Faculdade de Engenharia da Universidade do Porto

Network traffic sampling for improved signature and anomaly based intrusion detection

Jorge Amílcar Lopes Teixeira

A thesis submitted in partial fulfillment of the requirements for the Degree of Master in Electrical and Computers Engineering

Major in Telecommunications

Supervisor: José Ruela (Prof.)

Co-supervisor: Filipe Sousa (Eng.)

Co-supervisor: Filipe Abrantes (Eng.)

July 2008

U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MIEEC - MESTRADO INTEGRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES **2007/2008**

A Dissertação intitulada

"Network traffic sampling for improved signature and anomaly based intrusion detection"

foi aprovada em provas realizadas 16/Julho/2008

o júri

Presidente	Professor Doutor Pedro Alexandre Guimarães Lobo Ferreira do Souto Professor Auxiliar da Faculdade de Engenharia da Universidade do Porto [illegible]
	Professor Doutor André Ventura da Cruz Marnoto Zúquete Professor Auxiliar da Universidade de Aveiro [illegible]
	Professor Doutor José António Ruela Simões Fernandes Professor Associado da Faculdade de Engenharia da Universidade do Porto [illegible]

O autor declara que a presente dissertação (ou relatório de projecto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extractos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são correctamente citados.

Autor - Jorge Amilcar Lopes Teixeira

Jorge Amilcar Lopes Teixeira

Faculdade de Engenharia da Universidade do Porto

Abstract

Deploying Intrusion Detection Systems (IDS) inside a company or institution's network is becoming a common practice these days to prevent or mitigate the possibly devastating effects of worms, Distributed Denial-of-Service (DDoS) or any other attacks launched against them.

To improve its response time and allow for better communication and cooperation between network security experts about eminent or ongoing attacks it has been suggested to move IDS to the core of the network - the Internet backbone infrastructure. However, current IDS solutions are hardly scalable and cannot sustain the high volume levels of network traffic in realtime.

The implementation of sampling mechanisms has been regarded as a possible solution to that problem, but investigation is still ongoing about which sampling methods and which detection metrics provide the best compromises in terms of improved performance at multi Gigabit speeds while having low error rates at detecting attacks. This thesis addresses these and other aspects of high speed intrusion detection, namely using the two most common approaches: signature detection and anomaly detection.

The study is initiated with a survey of the current state of the art and enhanced with the creation and development of a framework specifically aimed at simplifying the comparison of different sampling methods and detection algorithms. It is also designed to allow the quick introduction of new methods and more complex algorithms.

Regarding anomaly detection, the preliminary tests show that different sampling methods affect detection in a similar way and that conservative sampling rates provide good results when using the entropy of the destination port as main metric. However, more aggressive sampling rates are not suitable for use with this metric and would need more complex algorithms.

Signature detection is not compatible with sampling, however, as its effectiveness drops sharply even with very modest sampling rates.

Acknowledgements

I would like to thank my supervisors for giving me the freedom to select the directions of my research while always maintaining a critic yet helpful spirit. Their suggestions also helped to overcome the time constraints and to finish this project with meaningful results.

Preface

Researching Intrusion Detection Systems (IDS) was my first choice from the long list of topics available for my thesis. I picked it for two main reasons: it was something new to me and it required mastery of various fields such as network monitoring, data mining/statistics collection, computationally efficient algorithms and even network element's hardware. Being a hot and exciting topic also ensured that my interest on it would remain high for the entire duration.

Despite being a very vast research area that can seem pretty daunting to someone who is just starting, I quickly focused my attention into the scalability problem because it provided a good balance between theory vs. practice and was suitable to the available time frame of one semester. Being in a Telecommunications Major myself and having concluded that the best modern detection metrics are the ones based on Information Theory concepts such as entropy also contributed to my convictions that my choice of topic could not have been more appropriate.

The somewhat short duration of the thesis work was its only downside, which left a bittersweet taste of "mission accomplished" and "plenty of room for improvement". Hopefully, others will also pick it up and contribute to make all our computer networks safer.

The Author,

Porto and FEUP, June 30th 2008

Contents

List of Figures

List of Tables

Acronyms

ACK	Acknowledge
AH	Authentication Header
API	Application Programming Interface
ARP	Address Resolution Protocol
AS	Autonomous System
BGP	Border Gateway Protocol
BPF	Berkeley Packet Filter
BSD	Berkeley Software Distribution
DMZ	De-Militarized Zone
DoS	Denial of Service
DPI	Deep Packet Inspection
ESP	Encapsulating Security Payload
FIN	Final
FPGA	Field-Programmable Gate Array
HIDS	Host-based Intrusion Detection System(s)
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection Systems
IETF	Internet Engineering Task Force
IP	Internet Protocol

IPFIX	IP Flow Information eXport
IPsec	IP security
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
IRC	Internet Relay Chat
ISP	Internet Service Provider
LAN	Local Area Network
NIC	Network Interface Card
NIDS	Network-based Intrusion Detection System(s)
O-D	Origin - Destination
OS	Operating System
P2P	Peer to Peer
PSAMP	Packet SAMPling
QoS	Quality of Service
RST	Reset
RTT	Round Trip Time
SCTP	Stream Control Transmission Protocol
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
SYN	Synchronize
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WAN	Wide Area Network

