to the middlebox.
- $d_{MB}(p) := t_d(p) - t_a(p)$, total delay imposed by the middlebox on packet p.
- t_T: Transmission time is defined as network transmission time for transmitting the entire packet p into the MB's outbound link to the destination.
- c: Link capacity of the link immediately after the shaper on the path to the receiver (i.e. the peak rate).
- σ: Size of the bucket (i.e. maximum number of allowed tokens to be accumulated in the bucket).
- ρ: The shaped rate (i.e. the token generation rate in token bucket algorithm).
- P_ρ: The packet, with size ℓ, that is being shaped by the shaper.
- P_c: The packet, with size ℓ, that is not being shaped by the shaper.
- TxQ_c: Transmission queue of the shaper's outbound link.
- RxQ_ρ: Receive queue (buffer) in the shaper for holding the non-conforming traffic.
- r: The sending rate of the probe packets as perceived by the middlebox. In other words, it is the minimum of the sending rate at the sender and the capacity of the narrow link between the sender and the middlebox.

[2] $t_a(p)$ and $t_d(p)$ are based on relative clocks.

6.4 Construction of $\tilde{P_A}$ and $\tilde{P_B}$

$\tilde{P_A}$ should include packets of all fixed size ℓ that are being shaped by the shaper at the rate of ρ, whereas $\tilde{P_B}$ consists of packets with size ℓ that are not shaped and are sent at the peak rate, c. That is how we construct the following two singletons:
$\tilde{P_A} = \{P_\rho\}$
$\tilde{P_B} = \{P_c\}$

In practice we should construct a stream of packets whereas sending identical packets. One reason is to ensure that our probe packets are not being influenced by RE-enabled (redundancy elimination) routers. However, for simplicity of our theoretical analysis, we assume all P_{ρ_i}'s be the same, and all P_{c_i}'s are identical.

6.5 Traffic Shaper *is* a Discriminator

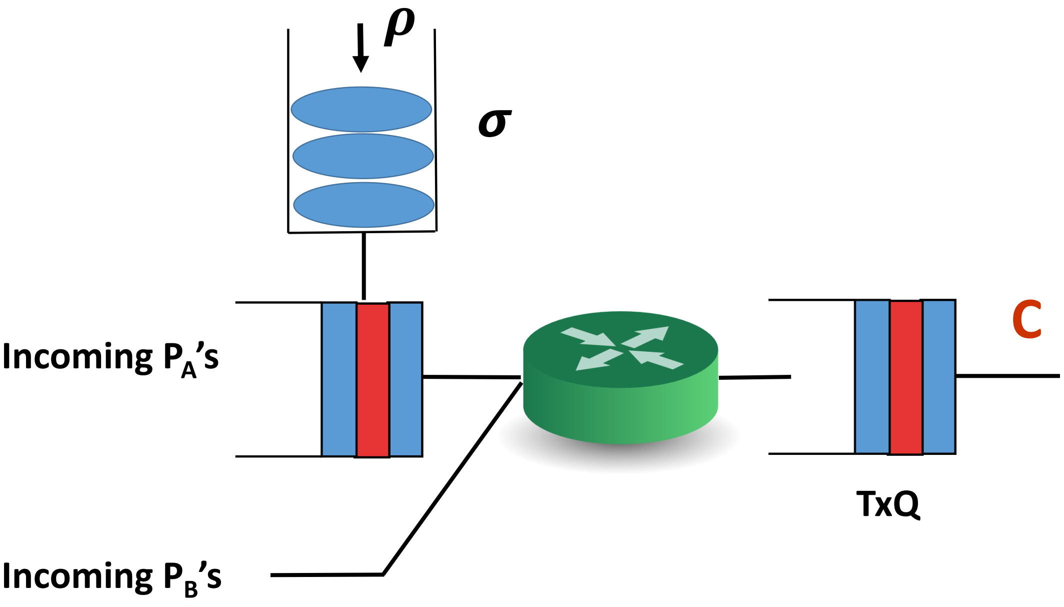


Figure 6.2: Traffic shaping effect on shaped and non-shaped flows.

Based on the values of r, c, and ρ, and given the fact that $\rho < c$, there are three possible scenarios:

6.6 $r < \rho < c$

This case is not interesting to us and we do not claim to detect it since our traffic is not going to be shaped (by the shaper in question) anyway, since the sending rate is less than the shaping rate. By our definition, the shaper is not interfering with the traffic flow, hence no reason to detect.

6.7 $\rho < c < r$

In this case the end-host is sending data more than the capacity of the link, c. However, to simplify our analysis by considering only one scenario, we reduce r so that $\rho < r \leq c$ (note that the endhost is capable of reducing its sending rate). To do so, the sender can estimate the link capacity using the approach discussed in Appendix B. Then, set $r = c$. Self-limiting the sending rate can be achieved by using the Linux `tc` command (LARTC). Note that, the sender also must increase its transmission buffer so that the probe packets are not dropped at the sender.

6.8 $\rho < r \leq c$

In this case, in the absence of cross traffic, TxQ_c will never queue any packets. P_ρ packets arrive to TxQ_c at the $\rho < c$ rate. P_c packets, in this case, arrive at the $r \leq c$ rate and also in this case no packets will be queued in TxQ_c

To show that traffic shaper is a discriminator, Now we want to calculate δ_i and then show that $\forall i.\delta_i \geq \delta_{min} \geq \Theta$. Let's define P_ρ^i and P_c^i as the i-th packet sent in the packet train in two different phases, meaning we send two separate packet trains, one consisting of only P_ρ^i's and the other consisting of P_c^i's. Now we want to calculate δ_i:

$$\delta_i = d_{MB}(P_\rho^i) - d_{MB}(P_c^i) \tag{6.1}$$

In the case of P_c^i's, $d_{MB}(P_c^i) = 0$ since they are not shaped and will immediately be transmitted since $r < c$. Hence, all we need to calculate is $d_{MB}(P_\rho^i)$.

$$\delta_i = d_{MB}(P_\rho^i) \tag{6.2}$$

In the case of P_ρ^i's, we, first, send as many packets needed to saturate RxQ_ρ. Therefore, any subsequent packets in this packet train will dropped until one slot is open in RxQ_ρ. Now we try to calculate $d_{MB}(P_\rho^i$ only after when RxQ_ρ is full. Let's say it takes K packets for RxQ_ρ to overflow [3]. In that case, we disregard all P_c^i's and P_ρ^i's for $i < K$ and only attempt to calculate the following:

$$\forall i \geq K. \delta_i = d_{MB}(P_\rho^i) - d_{MB}(P_c^i) = d_{MB}(P_\rho^i) \tag{6.3}$$

Now, for better layout our argument, we consider a specific example where $\dfrac{\rho}{r} = \dfrac{1}{2}$. In this scenario, the loss pattern depicted in Figure 6.3 will be observed by the receiver.

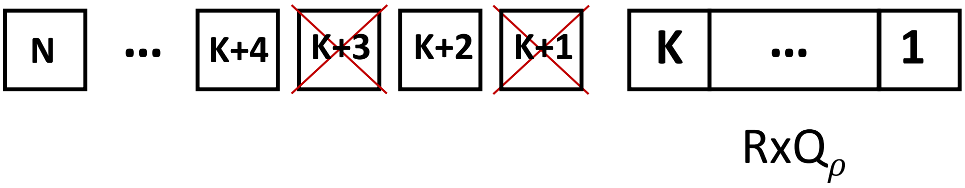


Figure 6.3: Loss pattern after RxQ_ρ overflows with $\dfrac{\rho}{r} = \dfrac{1}{2}$

Therefore, δ_i is either of the two values below:

- $+\infty$: which means the packet is dropped and never received by the receiver.
- $|RxQ_\rho| * \dfrac{l}{\rho}$: which is the time to transmit all packets in the queue including P_ρ^i.

[3] We can estimate K by sending many packets and observe the loss trend at the receiver. Packets with index $i < K$ should all be received, while the subsequent packets are dropped with a noticeable and rather consistent pattern

Which leads to $\delta_{min} = |RxQ_\rho| * \frac{l}{\rho}$ since $|RxQ_\rho|$ is finite. This value for δ_{min} is undesirable for two reasons: (1) its value is dependant on a factor that is not easily measurable by the end-hosts ($|RxQ_\rho|$), and (2) if $|RxQ_\rho|$ could be any arbitrary value set by the administrator and if it is set to a small value, δ_{min} will be small and hence might not be detectable. Remember that ℓ cannot exceed MTU, so the end-hosts can only increase the value of δ_{min} to a certain value. For this, it is desirable to perhaps rephrase our analysis, so that δ_i's are not dependant on the size of the receive queue.

To cope with this, we pair up consequent packets (i and $i+1$) and consider each pair as one packet j in our analysis (Figure 6.4).

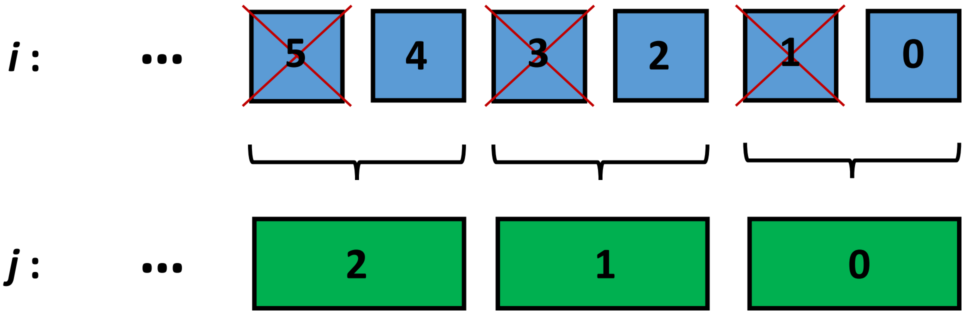


Figure 6.4: Grouping of packets with $\frac{\rho}{r} = \frac{1}{2}$

That is,

$$\forall i \geq K . P_\rho^j = P_\rho^i \oplus P_\rho^{i+1} \tag{6.4}$$

And consequently we define d_{MB} to be:

$$d_{MB}(P_\rho^j) = t_d(P_\rho^{i+1}) - t_a(P_\rho^i) \tag{6.5}$$

And lastly we calculate δ_j:

$$\delta_j = d_{MB}(P_\rho^j) - d_{MB}(P_c^j) = +\infty - 0 = +\infty \geq \Theta \tag{6.6}$$

which is true for all j's, and hence, $\delta_{min} \geq \Theta$ which we can conclude it is detectable when $\frac{\rho}{r} = \frac{1}{2}$.

So what happens if $\frac{\rho}{r} < \frac{1}{2}$? In this case, we can still pair up consequent packets the same way we did in the previous case. All the analysis will still hold because we always have at least one packet in each pair that will be dropped which makes $\delta_j = +\infty$.

So how about the case $\frac{\rho}{r} > \frac{1}{2}$? In this scenario, we can always increase the size of our group to ensure that there will always be at least one packet in each group that will be dropped. For example, $\frac{\rho}{r} = \frac{99}{100}$, then the group size should be 100. However, increasing the group size this impose a significant increase in the probing packets required for detection (as it is generically defined by how to detect Vahab's middleboxes). In the case of $\frac{1}{2}$, the number of probe packet train required for detection is multiplied by two to what is already needed by the general detection approach. Similarly, in the case of $\frac{99}{100}$, the number of packets required for detection is multiplied by 100! That clearly introduces a significant overheard and level of intrusiveness of our approach. However, normally shaping is applied to significant drop rates, and more commonly $\frac{\rho}{r} = \frac{\rho}{c} \leq \frac{1}{2}$. Finding a higher upper-bound than $\frac{1}{2}$ requires more careful investigation of intrusiveness as well as self-congestion that may lead to misdetections. We leave improving this upper-bound to future work.

6.9 Internet Experiments

For our Internet experiments, we used a set of three geographically distributed PlanetLab nodes connected via the open Internet (Table 5.2). For every scenario in our experiment, we performed 24 individual experiments, within a span of 24 hours, running only one experiment every hour. The reason behind running the experiment with this particular pattern is discussed in Chapter 2. Our results is summarized in Table by illustrating the normal distribution parameters in each scenario. The results confirm the existence of a significant gap in the loss ratio difference between the shaped and non-shaped packet trains, in the the presence the shaper, while in the absence of traffic shaping, both packet trains seem to be experiencing the same loss rate.

				Shaping		**No Shaping**	
				$\widehat{p}_H - \widehat{p}_L$		$\widehat{p}_L - \widehat{p}_H$	
Location	**IP Address**	**Est RTT**	$n_I :\ (\mu, \sigma)$	(μ, σ)	*median*	(μ, σ)	*median*
California	128.111.52.63	4 ms	$(87, 14)$	$(47.2, 13.4)$	48.1	$(2.3, 7.2)$	3.6
Russia	82.179.176.42	206 ms	$(92, 22)$	$(53.2, 2.8)$	56.9	$(-5.3, 2.8)$	-2.5
Australia	130.194.252.8	181 ms	$(111, 12)$	$(49.4, 8.9)$	47.4	$(3.5, 7.2)$	1.6

Table 6.1: Summary of the results from the Internet experiments.

7

Detecting Network Compression: A Delay-Based Approach

In this chapter we present a delay-based probing technique to detect the compression of traffic flows by intermediaries. Our proposed technique only uses packet inter-arrival times for detection. We use only the relative delays between arrival times of our probing packets for detection, so clock synchronization is unnecessary.

We propose two end-to-end approaches based on a receiver's cooperativeness in the detection process. A cooperative receiver is willing to make necessary changes on its machine or system to fully cooperate with the sender in the detection process. A responsive receiver, on the other hand, only responds to the sender's requests as long as they do not require any changes on the receiver's machine. For example, our approach assumes that the receiver responds to the sender's ICMP requests. Generally speaking, an approach that works in an uncooperative environment makes its deployment and use more practical.

Similar to our proposed loss-based approaches discussed in the previous chapters, our proposed solution here also uses only regular unicast probes, and thus it is applicable in today's Internet.