Glossary

anomaly detection
: method for detecting intrusions by identifying unusual characteristics

attack
: *see* **intrusion**

botnet
: set of compromised autonomous zombie hosts, despite being collectively remote controlled by a single entity

elephant flow
: extremely large (in total bytes) continuous flow measured over a network link

entropy
: measure of uncertainty or information content of a process

false negative
: type II error, i.e., failure to report an actual attack against a resource by considering it authorized use

false positive
: type I error, i.e., report of an attack against a resource when it is, in fact, authorized use

flash crowd
: effect caused by a sudden surge in the number of users requesting a certain resource (typically a website) due to it being recently large-scale advertised

flow
: set of packets sharing common properties, e.g. the 5-tuple srcIP, srcport, destIP, destport and protocol

intrusion
: any set of actions violating the access policy to a given resource

Intrusion Detection System
: computer system aimed at detecting, reporting and eventually responding to ongoing or imminent intrusions

mouse flow
: small (in total bytes) continuous flow measured over a network link

packet fragmentation
: act of breaking up a single IP datagram into two or more

rootkit
: program designed to take fundamental control of a computer system, without authorization by the system's owners and legitimate managers, that actively tries to conceal itself from detection

signature detection
: method for detecting intrusions using preset patterns that define known attacks as comparison

trojan
: piece of malware which appears to perform a certain action but in fact performs another such as transmitting a computer virus

Chapter 1

Introduction

1.1 Overview

The ability to communicate is paramount to human societies. Therefore, having reliable communication channels that do not distort, remove or append information while properly delivering it to its rightful destination has always been sought after. Computer networks have been developed to help fulfill these communicating needs and, following the trend of recent years, societies are becoming more and more dependent on them.

While the errors inherent to the non-ideal physical properties of materials have been mitigated by using advanced modulations and error-correcting codes, other sources of disruption gained relevance: unauthorized access and data tampering. Organizations of different size and nature make frequent use of resources available in their internal networks as well as in the agglomerate of inter-networked systems commonly denominated Internet and requirements of confidentiality, data integrity and availability are often threatened by intrusions that range from mild pranks to serious computer crimes involving large sums of money and/or sensitive information.

There are several security systems available to prevent and defend from said attacks, like firewalls (perimeter defense) or antivirus software (host defense), but they operate by reducing functionality, i.e., blocking ports or disallowing the execution of programs from unknown sources, which tends to contradict the open nature of most networks. Plus they can provide a false sense of security: what if a programming error in the firewall allows for an exploit or a brand new virus is set loose? These and other attacks may originate not only from external entities but also from within the organization, such as when users maliciously attempt to escalate their privileges or inadvertently bring in a worm-infected laptop. All those concerns led to an increase in popularity of Intrusion Detection Systems (IDS), whose role is to detect and alert whenever security appears to be compromised, so that appropriate action can be taken against the specific attacks, either by a system administrator or automated response agent.

Ideally, IDS would disregard *all* authorized uses of the monitored resources while missing *none* of the intrusions. The actual rates of false positives and false negatives are used as metrics for evaluating the efficacy of IDS and typically decrease with the increase in complexity of the detection engine. Sometimes, securing every host independently and as tightly as possible even at the expense of large overheads is considered acceptable (e.g., military databases), but a more typical scenario would involve monitoring just the network links in a more efficient manner. The crux of IDS has shifted from trying to obtain *minimal* error rates to maintaining *acceptable* error rates but not impacting the usability or performance of the monitored systems, at all. While high volumes of data are difficult to deal with in most situations, the largest obstacle in Network-based Intrusion Detection Systems (NIDS) design is the rate at which information accumulates, requiring architectures and algorithms capable of sustaining very high throughput. However, detection engines usually mean either long pattern-matching procedures that do not scale well or summarization and statistics-based methods that scale well but exhibit higher error rates. For the high-speed links in the network core, this latter solution is the only one implementable.

Some of the advantages of detecting intrusions in the core vs. at the fringe include early warnings on new threats, locating (and even blacklisting) the sources of attacks and preventing the core itself of collapsing when a major worm outbreak or massive Denial of Service (DoS) attack takes place. So, these large installations require scalable solutions, and it is unrealistic to consider that a backend Internet Service Provider (ISP) could thoroughly analyze *every single packet* being sent and received in every link it shares with its clients and peers while maintaining Quality of Service (QoS) and honoring the Service Level Agreements (SLAs). Therefore, aggregates and summaries of network traffic as reported by routers have been used to identify some intrusions, but the intrinsic loss of information leads to higher error rates.

The use of sampling can be both a complement and an alternative to such summaries, since they are generated from subsets of traffic and can be sampled themselves for further data reduction. Aiming at improving the error rates, different sampling strategies are being investigated in order to reduce the impact on various existing and novel IDS using diverse detection metrics that identify a broad scope of intrusions. Some of these sampling techniques are borrowed from the Internet measurement and analysis field but further modifications are being pursued to improve the overall effectiveness.

1.2 Objectives

The main goal for this thesis is to study and evaluate the effects of sampling in IDS and whether such technique could possibly be applied in the nearby future to the detection of anomalies in high-speed links, such as the ones connecting ISPs. To this effect, factors such as different sampling methods, detection algorithm complexity and different traffic patterns should be considered.

The first objective is to gather enough information about the state of the art of IDS and detailed knowledge about how they work and how can they be improved with sampling to perform better in high-speed links. This includes research about which traffic features are more resistant to sampling and how to perform the sampling efficiently.

To allow for the necessary testing, a prototype framework should be developed that can simplify the comparison of sampling methods and allow for the introduction of new detection algorithms. This should allow to establish some relationships between sample rates and errors they cause in metrics, which could conduct to false positives/false negatives.

There should also be some research about which types of intrusions can actually be detected and which cannot, as well as about complementary solutions.

Finally, the effects of sampling in signature detection should also be investigated for comparison sake.

1.3 Document structure

Past this introduction, the second chapter presents a primer on IDS including their historical background, basic concepts and architectures, followed by relevant aspects of common intrusions. It also contains a section about network traffic sampling, namely its current applications and most common methods.

In the third chapter, a survey of the state of the art follows, identifying related work and key references in those areas (analysis of tested sampling strategies and most promising network traffic features for anomaly detection) and briefly summarizing their most important conclusions. Some of these papers are broader by describing overall architectures and providing general directions for future work while others are very specific and focus on a single problem/solution. Together, they formed the rationale behind the following chapter.

The fourth chapter regards the actual implementation of a prototype framework to evaluate the impact of different sampling methods using various intrusion detection metrics. The architecture and internals of the system are explained, as well as the prominent implementation details.

The obtained results and respective analysis are reported in the fifth chapter. Details about the experimental conditions are also described.

Finally, the conclusions drawn upon the interpretation of the experimental data and future directions for research end this thesis.

Chapter 2

Background

2.1 Primer on Intrusion Detection Systems

In order to follow up this thesis, a basic knowledge of IDS and related concepts is required; therefore, a short primer is presented but can be safely skipped if the reader is already familiar with the area.

2.1.1 Origins and history

Initial works on the ability to detect unauthorized access to computing resources date back to 1980, when James Anderson published his now classic report [2] describing the possibility to find intruders based on the fact that unauthorized activities are inherently unusual (i.e., anomalous) and therefore would "stand out" from the typical usage patterns. A good example is his definition of the *masquerader* - someone who managed to obtain (steal) the credentials of another authorized user, therefore becoming an authorized user himself - and the audit information useful to detect his presence, such as the usage outside of normal time or the abnormal volume of data accessed. This *masquerader* was one of the user classes capable of performing internal attacks, as opposed to external penetration (much more difficult at that time when networking was still under heavy development).

Therefore, while Anderson's work introduced Host-based Intrusion Detection Systems (HIDS), it was Dorothy Denning who in 1987 presented a network-centered model [20] where audit trails were analyzed and matched against different user profiles, thereby setting the groundworks for future Network-based Intrusion Detection Systems (NIDS) development; in fact, most of Denning's work came from her involvement in the first functional IDS solution called Intrusion Detection Expert System (IDES) from SRI International [19]. Almost simultaneously, the Haystack project [48] used a novel approach where the audit data was compared against a set of predetermined patterns (or rule sets). In 1990, the Network Security Monitor (NSM) [28] became the first true implementation of a NIDS and its success and wide deployment greatly increased the industry's interest on these technologies. This was mostly visible around 1999, to the point USENIX even held a workshop [51] on the topic. Also relevant are the appearances of Bro [44] and Snort [46], two open-source implementations that defined the modern standard for IDS.

Meanwhile, the evolution of IDS and their growing complexity led to the publication of more detailed surveys [42, 4, 47] and/or taxonomies [18, 3], in order to classify and evaluate different approaches and solutions. They are particularly useful for a summarized view on the progress of the state of the art and for the wealth of their references. Much of the description of concepts presented in later sections was highly based on these sources.

2.1.2 Basic concepts

This sections contains brief definitions of key concepts pertaining to IDS, so that the relationships between them become evident under the context of this thesis. It does not intend to be a replacement for the established taxonomies.

The monitored *computer system* is a combination of hardware and software that provides *resources* (typically services) to *users*. A *user* needs not to be an actual person, it can be an agent instead. Which *users* have access to which *resources* is defined by the *access policy*, so whenever this policy is violated it becomes *unauthorized use* and constitutes an *intrusion* . The most common subset of *intrusions* typically stem from either malice or neglect of *users* and can compromise further access to *resources* by other *users* due to affecting its *confidentiality*, *integrity* or *availability*.

Under this generalization, *users* who engage in *intrusions* are also called *attackers* and tend to fall into one of four categories:

- *"script kiddies"* - jargon term for poorly skilled *attackers* who employ prepackaged exploit scripts targeted at known vulnerabilities.
- *viruses/worms* - self-replicating (and possibly self-modifying) programs containing a payload that performs the actual damage. A virus usually propagates only within the infected host (requiring human assistance to infect new machines), while the worm targets other hosts within network range.
- *botnet* - swarm of hosts who have been previously compromised (bots) and are controlled by an entity known as botmaster. More danger-

ous than viruses and worms because the botmaster can easily and frequently change the attack pattern.

- *hackers* - technically skilled persons, sometimes acting in group. They are the most dangerous since they supersede all previous categories.