7.1 Assumptions

In addition to assumptions presented for detecting network discriminators (Section 3.4), we assume that the packet delay results from propagation delay, service time and variable queuing delay. Furthermore, our approach presented in this chapter is based on the assumption that, in the absence of compression, packets of the same size are not treated differently based on the entropy of their payload.

7.2 End-to-End Bandwidth Estimation Techniques

Because of the nature of network compression effects on the available bandwidth and the fact that our approach is inspired by the algorithms used to estimate bandwidth, in this section we present end-to-end techniques and tools for measurements of the available bandwidth and capacity of a network path. The problem of bandwidth estimation has been extensively studied in the past. Many approaches are designed for cooperative end-hosts, and some are designed to work with responsive hosts.

End-to-end active probing schemes for bandwidth estimation are classified into three categories [PDM03]: Packet Pair/Train Dispersion (PPTD), Self-Loading Periodic Streams (SLoPS), and Trains of Packet Pairs (TOPP). In this section we briefly describe each of these techniques.

7.2.1 Packet Pair/Train Dispersion (PPTD)

The packet-pair technique was first introduced by Keshav [Kes95]. In this technique, the source sends multiple pairs of packets to the receiver. Each packet pair consists of two packets of the same size sent back-to-back. Then, the dispersion of a packet pair is used to measure the capacity of the path. The dispersion of a packet pair, δ_i, at a particular link of the path, is defined as the time distance between the last bit of each packet. With the assumption of no cross traffic, δ_i is:

$$\delta_i = \max\left(\delta_{i^*}, \frac{L}{C_i}\right) \tag{7.1}$$

where δ_{i^*} is the the dispersion prior to the link C_i, L is the packet size, and $\delta_0 = L/C_0$.

Measuring the dispersion at the receiver, δ_R, is what is used to estimate the path capacity, C (H is the number of hops between the end-hosts):

$$\delta_R = \max_{0 \le i \le H}\left(\frac{L}{C_i}\right) = \frac{L}{\min_{0 \le i \le H}(C_i)} = \frac{L}{C} \tag{7.2}$$

$$C = \frac{L}{\delta_R}. \tag{7.3}$$

Jain et al. [JR86] extended the packet-pair probing technique to packet trains, where more than two packets are sent back-to-back. The dispersion of a packet train at a link is defined as the time between the last bits of the first and last packets in the train.

PPTD probing techniques typically require cooperative end-hosts. It is, however, possible to perform PPTD measurements with only a responsive receiver. In that case, the receiver is expected to, for instance, respond to ICMP messages. However, the reverse path capacities and cross traffic may affect the results.

7.2.2 Self-Loading Periodic Streams (SLoPS)

SLoPS is a another methodology for measuring end-to-end available bandwidth. In this technique, the sender periodically sends a number of equal-sized packets to the receiver at a certain rate. This measurement methodology involves monitoring the arrival time variations of the probing packets. If the sending rate is greater than the path's available bandwidth, it will overload the queue of the bottleneck, which results in an increasing trend of one-way delay.

On the other hand, if the stream rate is lower than the available bandwidth, the probing packets will go through the path without overloading the queue: thus we do not expect to see an increasing trend. In this approach, the sender, through iterations, attempts to adjust the sending rate to get close to the available bandwidth.

7.2.3 Trains of Packet Pairs (TOPP)

Unlike the Self-Loading Periodic Streams technique that measures the end-to-end one-way delays of a packet train arriving at the receiver, TOPP [MBG00] increases the sending rate (R_S) until a point where the sender is sending faster than the path capacity (C). The receiver cannot receive faster than the available bandwidth at the bottleneck ($R_R < R_S$), so further increasing the sending rate above the available capacity means that the packets will get queued at the intermediate routers. As long as the sender is still sending within the path capacity, the receiving rate is not more than the available capacity. Thus, the ratio of sending rate to receiving rate is close to unity (i.e., $R_R = R_S$). Hence, TOPP estimates the available bandwidth to be the maximum sending rate such that $R_S \approx R_R$. The following equation is used to estimate the capacity C from the slope of R_S/R_R versus R_S:

$$\frac{R_S}{R_R} = \frac{R_S}{R_S + R_C} C \tag{7.4}$$

where R_C is the average cross traffic rate. TOPP is quite similar to SLoPS. In fact, most of the differences between the two methods are related to the statistical processing of the measurements.

Table 7.1 summarizes some of the publicly available bandwidth estimation tools and the methodology used in their underlying estimation algorithm.

7.3 Approach Overview

To detect if compression is provided on the network we exploit the unique effects of compression on network flows. Assuming the original packets were of the same size, compressed low entropy data packets are expected to be considerably smaller than compressed

Table 7.1: End-to-end bandwidth estimation tools

Tool	**Measurement Metric**	**Methodology**
bing	Path capacity	PPTD
bprobe	Path capacity	PPTD
nettimer	Path capacity	PPTD
pathrate	Path capacity	PPTD
sprobe	Path capacity	TOPP
cprobe	Available bandwdith	PPTD
pipechar	Available bandwdith	PPTD
pathload	Available bandwdith	SLoPS
IGI	Available bandwdith	SLoPS
pathchirp	Available bandwdith	SLoPS

packets containing high entropy data, which in turn leads to a shorter transmission delay. The added processing delay (d_C) caused by compression/decompression methods for low entropy packets is not greater than the high entropy packets ($d_{C_L} \leq d_{C_H}$ where L: low entropy, H: high entropy) [Sal04]. Based on these facts, the sketch of our approach is as follows:

Send a train of fixed-size packets back-to-back with payloads consisting of only low entropy data. Then send a similar train of packets, except these payloads contain high entropy data instead. We then measure the arrival times of the first and the last packet in the train, independently for low entropy (t_{L_1} and t_{L_N}, where N is the number of packets in a single train) and high entropy (t_{H_1} and t_{H_N}) packet trains. Since the number of packets in the two trains is known and all of the packets have the same uncompressed size, the following inequality will hold if some kind of a network compression is performed on the path:

$$\Delta t_L = t_{L_N} - t_{L_1} < \Delta t_H = t_{H_N} - t_{H_1} \tag{7.5}$$

The inequality 7.5 suggests that the total set of highly compressible low entropy packets gets to the destination faster than the set of less compressible high entropy packets. Conversely, if the packets are not being compressed by any intermediary, then the two sides of the inequality 7.5 should be almost equal. This suggests that a threshold should be specified to distinguish effects of compression from normal Internet variabilities:

$$\Delta t_H - \Delta t_L > \tau \tag{7.6}$$

The underlying rationale behind this approach is that because of the presence of compression and decompression, the receiving party should sense a relatively higher bandwidth when the train of low entropy data is sent, since the same amount of data is received, but in shorter time.

We used UDP packets to generate our probe train. When the receiver is cooperative, with the help of two TCP connections before and after the probe UDP train, the sender is able to send experiment parameters to the receiver, and the receiver uses the second TCP connection to send the recorded arrival times of the received packets to the sender for further analysis.

If the receiver does not cooperate, but is responsive, we attached an ICMP ping request packet to the head and tail of the UDP probing train. In this way, the sender performs the detection by analyzing the difference between the arrival time of the two ICMP ping reply packets, without relying on the receiver to provide any measurement information.

Our techniques are not designed to handle receivers who are neither cooperative nor responsive.

7.4 Detection Approach Parameters

7.4.1 Content of packet's payload

The payloads of the low entropy packets are filled with 0's. The payloads of the high entropy packets are filled with random bytes read from `/dev/urandom`, independently for each new experiment.

7.4.2 Packet size

Dovrolis et al. [DRM01b] argued that a maximum transmission unit (MTU) is not optimal for accurate bandwidth estimation. However, we used large packet size probes (1100 bytes) since the larger the packets, the more apparent are the effects of compression, which in turn leads to a more accurate detection. We emphasize that while our technique is based on measuring the bandwidth, for detecting compression we do not need to estimate the bandwidth *accurately*.

7.4.3 Inter-packet departure spacing

In our experiments we use an empirically set value of 100 μ-sec. In general, this number should not be too small to ensure that our experiment does not result in queuing overflows in the intermediate routers. Conversely, this value should not be too large, since sending the packets at a slow rate removes the aggregate effects of compression on the traffic flow.

7.4.4 Number of measurements

The presence of cross traffic, on average, differs at certain times of the day and follows a particular pattern [SNC04]. By performing measurements throughout the day, we hope to capture the effects of time-dependent cross traffic variation. We ran each scenario at every hour throughout an entire day, resulting in a total of 24 measurements.

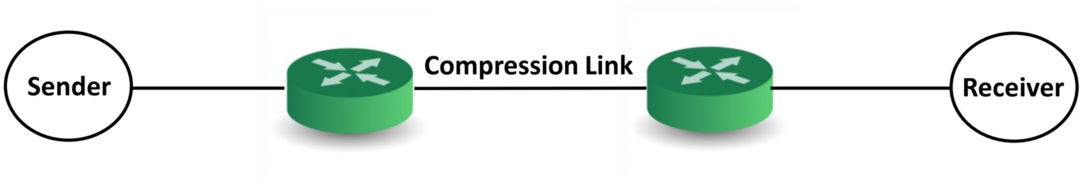


Figure 7.1: Topology used for simulation

7.4.5 Threshold

The selection of this value highly depends on the time precision and resolution of the machine performing the measurement time analysis. If the time precision on a typical machine with a typical operating system is 10 ms [Pax04], then we believe any value at least an order of magnitude higher (e.g. 100 ms) is a suitable selection for this parameter.

7.4.6 Number of packets

A careful selection of this number is vital, since a small train may not be sufficient to introduce a noticeable gap of aggregate compression, and on the other hand, a large number of packets would make our approach highly intrusive. In the next section, we show, through simulations, that this number is positively correlated with the available bandwidth. In our experiments we used 6000 packets for each train. Section 7.5.2 shows that this number is sufficient without being intrusive (Section 7.7.1).

7.5 Simulations

We simulated our approach using the `ns-3` simulator [ns3] under different network scenarios using the simple topology depicted in Fig. 7.1.

We used simulation to investigate how network characteristics can influence our detection rate. Each of the following subsections refer to a particular scenario. Within each subsection, we first describe the scenario setup, then finally present the obtained results.

7.5.1 Scenario I

In the first scenario we tested our approach using the topology in Fig. 7.1 when the capacity of all three links are equal (Fig. 7.2). As these results show, we can detect compression on a 100 Kbps or 5 Mbps link with a 100 msec threshold and a small to moderate number of packets.

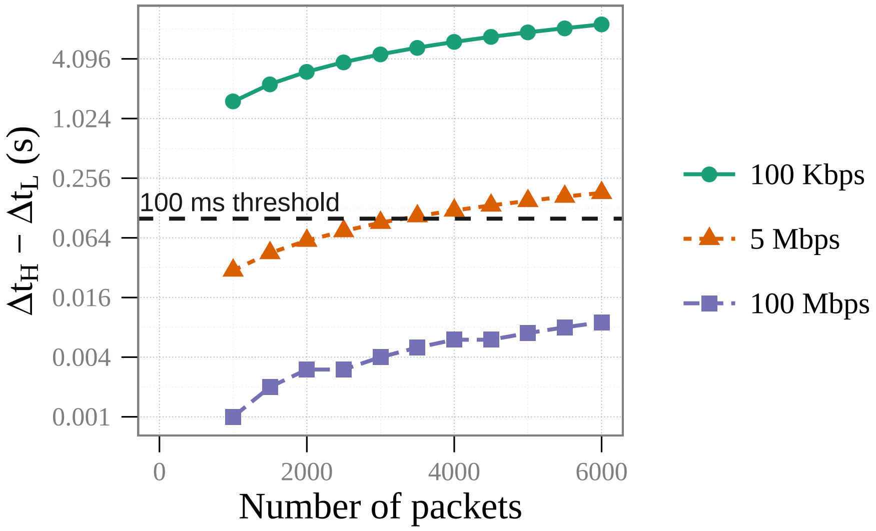


Figure 7.2: Comparison of number of probe packets required for detection for different bandwidths (in log-scale).

While the results verify our proposed approach, they show that in our detection mechanism, as the path capacity gets higher, more probe packets are required for the detection of compression effects. With 100 Mbps links, we cannot detect at a threshold of 100 msec even with 6000 packets. The trend of the curve suggests we would need several thousand more packets. This observation suggests that both the path capacity and the available bandwidth of the path in question have direct effects on the number of packets required for each measurement. It also suggests that a relative orders-of-magnitude larger number of probe packets is required for detection in high speed networks (e.g., 1Gbps). However, this is not a major disadvantage for our approach, since compression on high-speed networks is rarely deployed because the hardware required for compression for high-speed networks is expensive. In addition, the typical hardware compression components

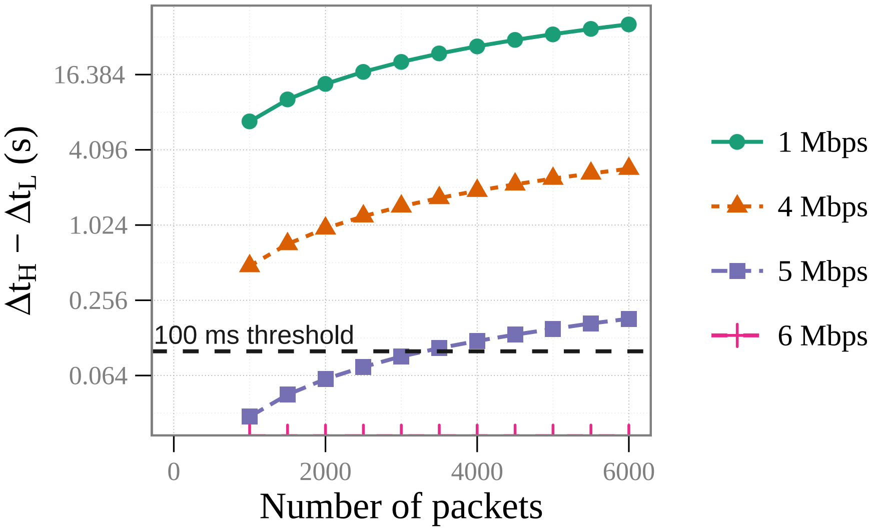


Figure 7.3: Comparison of scenarios based on how narrow the compression bottleneck link is (in log-scale).

have proved to be unable to compress/decompress fast enough for a high-speed network, thus creating a bottleneck on that link.

7.5.2 Scenario II

In this scenario we examined the relationship between how effective the deployed compression is and the detection rate of our detection scheme.[1] To carry out this simulation, we set the link capacity of the first and last links to a fixed rate of 5Mbps. We then ran the simulation on numerous values for the link capacity of the compression link, $C = \{1, 4, 5, 6\}$ (Mbps).

As the results show (Fig. 7.3), our detection works well only when the compression link is indeed the bottleneck of the path. There is a correlation between how narrow the compression link is compared to the overall path capacity, and the number of probe packets needed to detect compression. The narrower the link, the smaller the number of the probe packets needed for detection. In fact, sometimes an order-of-magnitude fewer

[1] Compression efficiency is determined by how much the packet size is reduced by applying the compression algorithm [JPS07].

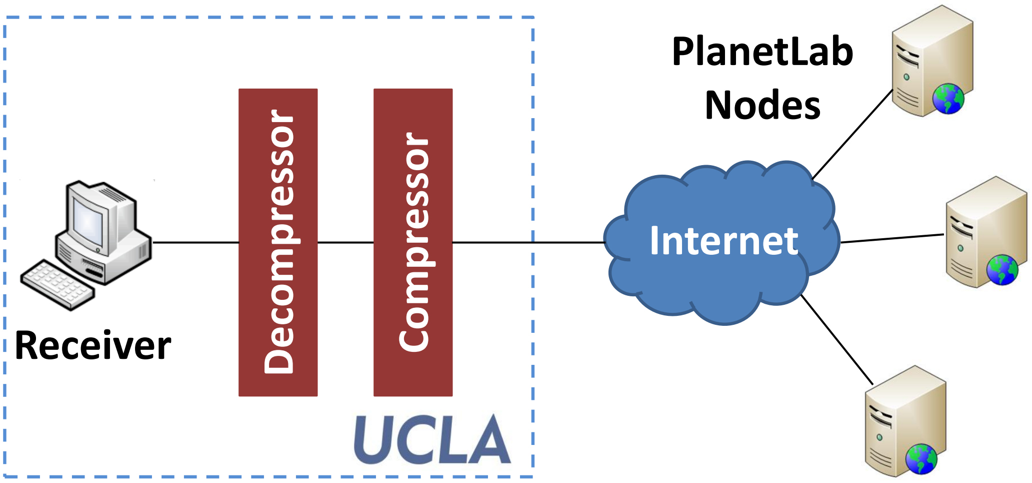


Figure 7.4: Environment used in our experiments

packets are needed, while maintaining approximately the same detection accuracy.

This stems from the fact that the effects of compression are more significant as the bottleneck becomes very narrow. On the other hand, compression is ineffective when it is not on the bottleneck. The result is that our detection method becomes rather ineffective (6 Mbps case in Fig. 7.3). However, we know that employing compression when it is not placed on the bottleneck link is a poor practice. For this reason, we expect that similar scenarios are relatively rare to find in practice, and hence this makes handling such scenarios less important. To summarize, simulation has confirmed that our approach detects only *effective* compression.

7.6 Internet Evaluation

In this section, we present our Internet evaluation to confirm that our method works on a real network. Here, we begin with the experiment setup, and follow with a demonstration of the results and an analysis.

7.6.1 Experiment Setup

To simulate the effects of link-layer, we used the Click Modular Software Router [KMC00]. The experiment environment and topology setup is depicted in Fig. 7.4. The compression and decompression components, as well as the receiver, were all located in UCLA. The senders are, however, remote PlanetLab [pla] nodes. We implemented LZCompressor and LZDecompressor Click elements for our experiment. Also, we reduced the sending transmission rate from the compression element to 1Mbps, making the compression link the narrow link of the path. Note that to confirm that our compression link is indeed the bottleneck, we used pathrate [DRM01b], a capacity estimation tools that has been proven experimentally to work well with PlanetLab nodes [LSB05a].

We could have placed both the sender and the receiver remotely and simulated a single bottleneck link on the path by routing the traffic stream through our local network. But by using a remote sender and a local receiver, we simulate a more common scenario, ensuring that the experiment matches the "no-valley" property, which derives from the provider-to-customer relationship and is the most commonly adopted routing policy by ASes [Gao01, WMW06].

7.6.2 Results

A set of ten geographically distributed PlanetLab nodes connected via the open Internet were selected for our experiment (Table 7.2). We define an experiment scenario uniquely as three elements: (1) a remote PlanetLab node, (2) whether we applied link-layer compression or not in the experiment, and (3) whether the cooperative receiver approach or the responsive receiver approach was used to detect compression in the experiment. For every scenario, we performed 24 individual experiments, within a span of 24 hours, running only one experiment every hour.

To illustrate how the aggregate data looks, we depicted the measurement histogram for one node when testing with the cooperative receiver (Fig. 7.5). Looking at the histogram, we observe that each set of data can be described by a normal distribution.

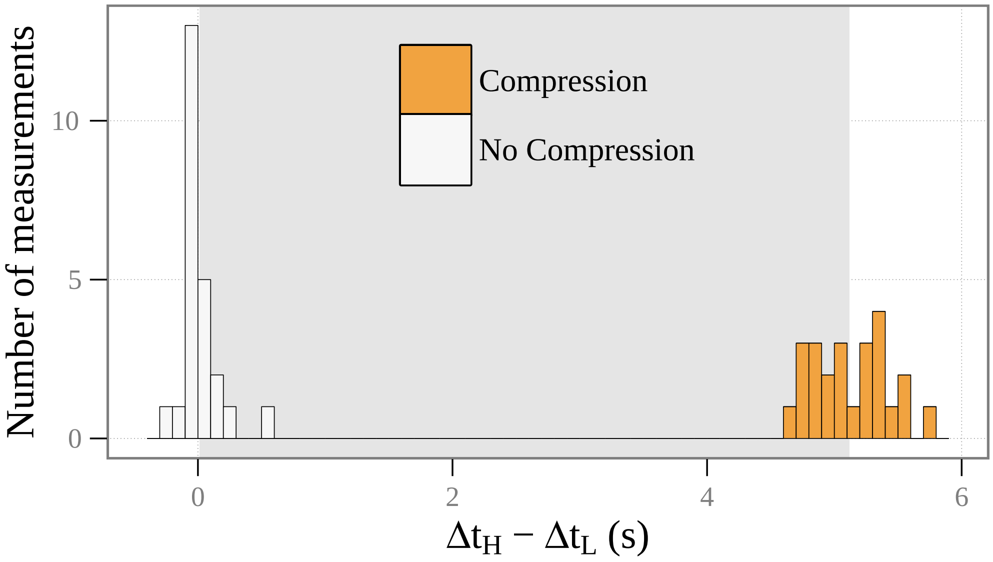


Figure 7.5: Histogram of $\Delta t_H - \Delta t_L$ for two sets of 24 measurements from a PlanetLab node in Singapore (the gray region indicates the gap between the means of the two distributions).

The noticeable gap between the two distributions is an indication of the compression effects. Also, another observation is that the measurements are slightly more spread for compression than non-compression. As discussed in Section 7.4.1, we use different sets of random bytes for the payloads of high entropy probe packets used for each experiment. This results in inconsistent compression ratios, which we suspect are responsible for the wider spread of timings when compression is applied.

Table 7.2 summarizes our results for all the experiment scenarios we performed by illustrating the normal distribution parameters in each scenario. The results confirm the existence of a significant gap in the presence and absence of network compression for all scenarios tested.

An observation from Table 7.2 is that the distributions for the responsive approach scenarios are relatively more spread compared to those of the cooperative approach. This is because in the responsive receiver approach, the ICMP reply packets travel the reverse path back to the sender, adding additional variability to the delay observations. In addition, since the effects of double compression are not very different than effects from single compression, a different kind of observation from our results suggests that, except for our own intermediary compression, there is no effective compression provided between the end-hosts selected in our scenarios in the duration of our experiment.

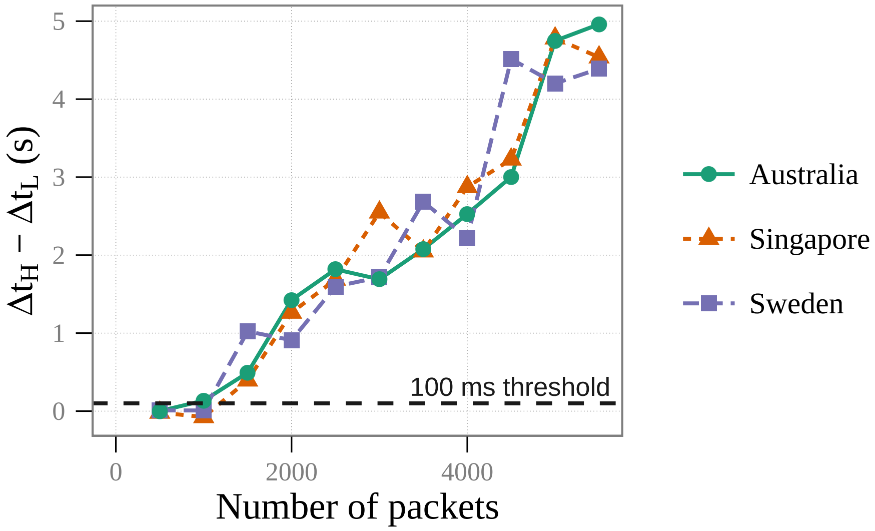


Figure 7.6: Comparison of the number of probe packets used for detection when using only a single measurement from three geographically diverse distant PlanetLab nodes.

Fig. 7.6 demonstrates the difference between the duration of the packet trains between high and low entropy for three different nodes, when considering only their first measurements. As can be seen, the plots are not as linear as those presented in the simulations in Section 7.5.1 where the measurements were performed in a clean and more controlled environment, yet they follow a similar increasing trend.

7.7 Discussion

7.7.1 Desirable Characteristics

In general, for any end-to-end active network measurements there is a set of desirable, and in some cases necessary, characteristics that should be satisfied.

			Cooperative Receiver				**Responsive Receiver**			
			Compression		**No Compression**		**Compression**		**No Compression**	
			$\Delta t_H - \Delta t_L$		$\Delta t_H - \Delta t_L$		$\Delta t_H - \Delta t_L$		$\Delta t_H - \Delta t_L$	
Location	**IP Address**	**Est RTT**	μ	σ	μ	σ	μ	σ	μ	σ
Singapore	203.30.39.238	201	5119	300	54	320	5010	430	78	450
California	128.111.52.59	5	5052	640	-42	290	4870	890	18	360
Brazil	200.17.202.195	188	5000	450	-46	130	5120	470	-35	620
Czech Republic	147.229.10.250	190	4389	440	-34	230	4102	830	25	1020
New Zealand	130.195.4.69	136	4632	530	79	410	5103	1050	-10	1205
Massachusetts	75.130.96.13	80	5388	410	13	230	5202	410	28	730
Canada	216.48.80.12	94	4624	440	161	700	4700	450	130	410
Sweden	192.16.125.11	185	1601	1220	115	390	2440	1300	150	1380
South Korea	143.248.55.128	173	4823	1140	155	550	4322	930	94	1100
Australia	130.194.252.8	188	4827	500	50	550	4730	780	-12	1240

Table 7.2: Summary of the results from the Internet experiments (msec).

7.7.1.1 Working with uncooperative intermediaries

Obtaining any information directly from routers is usually impossible, so end-to-end measurements should not rely on cooperation from them.

7.7.1.2 Robust against cross traffic

Cross traffic is usually present and can significantly affect the network measurements. Hence, measurement mechanisms should be accurate even in the presence of cross traffic. In addition, no assumption should be made on the characteristics of the cross traffic when considering its impact on the measurement.

We ran our experiment for a span of 24 hours and from different parts of the world. While we cannot truly confirm that during our measurements the network path experienced high volumes of cross traffic and congestion, the experiments were performed in realistic scenarios, accurately discovering the effects of compression in all of the scenarios. Normally (unless there is an ongoing long-term DDoS attack), cross traffic and high levels of congestion on a particular path persist for only a limited amount of time [MDH10].

7.7.1.3 Non-intrusive

Any detection approach using network measurements should not significantly affect the traffic in the path or the throughput of the other connections. If some active probing is necessary, it should be minimized. Also, a non-intrusive network measurement technique should not affect the actual property being measured. Our experiment consists of two sets of 6000 packets of 1100 bytes each; the sets are sent one minute apart. Each set adds up to a total size of 6.6 MB, which is equivalent to the size of a typical high-quality MP3 song [CLL02].

7.7.1.4 Short measurement time

Short measurement time is desirable, but not always required. For instance, it is required for available bandwidth measurements because the average available bandwidth can change over time; therefore, it is important to measure it quickly. On the other hand, IP-level or link-layer compression on the path is likely to persist for a longer period, so a quick measurement is less vital.

7.7.1.5 Minimal performance overhead

Minimal overhead ensures that the measurement can be performed on typical machines with typical resources, and also that it does not interfere with other processes running on those machines.

7.7.2 Timestamp Precision and Resolution Effects on Detection Quality

End-hosts that perform Internet measurements can introduce delays and bottlenecks that are due to hardware or operating system effects and have nothing to do with the network behavior they are measuring. This problem is particularly acute when the timestamping of network events occurs at the application level.

Clearly the more accurately we can measure the time, the better the outcome of the detection process. This is also true as the end-hosts use more accurate time resolution. However, in devising our technique and in the implementation of our experiments, we intentionally used only standard hardware and software. For instance, we avoided using any special hardware components for precise timestamping of packet arrivals, capturing packets at the kernel level, or packet sniffing applications such as libpcap [MLJ89]. This was done to ensure that our detection technique works for typical end-hosts with typical machines and resources. Our results confirmed this.

8 Detecting Middleboxes: A Systematic Methodology

If we can model it, we understand it.

In this chapter, based on lessons learned on preparing this dissertation, we recommend a set of steps to take for researchers who are interested in working on the problem of end-to-end detection of other middleboxes not addressed in this dissertation. In doing so, we present the phases required to achieve this and steps required in each phase.