Lastly, these *intrusions* can either come from the inside of the *computer system* or from any other connected network, as well as being perpetrated by *users* who already had elevated access rights.

Therefore, *Intrusion Detection Systems (IDS)* are ought to detect ongoing or imminent *intrusions* and alert a *security administrator* (possibly triggering some automatic counter-measures as well). An analogy can be made by comparing IDS to house alarms: upon detecting a security breach, the siren sounds and alerts the neighbors, who should take action and call the Authorities. Obviously, one IDS alone is not a full security system, it should be used as a complement (back to the analogy, the house should have solid doors and windows - otherwise the robber can get in so easily and so quickly he might not even care that the alarm is triggered).

2.1.3 Architectures

Information sources

Traditionally, the most common way to classify IDS was by their information source: Network-based Intrusion Detection Systems (NIDS) or Host-based Intrusion Detection Systems (HIDS). Nowadays, some current implementations are hybrid to some extent, trying to get the best of both worlds. NIDS detect attacks by monitoring the network links of a given system, usually at vantage points such as the De-Militarized Zone (DMZ) or main gateways. Typically, several hosts form a distributed array of probes that locally analyze traffic and report intrusions to a centralized main management console. HIDS use local audit trails, resource usage and system or error logs, so details about users and processes involved in intrusions can be obtained. They can also report their findings to a central management location.

If properly placed, a few probes from a NIDS can monitor a large network, hence being cost-effective; these probes are mainly passive elements, thus causing no noticeable overhead. Other advantages include the easiness of retrofitting existing infrastructures, their near stealth mode of operation (making NIDS hard to detect and subvert) and their intrinsic ability to overview the whole protected system.

The main disadvantage of using NIDS is the requirement of very efficient packet scanning (to reduce false negatives), which is hard to maintain on par with the growth of bandwidth and Internet usage levels causing unprecedented amounts of network traffic. The other significant disadvantage is that due to the increase of sophistication in evasion techniques [26, 38] and rise of encrypted traffic levels, some attacks can be hidden from NIDS.

Since HIDS are placed directly on the host, some intrusions that elude NIDS can still be detected, thus they exhibit a lower rate of false positives [34]. They can also detect other types of attacks such as buffer overflows and may monitor some of the encrypted traffic just before it gets encrypted or just after being decrypted. Additionally, the audit trails provide the means to detect inconsistencies in process execution typical of trojan and/rootkit activity.

The major downside of HIDS is that they need to be deployed in every host, thus its management is more complex and costly than NIDS. The significant overhead on every monitored machine must also be taken into account, especially in the case of busy servers.

Detection methods

In order to detect intrusions, two methods known as signature detection and anomaly detection are used. The former is computationally less demanding but restricted to previously known attack patterns while the latter has a higher false alarm rate due to changes in user behavior.

A signature is the sequence of actions characteristic of a specific intrusion. When running, the IDS compares current activity to its database containing all known signatures (these should be updated periodically to reflect new attacks). Whenever a match is found, an alarm is raised and the security administrator warned. Regarding known attacks, this method is very accurate and therefore it is the one found in the majority of currently deployed IDS [34]. However, the obvious downfall is that a small alteration on the attack pattern renders the IDS useless, not to mention brand new attacks. Another side-effect is that any authorized use that matches a signature will be incorrectly deemed an attack, causing a false positive.

The IDS with learning capabilities have the advantage of detecting new attacks by regarding them as anomalous behavior. This is achieved by providing an initial training period (the system should be devoid of attacks) and consider that as a baseline of normal use. Later on, whenever the system begins to act strangely, the IDS can signal an alert. There are two major inconveniences with this method, the first being the difficulty of implementing good learning algorithms and guaranteeing that no attacks are ongoing during the training period; the other concerns the less frequent accesses - since they did not occur during the training period (because they are rare), the IDS incurs in false positives. This anomaly based approach is also ineffective whenever the usage patterns vary greatly.

Most of the development in this area is centered around data-mining techniques (to select the features of the traffic that more explicitly indicate a change) and continuously learning models (based, for example, in artificial neural networks [24, 37]).

Information flow

Simplifying, an IDS is composed of three modules that can coexist in a single machine but are typically distributed: probing agents that collect audit information, detection core engines and monitoring stations (sometimes with internal actuators if automated responses are available, otherwise they just output alarm notifications to trigger a manned response by an administrator). A representation can be found on Figure 2.1.