8.1 Problem Definition Phase

8.1.1 What is that we want to detect?

It should be clearly stated that what is it exactly that we want to detect.

8.1.2 Assumptions

The scope of the general detection problem is generally very broad. Many variations of the problem can be formulated based on constraints and limitations imposed on the problem: e.g., the method used for detection and the degree of detection for the problem.

Before we begin with the design of any detection mechanism, we should prepare a list of assumptions to define the scope of the problem.

8.1.2.1 Degree of detection

A problem can be formulated to only answer an existential question about whether the network flow is influenced by some third-party or not. An instance of this problem can attempt to further characterize the influence to figure out what exactly the third party is doing to the network flow.

8.1.2.2 Locating the third-party node/link

The problem statement can be phrased from determining whether the path is influenced or not to precisely locating the third-party on the path or the link(s) influenced by it.

8.1.2.3 Method of detection

There are essentially two methods of network measurements for detection available to the end-hosts: active and passive measurements. Passive measurements have the goal of minimally affecting the measured network, by merely monitoring traffic on the network and inferring measurements from the observed traffic. Active measurements involve interacting with the network to make measurements, usually by sending probe packets.

In active measurements, we refer to the two end-hosts as the sender and the receiver. Based on how (and if) the receiver cooperates in the detection process, we present three variations to this problem:

A. Receiver is *uncooperative*:

   The receiver does not respond to the sender's requests that are beyond their primary purpose of communication. For examples, most web servers expect only HTTP requests and responses.

B. Receiver is *responsive*:

The receiver responds to the sender's requests beyond their primary purpose of communication as long as it does not require any changes on the receiver's machine. For example, the receiver responds to the sender's ICMP requests.

C. Receiver is *cooperative*:

The receiver is willing to make necessary changes on its machine or system to fully cooperate with the sender in the detection process.

8.1.2.4 Assistance from intermediaries or other parties

If all or some intermediaries on the path are responsive, active measures can also be used to get intermediaries to respond with valuable information. In another instance of the problem, the end-hosts assume that the intermediaries are uncooperative. Due to the end-to-end nature of our problem definition, we exclude the scenario where the intermediaries are cooperative. For the same reason, we exclude the instance where the end-hosts utilize help from volunteer nodes on the network that are not on the path.

8.1.2.5 Detection by comparing to unperturbed channel

One instance of the problem is when end-hosts have a model of the channel in the absence of third party's influence (perhaps captured in the past). In this case, the presence of the third party could be determined by comparing the current network behavior to the model. The other instance of this problem, however, is when the end-hosts do not have access to such information.

8.2 Preliminary Phase

We believe that when starting on the design of a detection mechanism, it is necessary to find answers to the following questions before offering any detection mechanism. In this phase, answering these five questions defines the steps to complete this phase.

In every step of this phase, we ensure that besides clearly addressing the question, and the scope of it, we also make sure to address the following questions about it: Why having an answer for it is important, or at least useful in the detection process? What are the potential challenges in answering it? What is my proposed plan to complete the step?

Some of the steps in this phase require thorough understanding of the current state of technology and available tools. Some require careful analysis and examination and some require extensive experimentation. We leave the technical details out of these steps here and limit ourselves to the high level presentation of the subproblem and the expected outcome. Of course, for instance, if a step requires experimentation, that means that questions must be answered, such as how exactly to implement it, or what would a suitable environment to run it on, etc.

[The Path Property Step "P"] 1) What are all the important pieces of information about the network path properties available to typical end-hosts?

Similar to any detection process in real world, special tools are needed to observe and to look for signs leading to detection. Hence, it is crucial to know what information and tools are available to endhosts that are observable and measurable to utilize them effectively for the detection process.

For instance, some network properties such as RTT, packet delay variation, available bandwidth, and hop count might perhaps be measured by the end-hosts, meaning they become available to them. On the other hand, some other valuable and helpful information such as the queue size of the routers on the path is not available to typical end-hosts. Technically, we are only interested in standard methods and tools available for standard computers with standard equipment. For instance, we assume in an end-host, processing time and packet arrival time can be measured and its TCP congestion control mechanism's behavior is observable. On the other hand, we exclude using irregular methods to obtain some information. For instance, we do not consider using measuring electromagnetic emissions to detect the queue size of a router.

We refer to the set of all available network properties about the path as P.

Challenges:

- Understanding exactly how well can these properties in P be measured and how the existing tools' imprecision and potential inaccuracy in measurements would affect the accuracy of the detection mechanism.

[The Influence Characteristics Step "I"] 2) What are the characteristics of the influence I?

To detect the presence of I, we must have a way to distinguish I from other types of influences. Therefore, it is necessary to find unique, indicative, and distinguishing characteristics of I.

This step requires gathering and understanding all, if any, publicly available and known information about the internal design of the influence in question. A thorough and perhaps creative analysis is necessary to find subtle unique characteristics about I that can be used to detect it. For instance, encryption applies relatively constant delay, or the fact that the effects of compression impose different delays for fixed length data but with low and high entropy content. Technically, we are mainly interested in observable and measurable effects on elements of P and not interested in hidden or non-measurable effects of I.

Challenges:

- These characteristics are not always obvious and might require thorough and creative analysis and maybe some experiments to find them.

[The Determinant Factors Step "D"] 3) What elements in P are impacted by I (and to what extent)?

The values of P, by definition, are all the endhosts can know. Therefore, the only way to detect I, through examining the path properties, is by looking at changes in their values in presence and absence of I.

This is an important step in identifying the indicative factors in detecting I. We define D as the set of all indicative elements of P in detecting I ($D \subseteq P$). Intuitively, if the value of a path property remains constant in the presence or absence of I, then it is not a helpful piece of information in the detection process. It is also important to figure out which of these elements are more indicative than the others in detecting I. Further, we must investigate any interdependency relationship between members of D.

Challenges:

- How exactly to assess the relative level of importance of $d \in D$ in detecting I?
- How inaccuracies in measurements influence our findings?
- How exactly to derive the interdependency relationship between members of D?

Literature would probably help finding answers in this step by examining what has already been done in this area. In some cases, theoretical analysis would lead to a hypothesis on D, where further experiments could verify it. We coule simulate the effects of I in a clean environment (i.e., in the absence of network variation and any other types of traffic than the one generated by the packet probes). We coule then look for changes in values of P.

[The Normal Network Step "N"] 4) What are the sufficient, yet unavoidable, assumptions about the normal network behavior, in the particular network environment, required to make any comments on detectability of I feasible?

Even if we make the assumption that the end-hosts have no access to information on the unperturbed channel, for any detection mechanism to work, there should be a precise definition for the notion of normal behavior to be used as a reference. For instance, in designing a mechanism to detect compression on the network (presented in Chapters 4 and 7), we assume that normal network behavior will treat packets with equal lengths equally, even in the case that the data entropy of their payload is different.

This is where we define the specific network environment (either in high level properties or low level). For instance, if it is multi-hop wireless network, then we expect a higher rate of loss of packets, whereas in a wired network, random losses are rare events. Another example is to impose an upper-bound on the value of RTT.

These are basically conditions, assumptions, and constraints on the elements of D. But only those that are in D are significant, because $P - D$, by definition, is expected to remain relatively constant, hence there is no need to check if they have deviated from normal behavior.

Basically, the violation of assumptions on the elements of D would be used to indicate the existence of such influence as we define them as deviations from normal network behavior.

Challenges:

- What is precisely "normal" and how to define it?
- Coming up with not only correct, but tight assumptions about normal network behavior is crucial. Failing in coming up with correct assumptions would lead to false positives. On the other hand, loose assumptions would almost certainly lead to undesirable false negatives.

[The Network Variation Step "V"] 5) Does normal network variation (e.g., network congestion, load balancing effects, link failures, and dynamic routing effects) influence the end-to-end detection of I?

Any proposed detection mechanism, to be useful, should work in a real network. An approach that only works in a controlled and isolated environment is not very useful.

If the answer is "no" to the question posed in this step, then we can completely skip this step. For instance, detecting a third party that sends out spoofed control packets is not affected by normal network variation.

However, if the answer is “yes”, (which is the case for all delay and loss-based influences) then we proceed to the following questions.

- What is the specific set of normal network variation that we care about in the detection process, V? For instance, one could focus only on network congestion due to its popularity.
- How well is I distinguishable from effects of V?

Note that it is generally true that congestion usually hinders the detectability of loss or delay-based influences. Clearly, if congestion, in some cases, helps the detection process, one could impose network congestion to makes detectability easier.

Challenges:

- Network variations that resemble normal loss and delay in the network are particularly challenging. This is because intentional and normal loss or delay in those cases are perhaps not easily distinguishable. This leads to another issue: how I is distinguishable from V?
- Active probing used for the detection process may contribute to network congestion.

8.3 Design Phase

This is a crucial phase in the process where we use the information obtained in the preliminary phase to produce the detection algorithm.

If, by considering only Steps P-I-D-N, we are able to devise an algorithm for detecting I, then this would indicate that I lies on the unknown detectable or strictly detectable region (left side of the green line in Figure 8.1). This is a reasonable first step here before proceeding to incorporating our findings in Step V of the preliminary phase. Clearly, failing to come up with such a detection mechanism does not imply that I is fundamentally undetectable.

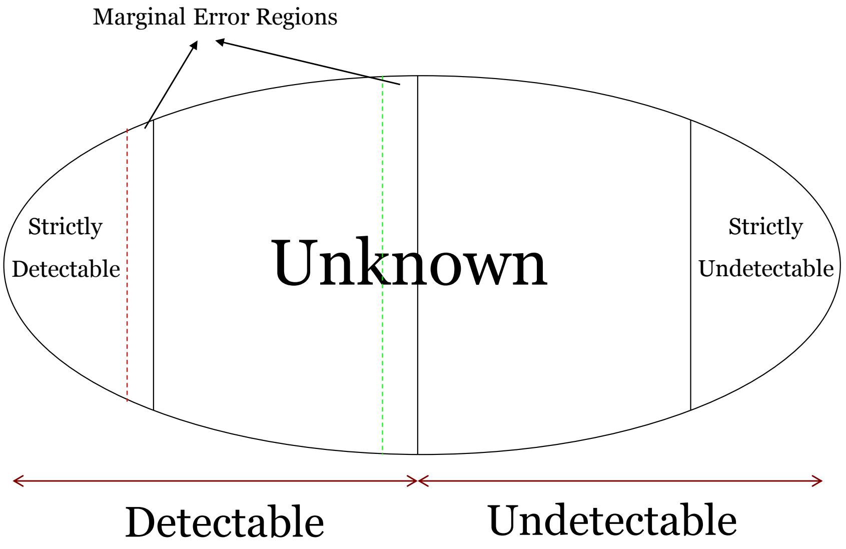


Figure 8.1: Partitioning the universal set to strictly detectable, strictly undetectable, and unknown problems with considering the marginal errors.

Note that $\forall d \in D$ used in the detection mechanism, inaccuracies involved in measuring d must be taken into consideration.

The detection phase requires creativity and knowledge of the subject to utilize the information gathered in the past phases in developing such detection process. We recommend testing the initial approach in a simulated environment to verify the approach. Then information from Step V could be incorporated to produce an approach that works well in real networks.

8.4 Evaluation Phase

Any proposed detection mechanism must be not only validated, but also evaluated under certain or all network conditions to confirm that it successfully detects the influence in question. This requires answers for some questions such as:

1) How can we validate/evaluate our proposed detection algorithm?
2) What is the most suitable environment to test it?
3) How confident are we in the results of the evaluation system?
4) What metrics should we use to evaluate our detection mechanism?

Today's open platform for network measurement and distributed system research, which are usually collectively referred to as global network testbeds, provide opportunities for controllable experimentation and evaluation systems at the scale of hundreds or thousands of hosts. These testbeds have been successfully used to advise a variety of important applications addressing IP reachability, prefix hijacking, and routing anomalies.

The Internet is growing in ways that make increasingly difficult to attain a global view of the network. Large swathes of the network cannot be probed directly from our research testbeds and a number of valuable measurement techniques have side effects that render them impractical.

PlanetLab [CCR03] is widely used by the research community and generally a suitable candidate for such purposes. For every researcher working on PlanetLab there is a need to know what bias the system introduces in the collected experiment results. For instance, one should keep in mind that PlanetLab is not completely representative of the current Internet. This testbed is all about small, long running services in specific locations. There are also some shortcomings associated to the use of PlanetLab as well. For instance, we normally have very little, if not no, control over the actual path between the chosen sender and receiver. Depending on the nature of the experiment, this could make PlanetLab somewhat ineffective.

Challenges:

- Find a platform to test the proposed detection mechanism on a global testbed, specifically one that allows us to possibly overload their nodes, if needed. It is highly desirable that the chosen testbed gives us some control over or information about the path between the nodes. For instance, how are we going to evaluate our compression detection mechanism if we do not know whether link compression exists on the path between two nodes of the testbed or not?

- What metrics should be used in the evaluation process and how to accurately measure them: e.g., false positive rate, detection rate, performance overhead, and packet delivery rate?

- Is there a need for a general evaluation system that applies to detecting all influences? How feasible is this idea?

- Recent studies, suggests that testbed results for Internet systems do not always extend to the targeted deployment. For example, Ledie et al. [LGS07] and Agarwal et al. [AL09] show that network positioning systems perform much worse in the wild than in PlanetLab deployment. Identify such inconsistencies to avoid false claims.

9 Overview of Complementary and Related Work

To best our knowledge, so far, all work done on revealing middlebox interference is very specific to the type of middlebox. In this chapter, we discuss prior work in detecting each specific type of middleboxes.

9.1 Firewalls and IDSs

FireCracker [SEA07] proposes a framework that, through tailored probes, could be used to blindly discover a firewall policy remotely as a blackbox and without prior knowledge about the network configuration. In a more recent work, SymNet [SPN13] proposes a static analysis technique that can model stateful middleboxes such as stateful firewalls and IDSs.

9.2 Packet Header Modifiers

Some work has been done to detect the presence of NATs and large scale NATs on the path [Muller2013][DHB13][Stoenescu2013]. There is also work to detect middleboxes that modify TCP Fields. Tracebox [DHB13] detects middleboxes that modify TCP sequence and acknowledgement numbers, as well as TCP MSS option. Glasnost [DMG10], on the other hand, detects modified TCP advertised window size.

9.3 Network Neutrality Violation

Detecting traffic discrimination has drawn more research attention than detecting other types of middleboxes. The majority of such detection mechanisms [Zhang2009][KD10b, DMG10][Weinsberg2011] use the relative discrimination technique, where they test each application (the measured flow) against a flow that is assumed to be non-discriminated (the baseline flow). NetPolice [Zhang2009] proposed an active probing methodology that replays application traces with limited TTL values to solicit TTL-expired messages from intermediate routers. They compare loss rates with an HTTP flow as a baseline, and show discrimination in backbone ISPs. One limitation of their approach is that NetPolice relies on router generated ICMP responses; the generation of such packets is subject to rate-limiting and vendor-specific lower-priority processing.

Bin Tariq et al. propose a passive detection methodology, NANO [TMF09], which uses throughput observations from many end-hosts to detect discrimination. They use causal inference in the client data, and information about confounding variables to group clients according to performance. BTTest focuses on detecting BitTorrent traffic blocking by ISPs using forged TCP RST packets. It emulates BitTorrent flows, and correlates client and server traces to detect RST messages. The authors show that ISPs mostly block BitTorrent in the upstream direction and classify based on payloads. Siganos et al. [Signos2009] use the BitTorrent protocol to measure download throughput across ISPs, and show that the measurements are correlated with performance reports from Akamai.

Lu et al. propose POPI [LCB07] to detect priority based forwarding. They use high-rate active probing to induce losses, and observe loss rates to analyze forwarding. Kuzmanovic et al. [DMG10] use passive monitoring at a single-hop path to observe service rates at different timescales and infer the parameters of a WFQ scheduler. Biczok et al. [Biczok2010] propose a method to detect discrimination with prior information about the classifier type. They send a single flow and observe performance difference between the sender and receiver sides. Mahajan et al. use ICMP probes and associate performance with geography to measure differences in backbone ISPs in NetDiff [Weinsberg2011]. Automated traffic classification often relies on machine learning and statistical techniques. A recent comparison of traffic classification methods is presented in [DMG10]. Commercial classification products include (but not limited to) Sandvine PTS8210 and Cisco NBAR. Traffic Morphing [LCB07] shows that it is possible to avoid certain kinds of classifiers by altering flow characteristics.

ShaperProbe [KD11] detects and further estimates the shaper parameters by having the end-user send traffic to a monitoring server, while keeping track of the rate at which the server receives the user's traffic.

9.4 Redundancy Elimination

Our preliminary work [PAR14] introduces an end-to-end detection mechanism for lossless network compression of traffic flows by intermediaries. In this paper, we described a method to allow end-to-end detection of intermediary-provided link-layer or IP-level compression. Such an end-to-end approach does not require any changes to or cooperation from intermediary nodes, making its deployment and use much more practical. We propose two end-to-end approaches based on a receiver's cooperativeness in the detection process. A cooperative receiver is willing to make necessary changes on its machine or system to fully cooperate with the sender in the detection process. A responsive receiver only responds to the sender's requests as long as they do not require any changes on the receiver's machine. For example, our approach assumes that the receiver responds to the sender's ICMP requests. In this approach we dealt with a single-sender/single-receiver path of a communication network. Our approach is resilient to cross traffic and other Internet variabilities, and is non-intrusive, making it both practical and deployable. Also, our proposed solution uses only regular unicast probes, and thus it is applicable in today's Internet.

Han et al. in [Han2011] briefly outline an approach to detecting RE-enabled routers on the path and leave elaboration and implementation to future work.

9.5 Sending Spoofed (Forged) Control Packets

BTTest [Dischinger2008] focuses on detecting BitTorrent traffic blocking by ISPs that use forged TCP RST packets. Weaver et al. [Weaver2009] also introduce a method to detect connection disruptions via forged TCP RST packet, but they focused only on detecting the method that China's Great Firewall uses.

9.6 Delaying or Dropping Selective Packets

Very few and even more specific works focused on detecting intentional delaying and dropping of selective packets. Song et al. [Song2007] propose two different approaches to detect and further accommodate delay attacks in wireless sensor networks. Chang et al. [Chang2010] identify TCP victims (as the first step to detect the attacker) by monitoring their drop rates to mitigate low-rate TCP-targeted attacks [Kuzmanovic2013].

9.7 End-to-End Network Performance Measurement

There are also a number of areas that are complementary to our proposed research scope. End-to-end network performance measurements for QoS assessment is an important complementary research in our work. Also another relevant and complementary work to our research is end-to-end network performance measurements. This is a well-studied research area. TCP congestion control mechanisms essentially are attempts to discover network conditions, albeit not necessarily intentional and mostly congestive conditions. Afanasyev et al. [ATR10] presents a comprehensive survey of TCP congestion control mechanisms. Many work have been done on measuring and modelling loss and delay on the Internet [ACI99, Bor00, Bol93, SBD05, BSU98]. End-to-end bottleneck detection and available bandwidth or capacity measurement is also another relevant work to our proposed problem.

As an example we examine measuring path capacity in the Internet which is a problem that has received a significant attention from the research community. The capacity of a path is formally defined as the maximum IP-layer throughput that a flow can get on that path and is determined by the link with the minimum capacity among all links on a path.

Existing capacity estimation tools can be classified in three categories. First, active tools that rely on the injection of measurement packets in the network. Most existing tools are active, including Pathchar, Pathrate [DRM01a] or Capprobe. The second category consists of so-called embedded tools that alter the pattern of users traffic to estimate capacity without introducing additional measurement traffic. A typical example is TFRC Probe, which is an embedded version of CapProbe. The third category consists of purely passive tools that aim at extracting capacity estimates from passively collected traces of traffic.

10

Future Work

We have pointed out a number of future work plans throughout this dissertation. In this chapter, we briefly lay out some of the more general future directions of our work.

1) A general framework for evaluation methodology:

Any proposed detection mechanism must not only be validated, but it must also be evaluated under certain or all network conditions in order to confirm that it successfully detects the influence in question. This requires answers for some questions such as how we can evaluate our proposed approach and what metrics we should for this assessment.

While there have been numerous works on detection of third party middleboxes (as discussed in Chapter 9), these proposed approaches do not use coherent and common validation techniques to evaluate their proposed detection mechanisms. This could be because there are fundamental challenges when it comes to validating detection mechanisms. Below we examine two of these possible challenges:

1. **Lack of ground truth knowledge:**

   To be able to assess the accuracy of the detection process based on typical metrics including false positives and false negatives rates, it is required to know whether such a middlebox exists on the path or not. This information is typically hidden from or unavailable to end users.

2. **Not sufficiently representative:**

   Today's open platforms for network measurement and distributed system research, which are referred to as global network testbeds, provide opportunities for controllable experimentation and evaluation systems at the large-scale number of hosts. The main problem is that these testbeds are not completely representative of the real world. Often global testbeds such as PlanetLab used for research purposes are placed in somewhat unconstrained networks, so it is very likely that the class of middleboxes in question do not exist on their access network. Also, recent studies suggests that testbed results for Internet systems do not always extend to the targeted deployment. For example, Ledie et al. [LGS07] and Agarwal et al. [AL09] show that network positioning systems perform much worse in the wild than in PlanetLab deployment.

While in this dissertation we used a coherent and systematic approach for validating our various detection mechanisms, devising a general evaluation framework that could be used by others requires more attention. This is one direction we plan to take in future.

2) Detecting middleboxes that delay packets in a different way that we discussed in this document:

In this document, we only discussed middleboxes that add some additional delay for every packet on particular network flows. However, the effect of delay by some middleboxes might not be observable on packet-basis (that is, if they add highly variable delays, so that only the aggregate effects are perhaps distinguishable). In other words, δ_{min}, which is defined in Chapter 3, could be zero or even negative, while $\sum_{i=1}^{N} \delta_i > 0$ for sufficiently large value of N always holds.

In addition, there are middleboxes that impose non-discriminatory delays. For instance, VPNs encrypt and decrypt every packet of every flow in a non-discriminatory fashion.

Detecting middleboxes with these unique types of interference by end-hosts is another direction of future work.

3) Detecting other types of payload-preserving third parties other than those that delay and/or drop packets:

There are many other types of middleboxes that, while preserving the content of the flow, affect the traffic flow other than by delaying or dropping packets. To name a few, there are middleboxes that send spoofed (forged) control packets, duplicate packets, split packets, or combine packets. For some of these cases, it might not be very clear why the end-hosts would want to know if that is what is happening to their traffic flow. However, at least it could help in characterizing such middleboxes. Needless to say, it is also valuable to detect as many middleboxes as possible.

4) Detecting middleboxes that drop packets in a different way that we discussed in this document:

Some third party middleboxes drop packets within the same flow intentionally for various reasons:

1. **Selective dropping:**

   Discriminatory buffer managers, such as WRED, drop packets already backlogged or arriving, based on the class of those packets. For example, a queue may have lower thresholds for lower priority packets. A queue build-up will cause the lower priority packets to be dropped, hence protecting the higher priority packets in the same queue.

2. **Indiscriminatory dropping:** An example of such intermediaries are those that intentionally drop packets through network dissuasion technique [PKR12] for opera-

tional management purposes. In this technique, individual UDP flows are throttled by their packets being randomly dropped through a predefined two-state first-order Markov chain model. TCP flows are throttled rather more deterministically through alternating throttling and normal intervals, but the idea is to throttle the network without introducing any discrimination.

11

Final Thoughts and Conclusions

One class of intermediaries that we defined in this dissertation is payload-preserving middleboxes. These middleboxes make no changes to the content of the traffic, giving the appearance that nothing has been done to the stream other than routing it to the destination. This transparency property can make end-to-end detection of such intermediaries very challenging in most cases. We had two contribution in this dissertation. We conclude this dissertation by pointing out these contributions:

- One contribution of this dissertation is to investigate the detectability of such middleboxes by seeking answers to this question: "can the sender or receiver (or both if they cooperate) determine that something of this kind has been done if they pay attention?"

- The second contribution of this dissertation is to view and analyze the universal set of all middlebox interferences on network traffic as a whole. We introduced the notion of normally detectable middleboxes, denoting that a certain type of middleboxes is detectable under normal Internet conditions, with a specific degree of certainty. In this research we have developed a general framework to detect network discriminators, which we have defined as middleboxes that delay and/or drop packets in a discriminatory fashion. We achieved this by modelling their interference on the network to propose a unified solution to the detectability problem, rather than ad hoc approaches that are only applicable to a specific type of middleboxes.

To illustrate the implementation feasibility of our generalization idea, we then used our results to detect a number of prevalent and important middleboxes: network compression, traffic prioritization, and traffic shaping/policing. In addition to the analytical approach that we took to solve our proposed problem, our results are also supported by network simulations and live Internet experiments.

In the dissertation we primarily focused on investigating the feasibility of the detection of payload-preserving middleboxes. While we believe our approaches are applicable in today's Internet and can be employed on typical end-hosts, we leave more thorough discussions on practicality and intrusiveness to future work. In additions, active probes are exogenous and also might be detectable (typically) and hence the third party can manipulates them (e.g., to prioritize them in the case of detecting strict priority queuing), thus rendering the detection tools useless. Lastly, we hope that this work inspires the new area of research that concerns with the detectability of third party interferences on the network, by introducing fruitful research problems.

A Implementation Details

In this chapter, we discuss our implemented `ns-3` modules and Click elements and configuration files used to validate our approaches in more depth. This is primarily intended as a guideline for other researchers to use our implemented components[1].

A.1 NS-3 Modules

The value of simulation for enabling faster and easier experimentation and validation of models, architectures and mechanisms in the field of networking has long been recognized. A long list of networking protocols have been studied using simulation environments and then subsequently tested in real circumstances and production environments. Using simulation, a researcher can validate and fine-tune a proposal to scales that are not easily and cheaply achieved otherwise, and can overcome practical difficulties that would otherwise prove insurmountable in an initial stage of an idea's development. Evaluation and testing in real networks would require big investments in software development and very often hardware development. In addition, a lot of equipment would be needed to conduct experiments on a large scale having multiple sources and destinations of different types. Small scale evaluation in a lab, wide-area scenarios, and custom simulators can all be valuable; however each has some significant shortcomings. It is difficult to reproduce the actual

[1] An even more detailed version of our implementation is presented in our three technical reports [PFM14, CP14a, CP14b].

mix of traffic and topologies found in real networks, they may require substantial effort and expense, and repetition of experiments under controlled conditions may be difficult. Therefore, real network testing is too costly for most research institutes at least in the first phases of development. Testing in a simulation environment is a more appropriate and inexpensive solution for evaluating networking mechanisms and protocols.

The `ns-3` simulator is a discrete-event network simulator targeted primarily for research and educational use. `ns-3` allows users to build virtual network topologies and simulate network activity, all virtual network activity can be observed this way and makes it ideal for experimentation. The modules are easily extended to add new functionality, a design feature we have taken advantage of to build these three modules. The `PointToPointNetDevice` class handles sending and receiving from a network node to the link. All three of the modules described below are accessed by this class.

The simulator is implemented in a modular fashion, using separate sets of C++ classes to model the behavior of the middlebox in question.

A.1.1 LZ Compression

The compression module is accessed using a member pointer in `PointToPointNetDevice`. The module is given a packet during `PointToPointNetDevice::Send()` which it compresses and returns in compressed form. The packet is then sent normally and decompressed by the next net-device in the `PointToPointNetDevice::Receive()` function. The compression function compresses the entire network packet, except the point to point link headers which have not yet been added by `PointToPointNetDevice`. Then a compression header is added with the original length of the data (Figure A.1). The decompression function reads this compression header and uses the original length to decompress the packet to its original format. The module uses the `inflate()` and `deflate()` methods from the LZ library [zli] with the `Z_BEST_COMPRESSION` flag set for `deflateInit`, this initializes the compression algorithm to prioritize best compression ratio over computation time. The `NO_FLUSH` flag is set for each `deflate()`, this lets the algorithm compress over the entire data set rather than in blocks.