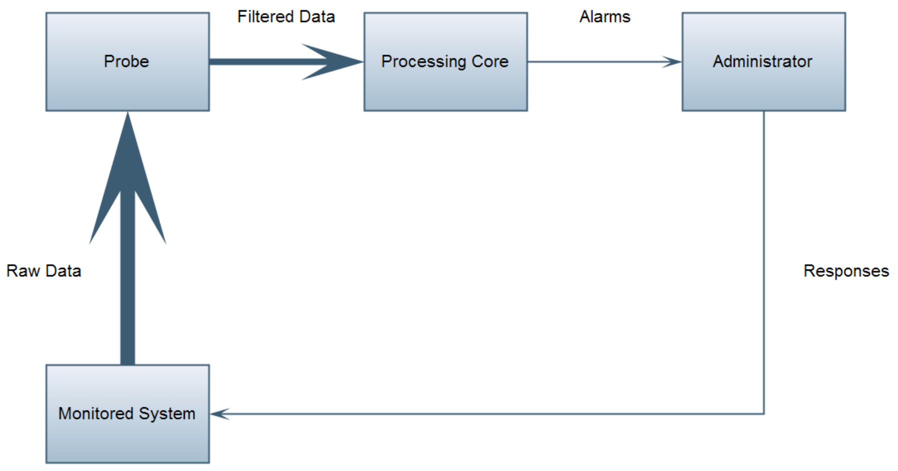


Figure 2.1: Information flow in an Intrusion Detection System

As previously stated, IDS can exist either in hosts or in the network itself. This second strategy has achieved greater popularity due to its lower cost and simpler deploying in large organizations (cf. deployment in a few strategic places - border routers, firewalls, etc. - vs. monitoring every single host). It then makes sense to research and develop strategies that can further reduce the overall workload and streamline the data propagation process. Considering the prohibitive amount of traffic currently found circulating in networks, one trivial method to enhance scalability of IDS is to perform some sort of filtering (data reduction) upon the initial data collection - this is where network traffic sampling appears as a potentially viable solution.

2.1.4 Common intrusions

The vast amount of network intrusions is typically categorized into three major types: scanning, DoS and penetration. All these can be initiated from the inside as well as from the outside of the local network.

The scanning of the network is typically the first step toward a larger attack and is characterized by sending different types of packets to various destinations in order to probe the existing systems for their topology, types of allowed traffic, active hosts, software running on said hosts and corresponding version numbers. This leads to a more precise identification of the most vulnerable targets for future attack. On the other hand, scanning the network is no more than browsing publicly available resources, so the illegality and immorality of such actions is disputed. Modern IDS can differentiate between benign scanning (looking for a webpage, shared repositories, etc.) and vulnerability assessment scanning. Scanning is a very common form of attack (despite not causing damage per se) and a recent survey can be found in [1], where problems such as probe location and the metrics used to define malicious behavior are discussed. It should be noted that the authors acknowledged the problem of analyzing huge logs spanning over twelve years and their approach was simply to use the data of every 30th day (a trivial method of systematic sampling).

As for DoS, its objective is to disrupt the access to resources by third parties, typically by slowing down the response time by an overflow of requests. These are extremely important in the areas of online commerce, where every second of unreachability provokes loss of income. Sometimes, vulnerabilities in the systems allow for exploitations rather than flooding, as exemplified by the "ping of death" where a very large (illegally formed) packet caused havoc in various systems in the 1990s until proper patches came out. DoS is especially effective (hard to counter) when it is performed by several distributed hosts, normally under the form of a botnet. It can also mimic a flash crowd making its detection even harder but solutions exist, as shown in [32].

A penetration occurs whenever data or privilege tampering is achieved, making this type of attack possibly devastating. Unlike DoS, the system is not simply brought down to its knees, it is actively running, except now under the influence of a rogue agent. The most common way to gain control of a system is by means of software design flaws, allowing the attacker to either escalate its privileges (typically to root) or at least to be granted read/write access on sensitive files. Sometimes this can be done in two steps: using public access to exploit a flaw and be granted local user access and then escalating privileges to root, therefore taking complete control of the system, which is dangerous in two ways: all other hosts on the network are probably unaware that this particular host is compromised, so they will continue to trust it (thus increasing the chances of becoming compromised themselves) and finally because the compromised system can launch an attack to other external systems, with all the legal implications.

Worthy of mention is the current expansion of the botnet threat, as described by [45]. They are still mostly controlled by Internet Relay Chat (IRC) scripts - to which detection methods exist; see [6] - though some variations using Peer to Peer (P2P) systems have emerged (see [53]). Due to their mutable nature, they can perform all the previous attacks with great impunity, since they are hard to track and even when identified, not much can be done other than warning the respective system administrator that some of its equipments are being used to conduct attacks.

2.2 Network traffic sampling

With the advent of high-speed networks, the traditional monitoring systems were no longer capable of keeping up with the intense traffic and sampling techniques were researched to reduce the distortion of the measured indicators (typically packet counts, byte counts, flow counts, etc.). This section contains the most common techniques and how they compare among themselves.

2.2.1 Packet sampling methodologies

In essence, the purpose of sampling is to extract specific characteristics about a parent population at a lower cost than a full census would require. When the sustained arrival rate of new packets surpasses the processing speed of the monitoring equipment, sampling also helps by replacing the uncontrolled discarding of packets with a controlled selection process (introducing a controlled amount of error).

The characteristics relevant for IDS are different from those required by traffic analysts, but the methodology used is both simple and easily adaptable, thus being a good starting point to achieve quick but no-nonsense results. Common sense dictates that increasing the numbers of samples tends to give better results, but with every sample comes a cost in terms of CPU time, memory, etc., that may just not be worth it and must be carefully weighted to find an optimal solution, though the definition of “good enough accuracy" will vary with every network administrator and the specific goal for collecting that data in first place.

While not related to IDS, a study about different sampling methods is described in [17], where time and event-driven triggers determine the sampled packets with different granularities, with the goal of determining how they affect two common metrics: distribution of packet sizes and interarrival times. By that time (1993), it was already evident that measuring traffic characteristics of a certain full trace interval was unfeasible in many situations and that sampling presented itself as the obvious approach to estimating such characteristics. To evaluate the performance and thus suitability of different sampling strategies, the authors compared the distributions of the original population to those obtained by sampling using techniques from statistical processing.

The three main sampling methods used on that study are those depicted on Figure 2.2, namely systematic sampling, stratified random sampling and simple random sampling. Each of them can be implemented using triggers based on time or events (number of packets) and at different granularities (sampling rates). Finally, it is also very important to take into account the duration of the sampling interval vs. the original unsampled data (an interval itself, but assumed to be the parent population) as some comparison techniques are sensitive to this.

Systematic sampling consists of picking every k^{th} packet of the data set (i.e., the first of each bucket of size k, while stratified random sampling is selecting a random packet out of every bucket. Simple random sampling is, as the name implies, a uniform random selection of n packets from the entire set.

Packet selection is triggered when the associated counter depletes or timer expires, depending whether the sampling is packet-driven or time-driven. When using a timer, the first packet arriving past its expiration is sampled. Since it is assumed that the link is active, i.e., there are always some packets transversing it, this necessary approximation causes no significant impact.

If the packets (population) were randomly ordered, all methods should be equivalent. Systematic sampling does spread the samples more evenly among the population, so it may provide more precise results than stratified random. In fact, systematic sampling is more precise than simple random sampling if the variance within the systematic samples is larger than the population variance as a whole. But if correlation exists between pairs of elements within the systematic sample, then stratified or simple random sampling will be better. For populations with a linear trend, stratified random sampling would provide the best precision.

Their results show that time-driven sampling is worse than event-driven

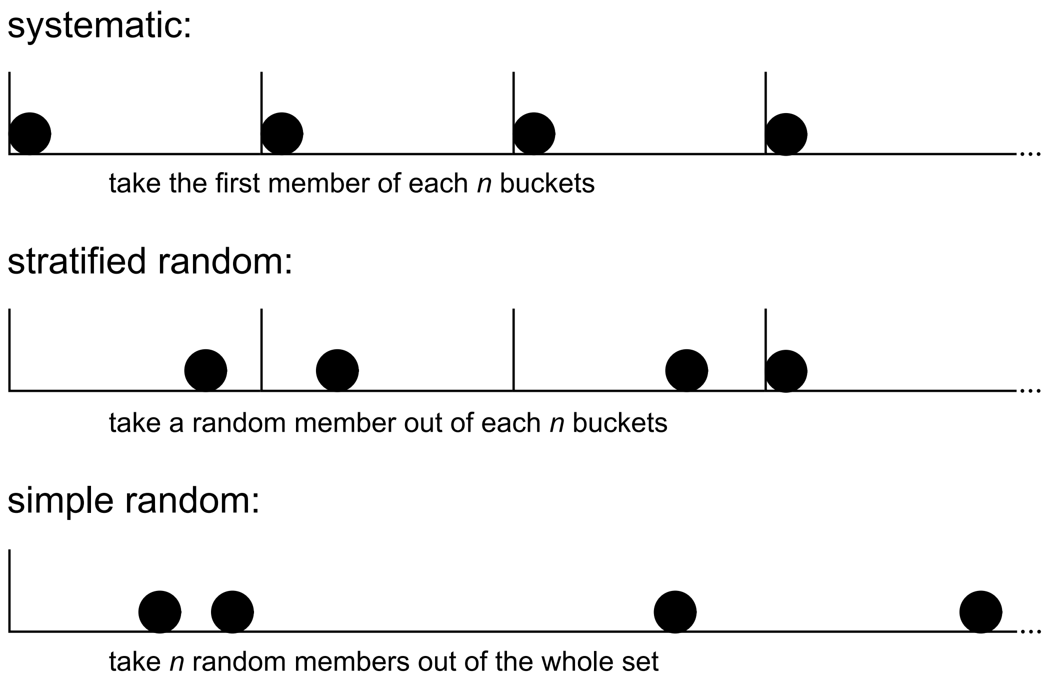

systematic:
take the first member of each n buckets
stratified random:
take a random member out of each n buckets
simple random:
take n random members out of the whole set

Figure 2.2: Abstract representation of three sampling methods

sampling and that little difference exists between classes of sampling (systematic, stratified random or simple random). The poor performance of time-driven sampling can be explained by the bursty nature of network traffic; sampling in the buckets where higher bursts occur will miss more packets than on other buckets with less traffic. Regarding granularity, no conclusive answer is given, though the results show the expected behavior that decreasing the sampling fraction yields poorer results. As for the interval of the observation, again the predictions were correct: longer ones obtain better results.

In summary, that study is aged and the data used may no longer represent the information flow found on modern Local Area Network (LAN) and Wide Area Network (WAN), but the methodologies are still valid and an important foundation for the initial stages of studying how to measure the performance of different sampling methods for use in IDS.

The same methods are analyzed in [56] but this time applied to the measure of packet delay in a video stream. Other than more articulate definitions of the previous three sampling methods and a few caveats that should be prevented (such as provisioning for the possibility of no packets arriving within a time bucket or the overhead of generating a random number in the cases of simple and stratified random), not much is added in terms of methodology. However, the published results are quite interesting as they show the superiority of random sampling vs. systematic sampling due to the latter's high sensitivity to serial correlation (present in the video stream). It is also shown that, if a priori information is available, it can be used in the subgroup division step of stratified sampling to produce even more accurate results. A related study [57], this time using online game data, reflects the same conclusion: stratified random sampling provides the best results; but they also note that the differences between methods are small.