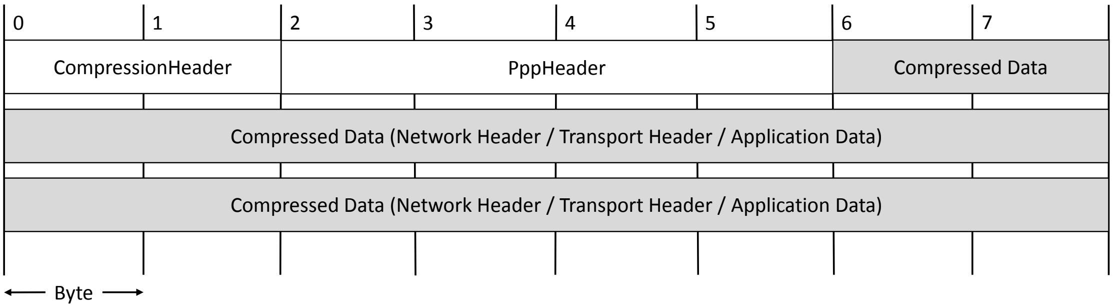

0
1
2
3
4
5
6
7
CompressionHeader
PppHeader
Compressed Data
Compressed Data (Network Header / Transport Header / Application Data)
Compressed Data (Network Header / Transport Header / Application Data)
Byte

Figure A.1: Header format of the compressed packet using our `ns-3` module.

Example Usage:

```
CompressionHelper myCompression;
myCompression.Install(netDevices);
```

Figure A.2: Usage example for `ns-3` the LZ compression module.

Simulation Validation:

To show that compression behaves as expected we ran three experiments on the topology shown in Figure 4.4. First a series of packets with a payload of all zeros were sent with no compression, then the same packets were sent with compression enabled, and finally a series of packets with random data were sent with compression enabled. With all other conditions equal, we should expect to see the zero packets with compression arrive faster than the uncompressed and random packets since they are moving faster along the second link.

A.1.2 Delay Discriminator

The delay discriminator module inherits from the `Queue` class and is passed to the `PointToPointNetDevice` as its `Queue` pointer. The class simulates a receive queue, a classifier, a delayer, and a transmission queue. When the net-device passes a packet to the delay discriminator, it is queued for delaying in the receive queue. The delayer takes

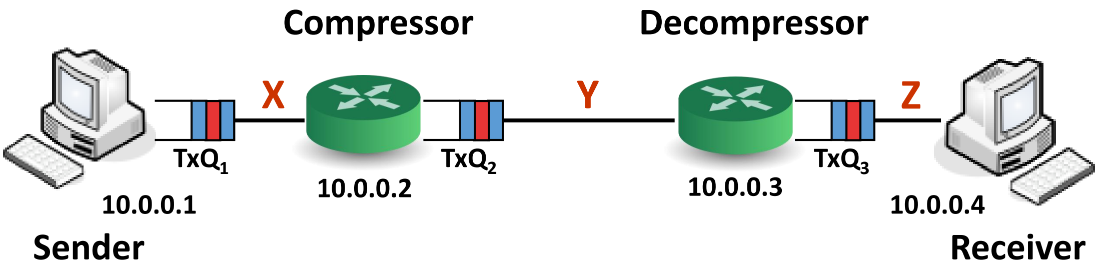


Figure A.3: Simple topology to test our implemented LZ compression module.

a packet from the queue and checks its destination port against a configurable map from ports to delay times and applies the appropriate delay. After this delay has passed, and there is room in the transmission queue, the packet is added to the transmission queue. If the transmission queue is full beyond a designated threshold then the delayer will hold the packet until the the queues size is below the threshold. After a packet has been passed to the transmission queue, the delay will take another one from the primary queue.

Example Usage:

```
DelayHelper myDelay;
myDelay.SetAttribute(“RxQueueSize”, UintegerValue (4000));
myDelay.SetAttribute( “TxQueueSize”, UintegerValue (4000) );
myDelay.SetAttribute(“TxQueueThresholdPercentage”, UintegerValue (95));
myDelay.SetAttribute(“Port1”, UintegerValue (2000));
myDelay.SetAttribute(“Delay1”, StringValue “100 nanoseconds”);
myDelay.Install(netDevices.Get(1));
```

Figure A.4: Usage example for `ns-3` the delay discriminator module.

Simulation Validation:

To show that delay discrimination behaves as expected, we ran two sets of experiments on the topology shown in Figure A.5. In the first set we sent a series of slowly spaced packets across the network, once with no delay and once with a 100 μ-sec delay. Since the packets were too far apart to queue at the nodes, we should see each packet in the second experiment arrive at exactly 100 microseconds later than it did in the first experiment.

In the second set, we sent a series of closely spaced packets, set all the bandwidths equal, and introduced a 1 ms delay. even though all bandwidths are equal, there should still be packet loss due to queueing behind the delayer.

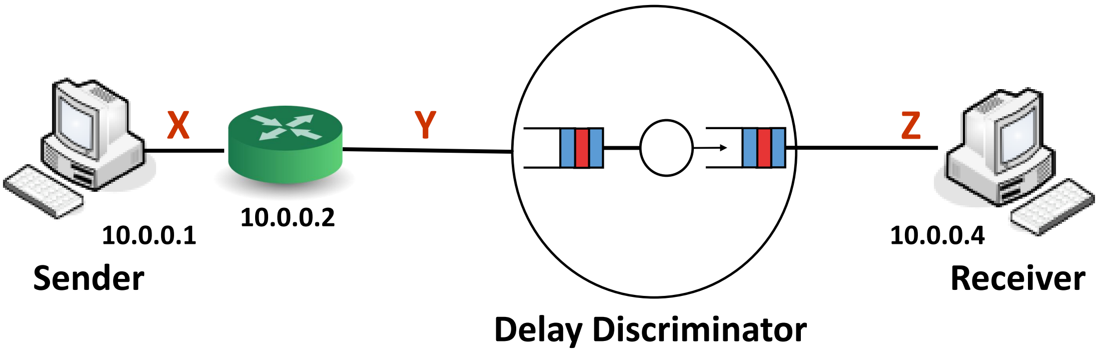


Figure A.5: Simple topology to use the delay discriminator module.

A.1.3 Strict Priority Queueing

The strict priority queueing module is similar to the delay discriminator. It inherits from `Queue` and is assigned as the transmission queue of `PointToPointNetDevice`. The module simulates a classifier, a high-priority queue and a low-priority queue, and a scheduling delay. When a packet is queued from the net-device, the classifier checks the destination port to determine if it is high or low-priority and places it into the appropriate queue. The scheduling delayer only dequeues from the low-priority queue if the high-priority queue is empty. The scheduling delayer applies a uniform delay to all packets. After the delay it ensures there is room in the transmission queue and then adds the packet. If the transmission queue is filled beyond the threshold then its behavior is the same as the delay discriminator.

Example Usage:

Simulation Validation:

To show that strict priority queueing behaves as expected, we ran an experiment on the topology shown in Figure A.7. In the first we sent a series of packets that alternated one high-priority and two low-priority and dropped the bandwidth and transmission queue threshold on second link. Since the high-priority queue will be filled as long as they keep

```
DelayHelper myDelay;
myDelay.SetAttribute("LowQueueSize", UintegerValue (4000));
myDelay.SetAttribute("HighQueueSize", UintegerValue (4000));
myDelay.SetAttribute("TxQueueSize", UintegerValue (4000));
myDelay.SetAttribute("TxQueueThresholdPercentage", UintegerValue (95));
myDelay.SetAttribute("Port1", UintegerValue (2000));
myDelay.SetAttribute("Priority1", StringValue "high");
myDelay.SetAttribute("Port2", UintegerValue (3000));
myDelay.SetAttribute("Priority2", StringValue "low");
myDelay.Install(netDevices.Get(1));
```

Figure A.6: Usage example for `ns-3` the SPQ module.

sending, we should not see any low priority packets come in until the very end of the experiment.

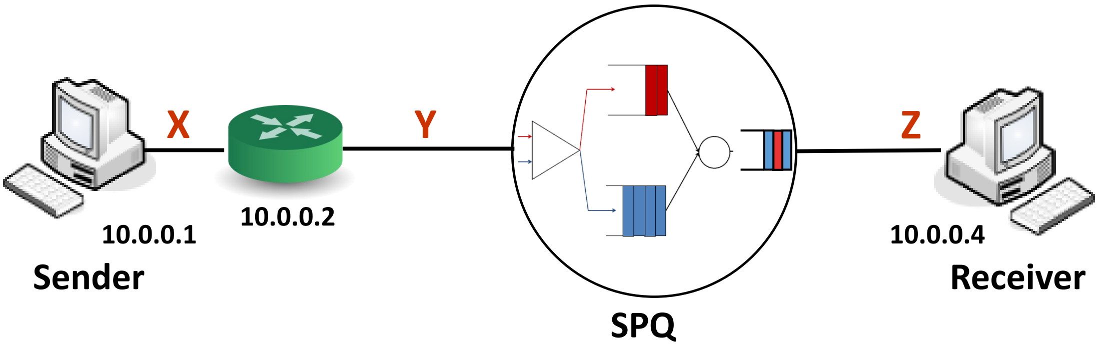


Figure A.7: Simple topology to use the SPQ module.

A.2 Click Elements

Click [KMC00] is a modular software router that can be used for fast prototyping of routing protocols. A software routers protocol is configured as a directed graph of asynchronous elements, the transitions between elements are implemented by either pushing or pulling packets to ensure these asynchronous components cooperate.

A.2.1 LZ Compression

Compression Element:

The Compression element has a packet pushed to its input, it compresses the data buffer of the entire packet using the deflate() method from the LZ library [zli]. The Z_BEST_COMPRESSION flag is set for deflateInit, this initializes the compression algorithm to prioritize best compression ratio over computation time. The NO_FLUSH flag is set for each deflate(), this lets the algorithm compress over the entire data set rather than in blocks. The compressed packet is then pushed to output[0]. If for any reason compression does not work, the packet is pushed to output[1]. It is up to the configured switch what to do with these packets.

Decompression Element:

The Decompression element has a packet pushed to its input, it compresses the data buffer of the entire packet using the inflate() method from the LZ library. The decompressed packet is then pushed to output[0]. If for any reason decompression does not work, the packet is pushed to output[1]. It is up to the configured switch what to do with these packets.

Compression Switch:

eth0 pushes packets to the Compressor to compress as described above, compressed packets are pushed to a queue to await the BandwidthShaper. The compressed packet is then sent out of eth1. Uncompressed packets (compression failed) are sent to the PacketSink to be discarded (Figure A.8).

Decompression Switch:

eth0 pushes packets to the Decompressor to decompress as described above, successfully decompressed packets are pushed to a queue to await eth1. Packets that could not be decompressed are sent to the PacketSink to be discarded (Figure A.9).

Validation:

To show that the compression switch behaves as expected we ran three experiments using two computers running click router. The first computer ran Compression Switch and was connected to the internet on eth0 and the second computer on eth1. The

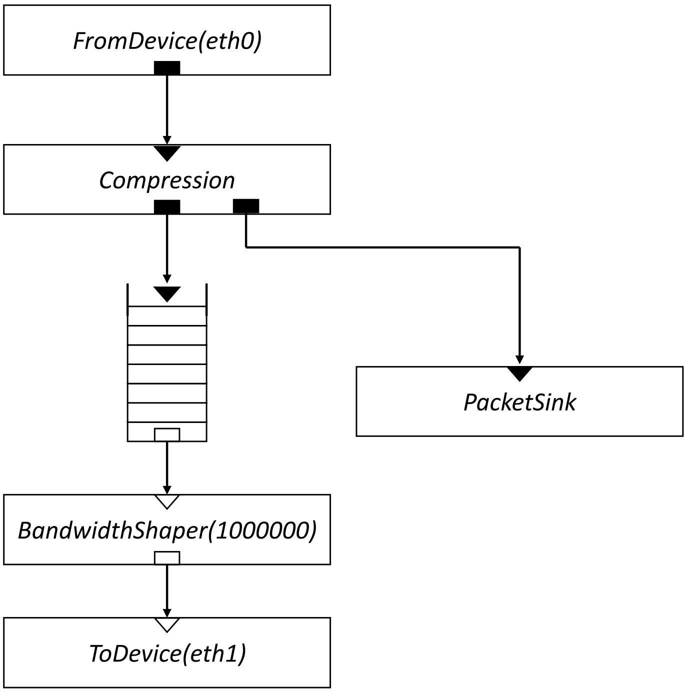


Figure A.8: Click configuration file tree diagram to use the LZ compression element.

second computer was connected to the first on `eth1` and a third computer on `eth2`. In the experiments we sent packet streams from a remote computer on the internet to the computer reachable through the two computers running click router. First a series of packets with a payload of all zeros were sent with no compression, then the same packets were sent with compression enabled, and finally a series of packets with random data were sent with compression enabled. With all other conditions equal, we should expect to see the zero packets with compression arrive faster than the uncompressed and random packets since they are moving faster along the second link.

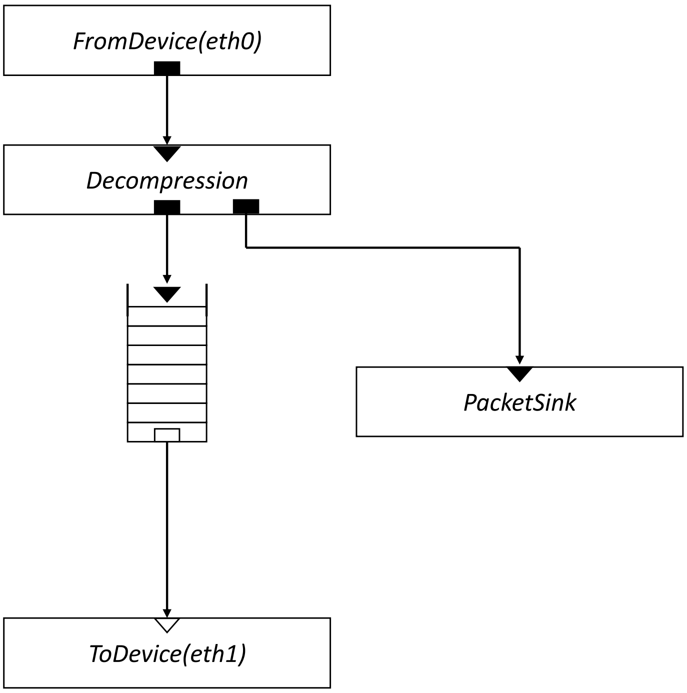


Figure A.9: Click configuration file tree diagram to use the LZ decompression element.