Summarizing, and because the targets of this thesis are high-speed backbone links, it is expected to obtain very similar results with any of these sampling methods, because the aggregate of traffic present in the monitored segment comes from different sources and gets arbitrarily mixed, thus loosing much of any potential correlation. Also, because of the high volume of packets, the implementation cost (complexity and overhead) of each method will have considerable weight; the advantage sits with the systematic sampling that requires only a counter or a timer while the others require a random number generator and, in the case of stratified random sampling, a multi-step algorithm.

Chapter 3

State of the art

There is plenty of literature involving data reduction methods to efficiency improve IDS, but few focused specifically on traffic sampling. In this chapter, relevant articles from the areas of intrusion detection and network traffic measurement are cited and their conclusions summarized in order to provide a starting point for more research and experimentation. It is worth noting that distinct approaches about how to improve IDS for the monitoring of high-speed links have been considered over the years, but none came out openly as "victorious" nor was deemed "obsolete". Therefore, it makes sense that this state of the art contains not only the directly related work but also the (viable) alternatives allowing the comparison of advantages and shortcomings.

3.1 Sampling in NIDS

3.1.1 The need for sampling

With the steady increase of the bandwidth on most network links, it has become difficult to efficiently monitor it for attacks. Unlike with the analogy of the burglar alarm (where *any* movement is a sign of intrusion), this time there is plenty of benign activity - so much that it becomes nearly impossible to monitor *all* the activity (and thus pinpoint the actual attacks).

It is true that summary statistics such as the ones reported by SNMP and NetFlow (see section 3.1.3) could be used as pre-reduced data sources, but even generating them from *all* traffic is hard to accomplish; therefore, sampling was introduced in most network traffic collecting systems. Derived from its use there, applications of sampling in IDS are being proposed and investigated in literature.

3.1.2 Packet capturing

In order to be analyzed, a packet must first be captured. The performance of this action is highly dependent on a number of factors, namely the Network Interface Card (NIC) type and the Operating System (OS) being used. In fact, a more precise answer would include the number of interrupts generated by the NIC, the current load of the machine and the number of context switches between kernel/user-space and the respective amount of bytes transfered.

Interrupts are triggered by the hardware and can be reduced if the NIC implements a queuing mechanism that allows it to transfer multiple packets to the kernel in a single batch, instead of one by one as they arrive on the wire. There is some loss of accuracy regarding the arrival timestamp, but in IDS that is irrelevant. Context switches can also be minimized by copying a bundle of packets in the kernel into userspace instead of separately, this time with no penalty other than a processing delay (again irrelevant). The amount of bytes to copy can be reduced by disregarding the packet payload partial or totally because most information can be inferred from the header; a remarkable exception is when the system is suppose to perform Deep Packet Inspection (DPI), which requires the full payload to be most effective. In this case, it would be convenient to process the packet directly in the kernel.

Each packet needs to be dissected, analyzed and accounted for the statistics, so there is a tradeoff between the complexity of the detection algorithm, the size of the buffers in the NIC/kernel and the link's nominal rate. Sampling relaxes the number of packets analyzed per interval but the detection algorithm must use metrics that are relatively unaffected by that sampling method in order for the system to be effective.

The actual capture can be done by dedicated equipment monitoring a specific segment (uses all the available resources, which may included specialized hardware such as Field-Programmable Gate Array (FPGA) chips) or implemented as secondary functions directly on the router elements. Each approach has its strengths and weaknesses:

Stand alone device

+ Uses all available resources
+ Independent of existing equipments
– Typically expensive
– Requires a monitoring port or even an optical splitter if the environment is switched

Additional function on router

+ Re-uses existing equipment
– Available resources are highly dependent on traffic load
– May impact the primary function of packet routing

In [29], the authors used the first approach to develop a monitoring system for use at multi Gigabit speeds. In fact, they aggregate the packets into flows, but they actually sample every packet, recording some of its properties in the mentioned flow record for later analysis. While this semi-offline method does not seem adequate for IDS, the fact that capturing *all* packets under certain circumstances is possible, warrants a more detailed study on the available hardware capabilities and suggests reflection toward the design of new IDS architectures.

The system was already implemented at an ISP but for slightly lower speeds. It consisted on an optical splitter copying packets to a specialized DAG packet capture card from Endace[22] connected to a PC. They describe in detail the technical difficulties in capturing packets at higher speeds, both using stand alone equipment and in-router capabilities, concluding that the stand alone solution is more effective.

Their reasoning behind all this is simple: on average, a packet arrives every 240ns (assuming 300 byte packets) to a 10Gbit/s interface; that time equates to a certain number of instructions/memory accesses that the processor can do in between packets. If the analysis can't be done in such short period (most likely the case with IDS), sampling can be introduced so that the window is extended to accommodate all the required processing. This approach is not tested by the authors, as their only processing is registering the timestamp, flow lookup and logging some of the Internet Protocol (IP) and Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) fields, which they manage to perform within the alloted interval. Of course, for higher speeds, sampling becomes strictly necessary.