B Path Capacity Estimation

Throughout this dissertation, we mentioned that end-hosts are able to estimate the path capacity, c_p to a reasonable accuracy. In fact, one of the parameters in our detection mechanism is this value. Therefore, we felt compelled to discuss what capacity is and what are the tools available to measure path capacity.

Measuring path capacity in the Internet is a problem that has received a significant attention from the research community. The capacity of a path is formally defined as the maximum IP-layer throughput that a flow can get on that path and is determined by the link with the minimum capacity among all links on a path.

Existing capacity estimation tools can be classified in three categories. First, active tools that rely on the injection of measurement packets in the network. Most existing tools are active, including Pathchar, Pathrate [DRM01a] or Capprobe. The second category consists of so-called embedded tools that alter the pattern of users traffic to estimate capacity without introducing additional measurement traffic. A typical example is TFRC Probe, which is an embedded version of CapProbe. The third category consists of purely passive tools that aim at extracting capacity estimates from passively collected traces of traffic. Three passive examples of capacity estimation tools are: PPrate, Nettimer and MultiQ. MutliQ is an element of the Click modular router [KMC00], while PPrate is integrated in the Intrabase analysis tool. PPrate, Nettimer and MultiQ are based on packet dispersion techniques and work with TCP connections.

Location	**IP Address**	**Upstream Capacity**	**Downstream Capacity**
		(Capacity Resolution)	**Capacity Resolution**
California	128.111.52.63	85.5 (5.1)	88.5 (5.0)
Russia	82.179.176.42	87 (7.2)	84 (4.3)
Australia	130.194.252.8	92 (6.0)	80 (4.7)

Table B.1: Path capacity measurements conducted from our receiver located in UCLA and the set of PlanetLab nodes used in our Internet experiments (Mbps).

While there are many tools proposed to estimate path capacity, we specifically chose Pathrate [DRM01a], since according to [LSB05b] it is the only capacity estimation tool that could successfully run and obtain estimates on PlanetLab which is our global testbed for evaluation of our detection methodologies.

Table B.1, uses the summary of path capacity measurements conducted from our receiver located in UCLA and the set of PlanetLab nodes used in our Internet experiments.

C Router Buffer Queues

Since our major proposed approach is loss-based, and specifically relies on saturating queues of the third party middlebox, it is important to understand how the internal buffers of routers work and where and how the packets are dropped. This understanding also constitutes our simulated middleboxes implemented in `ns-3` to more accurately simulate a typical modern router. In this chapter, we focus on the forwarding engine, particularly fast switching, input and output queues and scenarios in which they overflow.

C.1 A Router Internal Operation Overview

A modern router is a complex piece of equipment. Routers can be divided into three key components: a routing engine, a forwarding engine and a management agent. The function of the routing engine is to process routing information (exchanged between routers using a routing protocols such as the Border Gateway Protocol, BGP) so as to compute routes (using a shortest path algorithms) that are stored in routing information bases (RIB) and that are composed by a destination, a next-hop interface, and a metric. Routing entries are subsequently used to populate the forwarding information base (FIB) whose entries are used by the forwarding engine. The function of the forwarding engine is to transfer incoming traffic to an outgoing interface directed towards a router closer to the traffic destination by performing a longest match prefix lookup (or some other mechanism) using the incoming traffic destination address. This forwarding process is connectionless implying that at each hop the forwarding decision is taken independently for each datagram.

Received packets are processed by the link layer protocol controller, which handles the link layer protocol (e.g. Ethernet) used over the physical link. This also checks the received frame integrity (size, checksum, address, etc). Valid frames are converted to packets by removing the link layer header and are queued in the receive queue. This is usually a First-In-First-Out (FIFO) queue, often in the form of a ring of memory buffers. The buffers are passed into the input to the forwarding engine. This takes each buffer, one at a time, and removes it from the interface receiver. The packet is then forwarded to an appropriate output interface, corresponding to the best path to the destination specified in the destination address of the IP packet header. At the output interface, the packet (together with a new link layer header) is placed into a transmit queue until the link layer processor is ready to transmit the packet. This, like the receive queue, is a FIFO queue, and usually also takes the form of a ring of memory buffers. Each out-going packet requires a new link layer protocol header to be added (encapsulation) with the destination address set to the next system to the receive the packet. The link protocol controller also maintains the hardware address table associated with the interface. This usually involves using the Address Resolution Protocol (ARP) to find out the hardware MAC addresses of other computers or routers directly connected to the same cable (or LAN). The packet is finally sent using the media interface with the hardware address set to the next hop system. When complete, the buffer (memory) allocated to the frame, is freed, that is, it is returned as an empty buffer to the receive queue, where it may be used to store a new received packet.

C.2 Introduction to Forwarding

This section gives a simple description of the forwarding process. After determining the link layer frame is valid, the forwarding engine then starts processing the network layer information. It reads the network layer (IP) packet headers and checks various parts of the header, to ensure the packet is not damaged or illegal. It then uses a local Forwarding Table (known as the “Forwarding Information Base (FIB)”) to identify where in the network the packet should be routed to (i.e. which output interface should be used).

Once the appropriate output interface has been identified, the forwarding engine then requests the packet switch to form a connection to the appropriate output interface. The packet is then moved through the router to the output network interface controller. Although large routers actually implement a switch as a hardware component, most smaller routers do not actually contain a real switch. In other routers, the switch takes the form of a shared memory data structure in which all received packets are stored. The switching operation therefore consists of removing a pointer from the receive queue, and copying the value of the pointer to the appropriate transmit queue. In some cases, the entire packet data is copied from one bank of receive memory to another transmit memory using a computer bus.

The operation of the router is controlled by one or more general purpose processor which is usually similar to a standard high-end PC CPU. The processor’s performs various tasks, which may be divided into three groups: (1) Process Switching (2) Fast Switching (3) Routing.

C.3 Fast Switching

In many cases, a number of packets are sent by the same end system to the same destination IP address. Using process switching, each of these packets is handled independently — just as one would imagine for a connection-less protocol. But, this processing is costly when performed in this way. In fact, once one packet has been process switched, the router now understands the way to switch all successive packets to the same destination. That is the reason why, the process switching cached (or stored a copy of the outcome) the forwarding decision after it has been made.

Using the cached information (IP destination address, port number, link address, and any other necessary details), can significantly speed-up the forwarding by by-passing many decisions. This is known as the “Fast Path” or “Fast Packet Forwarding” and is outlined below: This scheme is much faster than process switching. However, the fast path may only be used for packets which have previously been sent to the same address. The first packet is therefore always process switched.

In practice, it is unwise to keep any cached information for too long. This prevents the information becoming stale (e.g. when a router fails). Cisco recommends a small part of the cache (e.g. 1/20th) is deleted every minute. Since it is very computationally expensive to find all the entries in the table referring to a single route of link layer address, the entire table is deleted (purged) whenever the routing table or an interface ARP table changes.

Fast switching is very effective at the edge of networks, or within private networks (where there are comparatively few destination addresses and routes). As the number of entries in the forwarding information base increases, the impact of purging the table becomes more and more significant. The rate of purges increases with the number of routers being communicated with. To help this, as the size of the forwarding table increases, the proportion of addresses may need to be deleted. Fast switching provides little advantage at the centre of the internet (core).

- The output queue is always used if there are any packets waiting there, this helps reduce the re-ordering of packets when packets for the same destination are being both process and fast switched.

- Some interfaces use a number of queues (from a few to several thousands). Normally one queue is reserved for network control data (such as routing packets) to ensure these are never delayed (in overload, such packets are particularly important since failure to receive them can impact the stability of the network). If a number of additional queues are being used by the interface, the packet is placed in the output queue, rather than the FIFO ring.

C.4 Input Queue Overflow

In this and the next section, we discuss in what scenarios buffer overflows happen in either of input or output queues.

When a packet enters the router, the router attempts to forward it at interrupt level. If a match cannot be found in an appropriate cache table, the packet is queued in the input queue of the incoming interface to be processed. Some packets are always processed, but with the appropriate configuration and in stable networks, the rate of processed packets must never congest the input queue. If the input queue is full, the packet is dropped. Each interface owns an input queue onto which incoming packets are placed to await processing by the Routing Processor (RP). Frequently, the rate of incoming packets placed on the input queue exceeds the rate at which the RP can process the packets. Each input queue has a size that indicates the maximum number of packets that can be placed on the queue. Once the input queue becomes full (the maximum number of packets is on queue), the interface drops additional incoming packets. The interface enters a throttling mode in which incoming packets are not accepted. The throttling period allows the RP to process the backlog of packets on the input queue. The input queue overflow scenario occurs most often when a higher speed interface feeds packets to a lower speed interface.

For Cisco routers the queue length value can be between 0 and 4096, while the default value is 75. Cisco recommends that for most other interfaces, queue length must not exceed 100. These are the conditions for input queue drop counter. They usually occur when the router receives bursty traffic and cannot handle all packets.

- The Rx FIFO which is accessible by the interface PHY and interface DMA is full and any new frames that arrive in this condition will be dropped (or overflows).
- The Rx ring which is accessible by the interface DMA and interface driver code is full. Any new frame transfers from the DMA cannot proceed with this condition, since there are no free entries in Rx ring and hence the frames sent are dropped (termed as overrun condition).

C.5 Output Queue Overflow

Each interface owns an output queue onto which the Routing Processor (RP) places outgoing packets to be sent on the interface. Sometimes the rate of outgoing packets placed on the output queue by the RP exceeds the rate at which the interface can send the packets. Each output queue has a size that indicates the maximum number of packets that may be held on the queue. Once the output queue becomes full (the max number of packets is on queue), the RP drops additional outgoing packets. The output queue overflow scenario occurs most often when the RP tries to send many packets at once.

For instance, assume a remote source-route bridging / Transmission Control Protocol (RSRB/TCP) local-ACK configuration[1]:

- The RP is responsible for flow control of the Logical Link Control, type 2 (LLC2) sessions.
- If the RP is local-ACKing 50 LLC2 sessions and the TCP pipe is suddenly closed, the RP sends disconnect requests (DISCs) for each LLC2 session.
- 50 DISCs are placed on the output queue of the output interface, but some may be dropped if the output queue overflows.

Output drops are caused by a congested interface. For example, the traffic rate on the outgoing interface cannot accept all packets that should be sent out.

[1] The examples used in this Chapter are borrowed from Cisco documents.

References

[141] Cisco Documents Document ID: 14156. *Understanding Data Compression.* `www.cisco.com/application/pdf/paws/9289/wan_compression_faq.pdf`.

[ACI99] M. Arai, A. Chiba, and K. Iwasaki. “Measurement and modeling of burst packet losses in internet end-to-end communications.” In *Dependable Computing, 1999. Proceedings. 1999 Pacific Rim International Symposium on*, pp. 260–267. IEEE, 1999.

[AHV98] Jonas Andren, Magnus Hilding, and Darryl Veitch. “Understanding end-to-end internet traffic dynamics.” In *Global Telecommunications Conference, 1998. GLOBECOM 1998. The Bridge to Global Integration. IEEE*, volume 2, pp. 1118–1122. IEEE, 1998.

[AL09] S. Agarwal and J.R. Lorch. “Matchmaking for online games and other latency-sensitive P2P systems.” In *ACM SIGCOMM Computer Communication Review*, volume 39, pp. 315–326. ACM, 2009.

[ATR10] Alexander Afanasyev, Neil Tilley, Peter Reiher, and Leonard Kleinrock. “Host-to-Host Congestion Control for TCP.” *Comm. Surveys Tutorials*, **12**(3):304–342, 2010.

[BB95] A. Bakre and BR Badrinath. “I-TCP: Indirect TCP for mobile hosts.” In *Distributed Computing Systems, 1995., Proceedings of the 15th International Conference on*, pp. 136–143. IEEE, 1995.

[BH01] J. Border and J. Heath. “RFC3051: IP Payload Compression Using ITU-T V. 44 Packet Method.” 2001.

[BKG01] J. BORDER, M. KOJO, J. GRINER, et al. “RFC3135.” *Performance enhancing proxies intended to mitigate link-related degradations*, 2001.

[Bol93] J.C. Bolot. "End-to-end packet delay and loss behavior in the Internet." *ACM SIGCOMM Computer Communication Review*, **23**(4):289–298, 1993.

[Bor00] M.S. Borella et al. "Measurement and interpretation of internet packet loss." 2000.

[BPK99] H. Balakrishnan, V.N. Padmanabhan, and R.H. Katz. "The effects of asymmetry on TCP performance." *Mobile Networks and Applications*, **4**(3):219–241, 1999.

[BS97] Kevin Brown and Suresh Singh. "M-TCP: TCP for mobile cellular networks." *SIGCOMM Comput. Commun. Rev.*, **27**(5):19–43, October 1997.

[BSU98] M.S. Borella, D. Swider, S. Uludag, and G.B. Brewster. "Internet packet loss: Measurement and implications for end-to-end QoS." In *Architectural and OS Support for Multimedia Applications/Flexible Communication Systems/Wireless Networks and Mobile Computing., 1998 Proceedings of the 1998 ICPP Workshops on*, pp. 3–12. IEEE, 1998.

[CB02] Brian Carpenter and Scott Brim. "Middleboxes: Taxonomy and issues." Technical report, RFC 3234, February, 2002.

[CCR03] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. "PlanetLab: an overlay testbed for broad-coverage services." *SIGCOMM Comput. Commun. Rev.*, **33**(3):3–12, July 2003.

[CJ99] S. Casner and V. Jacobson. "RFC2508: Compressing IP/UDP/RTP headers for low-speed serial links." *Internet Engineering Task Force. URL http://www. ietf. org/rfc/rfc2508)*, 1999.

[CLL02] Jacky C Chu, Kevin S Labonte, and Brian N Levine. "Availability and locality measurements of peer-to-peer file systems." In *ITCom 2002: The Convergence of Information Technologies and Communications*, pp. 310–321. International Society for Optics and Photonics, 2002.

[CMN06] W. Chen, U. Mitra, and M.J. Neely. "Packet Dropping Algorithms for Energy Savings." In *Information Theory, 2006 IEEE International Symposium on*, pp. 227–231. IEEE, 2006.

[CP14a] Robert Chang and Vahab Pournaghshband. "Implementation of Link-Layer Compression in Click and NS-3." Technical report, UCLA, 2014.

[CP14b] Robert Chang and Vahab Pournaghshband. "Implementation of Strict Priority Queuing in NS-3." Technical report, UCLA, 2014.

[Daw] S. Dawkins et al. "Internet Draft: Performance Implications of Link-Layer Characteristics: Slow Links, 1998.".

[DHB13] Gregory Detal, Benjamin Hesmans, Olivier Bonaventure, Yves Vanaubel, and Benoit Donnet. “Revealing Middlebox Interference with Tracebox.” In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC ’13, pp. 1–8, New York, NY, USA, 2013. ACM.

[DMG10] Marcel Dischinger, Massimiliano Marcon, Saikat Guha, P Krishna Gummadi, Ratul Mahajan, and Stefan Saroiu. “Glasnost: Enabling End Users to Detect Traffic Differentiation.” In *NSDI*, pp. 405–418, 2010.

[DMH] Marcel Dischinger, Alan Mislove, Andreas Haeberlen, and Krishna P. Gummadi. “Detecting BitTorrent Blocking.”.

[DRM01a] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. “What do packet dispersion techniques measure?” In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pp. 905–914. IEEE, 2001.

[DRM01b] Constantinos Dovrolis, Parameswaran Ramanathan, and David Moore. “What do packet dispersion techniques measure?” In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pp. 905–914. IEEE, 2001.

[Dsh95] Jewgeni H Dshalalow. *Advances in Queueing Theory, Methods, and Open Problems*, volume 4. CRC Press, 1995.

[ECB99] Mathias Engan, Steven Casner, Carsten Bormann, and T Koren. “RFC 2509: IP header compression over PPP.” Technical report, RFC 2509, February, 1999.

[FJ93] Sally Floyd and Van Jacobson. “Random early detection gateways for congestion avoidance.” *Networking, IEEE/ACM Transactions on*, **1**(4):397–413, 1993.

[FM98] R. Friend and R. Monsour. “RFC2395: IP payload compression using LZS.” 1998.

[FS96] R. Friend and W.A. Simpson. “RFC1974: PPP Stac LZS Compression Protocol.” 1996.

[Gao01] Lixin Gao. “On inferring autonomous system relationships in the Internet.” *IEEE/ACM Transactions on Networking (ToN)*, **9**(6):733–745, 2001.

[Hon08] M. Honan. “Inside Net Neutrality: Is your ISP filtering content?”, 2008.

[HP] HP Support FAQs. *Using Compression with HP Router Products*. `http:\/\/www.hp.com\/rnd\/support\/faqs\/pdf\/comp.pdf`.

[IA01] J. Ishac and M. Allman. "On the performance of TCP spoofing in satellite networks." In *Military Communications Conference, 2001. MILCOM 2001. Communications for Network-Centric Operations: Creating the Information Force. IEEE*, volume 1, pp. 700–704. IEEE, 2001.

[ipe] "Iperf: A measure network performance tool." `http://iperf.fr/`.

[Jac90] V Jacobson. "RFC 1144: Compressing TCP." *IP headers for low speed serial links*, 1990.

[JPS07] LE Jonsson, G Pelletier, and K Sandlund. "RFC 4995: The Robust Header Compression (ROHC) Framework." *Network Working Group*, pp. 1–40, 2007.

[JR86] R. Jain and S. Routhier. "Packet trains–measurements and a new model for computer network traffic." *Selected Areas in Communications, IEEE Journal on*, **4**(6):986–995, 1986.

[KD10a] Partha Kanuparthy and Constantine Dovrolis. "Diffprobe: Detecting ISP Service Discrimination." In *Proceedings of the 29th Conference on Information Communications*, INFOCOM'10, pp. 1649–1657, Piscataway, NJ, USA, 2010. IEEE Press.

[KD10b] Partha Kanuparthy and Constantine Dovrolis. "Diffprobe: Detecting ISP Service Discrimination." In *Proceedings of the 29th Conference on Information Communications*, INFOCOM'10, pp. 1649–1657, Piscataway, NJ, USA, 2010. IEEE Press.

[KD11] Partha Kanuparthy and Constantine Dovrolis. "ShaperProbe: end-to-end detection of ISP traffic shaping using active methods." In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pp. 473–482. ACM, 2011.

[Kes95] S. Keshav. "A control-theoretic approach to flow control." *ACM SIGCOMM Computer Communication Review*, **25**(1):188–201, 1995.

[KK03] Aleksandar Kuzmanovic and Edward W. Knightly. "Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants." In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, pp. 75–86, New York, NY, USA, 2003. ACM.

[KMC00] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. "The Click modular router." *ACM Transactions on Computer Systems (TOCS)*, **18**(3):263–297, 2000.

[KR09] E. Kline and P. Reiher. "Securing data through avoidance routing." In *Proceedings of the 2009 workshop on New security paradigms workshop*, pp. 115–124. ACM, 2009.

[LB00] Kevin Lai and Mary Baker. "Measuring link bandwidths using a deterministic model of packet delay." In *ACM SIGCOMM Computer Communication Review*, volume 30, pp. 283–294. ACM, 2000.

[LCB07] Guohan Lu, Yan Chen, Stefan Birrer, Fabián E Bustamante, Chi Yin Cheung, and Xing Li. "End-to-end inference of router packet forwarding priority." In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pp. 1784–1792. IEEE, 2007.

[LGS07] J. Ledlie, P. Gardner, and M. Seltzer. "Network coordinates in the wild." In *Proc. of NSDI*, volume 7, pp. 299–311, 2007.

[LSB05a] Sung-Ju Lee, Puneet Sharma, Sujata Banerjee, Sujoy Basu, and Rodrigo Fonseca. "Measuring bandwidth between planetlab nodes." In *Proceedings of the 6th international conference on Passive and Active Network Measurement*, PAM'05, pp. 292–305, Berlin, Heidelberg, 2005. Springer-Verlag.

[LSB05b] Sung-Ju Lee, Puneet Sharma, Sujata Banerjee, Sujoy Basu, and Rodrigo Fonseca. "Measuring bandwidth between planetlab nodes." pp. 292–305, 2005.

[LSG04] M. Luglio, M.Y. Sanadidi, M. Gerla, and J. Stepanek. "On-board satellite Split TCP proxy." *Selected Areas in Communications, IEEE Journal on*, **22**(2):362–370, 2004.

[MBG00] Bob Melander, Mats Bjorkman, and Per Gunningberg. "A new end-to-end probing and analysis method for estimating bandwidth bottlenecks." In *IEEE Global Telecommunications Conference*, pp. 415–420, 2000.

[MDH10] Danny McPherson, R Dobbins, M Hollyman, C Labovitzh, and J Nazario. "Worldwide infrastructure security report, Volume v, Arbor Networks." 2010.

[MLJ89] Steve McCanne, Craig Leres, and Van Jacobson. "Libpcap.", 1989.

[MSM09] A.T. Mzrak, S. Savage, and K. Marzullo. "Detecting malicious packet losses." *Parallel and Distributed Systems, IEEE Transactions on*, **20**(2):191–206, 2009.

[nok13] "Nokia hijacks mobile browser traffic, decrypts HTTPS data.", 2013.

[NR13] Hung X Nguyen and Matthew Roughan. "Rigorous statistical analysis of internet loss measurements." *IEEE/ACM Transactions on Networking (TON)*, **21**(3):734–745, 2013.

[ns3] "PlanetLab: An Open network platform." `http://www.planet-lab.org/`.

[PAR14] Vahab Pournaghshband, Alexander Afanasyev, and Peter L. Reiher. "End-to-End Detection of Compression of Traffic Flows by Intermediaries." In *IEEE/IFIP Network Operations and Management Symposium*, NOMS '14. IEEE, 2014.

[Pax04] Vern Paxson. "Strategies for sound internet measurement." In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, IMC '04, pp. 263–271, New York, NY, USA, 2004. ACM.

[PDM03] Ravi Prasad, Constantinos Dovrolis, Margaret Murray, and KC Claffy. "Bandwidth estimation: metrics, measurement techniques, and tools." *Network, IEEE*, **17**(6):27–35, 2003.

[Per98] R. Pereira. "RFC2394: IP Payload Compression Using DEFLATE." 1998.

[PFM14] Vahab Pournaghshband, Matin Fouladinejad, and Mahdi Moghaddam. "Taracom: End-to-End Detection of Third-Party Compression." Technical report, UCLA, 2014.

[PKR12] V. Pournaghshband, L. Kleinrock, P. Reiher, and A. Afanasyev. "Controlling Applications by Managing Network Characteristics." *Proceedings of IEEE ICC 2012 Communication and Information Systems Security Symposium*, June 2012.

[pla] "ns-3: An Open Simulation Environment." `http://www.nsnam.org/`.

[PSR12] V. Pournaghshband, M. Sarrafzadeh, and P. Reiher. "Securing Legacy Mobile Medical Devices." *Proceedings of the 3rd International Conference on Wireless Mobile Communication and Healthcare (MobiHealth)*, November 2012.

[R00] Segura R. "Asymmetric Networking Techniques For Hybrid Satellite Communications." *NATO Technical Note 810*, 2000.

[RAJ11] J. Rajahalme, S. Amante, S. Jiang, and B. Carpenter. "RFC6437: IPv6 flow label specification." 2011.

[Ran96] D. Rand. "RFC1978: PPP Predictor Compression Protocol." 1996.

[Rem] "Enterprise Network and Data Security Spending Shows Remarkable Resilience.".

[RWW04] Ramaswamy Ramaswamy, Ning Weng, and Tilman Wolf. "Characterizing network processing delay." In *Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE*, volume 3, pp. 1629–1634. IEEE, 2004.

[Sal04] David Salomon. *Data Compression.: The Complete Reference.* 2004.

[SBD05] J. Sommers, P. Barford, N. Duffield, and A. Ron. "Improving accuracy in end-to-end packet loss measurement." In *ACM SIGCOMM Computer Communication Review*, volume 35, pp. 157–168. ACM, 2005.

[SEA07] Taghrid Samak, Adel El-Atawy, and Ehab Al-Shaer. "Firecracker: A framework for inferring firewall policies using smart probing." In *Network Protocols, 2007. ICNP 2007. IEEE International Conference on*, pp. 294–303. IEEE, 2007.

[Sha86] M. Shapiro. “Structure and encapsulation in distributed computing systems: the Proxy principle.” In *The 6th International Conference on Distributed Computing Systems*, 1986.

[SHS12] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. “Making middleboxes someone else’s problem: network processing as a cloud service.” *ACM SIGCOMM Computer Communication Review*, **42**(4):13–24, 2012.

[SMP01] A. Shacham, B. Monsour, R. Pereira, M. Thomas, and I.P.P.C. Protocol. “RFC 3173.” *IP Payload Compression Protocol (IPComp)*, 2001.

[SNC04] Augustin Soule, Antonio Nucci, Rene Cruz, Emilio Leonardi, and Nina Taft. “How to identify and estimate the largest traffic matrix elements in a dynamic environment.” In *Proceedings of the joint international conference on measurement and modeling of computer systems*, SIGMETRICS ’04, pp. 73–84, New York, NY, USA, 2004. ACM.

[SPN13] Radu Stoenescu, Matei Popovici, Lorina Negreanu, and Costin Raiciu. “SymNet: static checking for stateful networks.” In *Proceedings of the 2013 workshop on Hot topics in middleboxes and network function virtualization*, pp. 31–36. ACM, 2013.

[SRA12] Justine Sherry, Sylvia Ratnasamy, and Justine Sherry At. “A Survey of Enterprise Middlebox Deployments.” 2012.

[SZC07] H. Song, S. Zhu, and G. Cao. “Attack-resilient time synchronization for wireless sensor networks.” *Ad Hoc Networks*, **5**(1):112–125, 2007.

[TMF09] Mukarram Bin Tariq, Murtaza Motiwala, Nick Feamster, and Mostafa Ammar. “Detecting Network Neutrality Violations with Causal Inference.” In *In Proc. of the CoNEXT Conference*, 2009.

[ura] “random(4) man page.” `http://man7.org/linux/man-pages/man4/random.4.html`.

[VR10] S. Vincent and J.I.J. Raja. “A survey of IP traceback mechanisms to overcome denial-of-service attacks.” In *Proceedings of the 12th international conference on Networking, VLSI and signal processing*, pp. 93–98. World Scientific and Engineering Academy and Society (WSEAS), 2010.

[WMW06] Feng Wang, Zhuoqing Morley Mao, Jia Wang, Lixin Gao, and Randy Bush. “A measurement study on the impact of routing events on end-to-end internet path performance.” In *ACM SIGCOMM Computer Communication Review*, volume 36, pp. 375–386. ACM, 2006.

[WQX11] Zhaoguang Wang, Zhiyun Qian, Qiang Xu, Zhuoqing Mao, and Ming Zhang. “An untold story of middleboxes in cellular networks.” *ACM SIGCOMM Computer Communication Review*, **41**(4):374–385, 2011.

[wre] "Technical Specification from Cisco, Distributed Weighted Random Early Detection.".

[YB94] R. Yavatkar and N. Bhagawat. "Improving end-to-end performance of TCP over mobile internetworks." In *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, pp. 146–152. IEEE, 1994.

[ZD01] Yin Zhang and Nick Duffield. "On the constancy of Internet path properties." In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pp. 197–211. ACM, 2001.

[zli] `http://www.zlib.net/`, Title = zlib Library.