They also point out the issues associated with identifying flow termination and concluded that a static inactivity timeout would be adequate. Based on different traces, a timeout of 60s provided the best tradeoff between accuracy and amount of memory required to maintain state.

3.1.3 Flow measurement

An alternative to deploying probe hosts to perform network intrusion detection is to use the readily available flow information present in most recent routers such as Cisco's NetFlow[15], recently standardized by the Internet Engineering Task Force (IETF) as IP Flow Information eXport (IPFIX).

NetFlow

The name NetFlow can be used to describe the mechanism that collects and caches the flow information or the network protocol used to transmit that information elsewhere. In this section, and unless specifically noticed, the former meaning is used.

NetFlow is a tool provided by Cisco as part of their IOS Software running on network elements such as routers or switches to measure and report flow traffic passing through them. A brief analysis based on the documentation available online ([11, 14, 13, 12, 16, 10]) follows.

"The Cisco IOS NetFlow feature set allows for the tracking of individual IP flows as they are received at a Cisco router or switching device. NetFlow identifies packet flows for both ingress and egress IP packets. It does not involve any connection-setup protocol, either between routers or to any other networking device or end station. NetFlow does not require any change externally either to the packets themselves or to any networking device. NetFlow is completely transparent to the existing network, including end stations and application software and network devices like LAN switches. Also, NetFlow capture and export are performed independently on each internetworking device; NetFlow need not be operational on each router in the network."

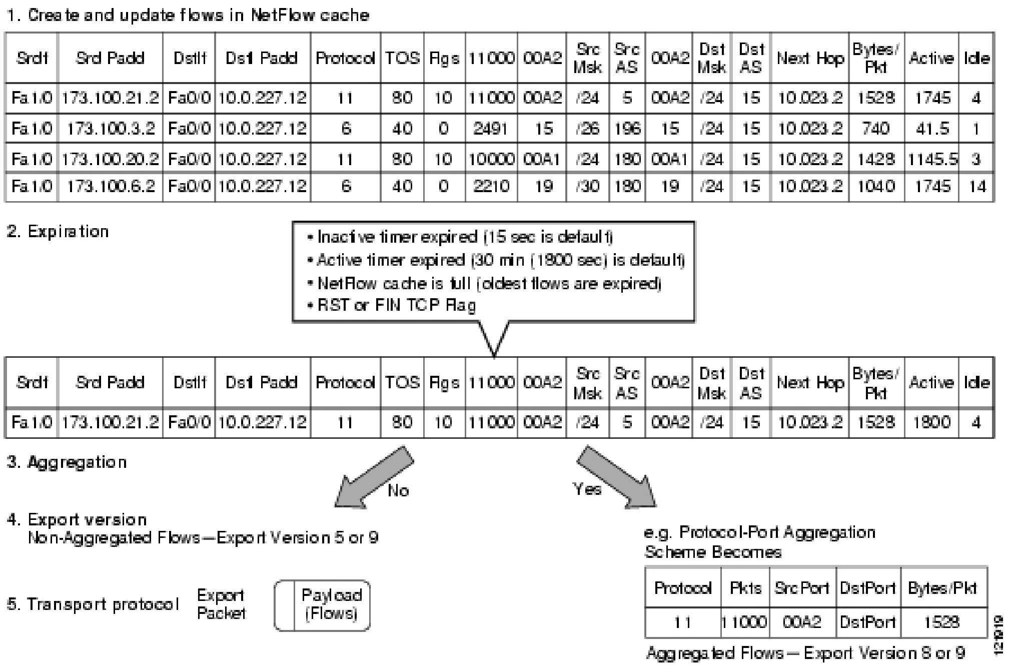
NetFlow was developed by Cisco to aid network administrators in the tasks of accounting, billing, network planning, traffic engineering, and user or application monitoring by incorporating into their lines of network equipment some low-overhead statistics modules and devising a protocol to transmit that data over the network for further analysis. It is an improvement over Simple Network Management Protocol (SNMP) because it keeps state of flows, allowing for more granular understanding on the distribution of bandwidth and finding out which IP addresses are sourcing or receiving such traffic and using which protocols.

The concept of flows allows for more scalability as large amounts of data get condensed into a couple of flows present in NetFlow's cache, a local high-performace database. This data can be accessed through the terminal interface, but most commonly it gets sent automatically to a reporting server designated NetFlow Collector, which has the job to analyze the exported flows and produce comprehensible reports by means of data combination. This automated export occurs whenever flows are considered to have terminated (either explicitly by means of a Final (FIN) in TCP or after a timeout period): a couple of terminated flows are normally bundled together and sent by UDP (more recently also by Stream Control Transmission Protocol (SCTP)) to the Collector.

This decision on when to export is important for time synchronization purposes, so the conditions for flow termination should be clearly known:

- explicit termination: TCP FIN or Reset (RST)
- inactivity: default 15 seconds
- long-livedness: default 30 minutes
- memory full: terminate oldest flows

One timer purges the cache of broken/unreliable connections and the other prevents staleness of very long connections (these split flows can be recombined in the Collector to obtain the total data transfer, for example). A simple overview of NetFlow can be see on Figure 3.1.


1. Create and update flows in NetFlow cache
2. Expiration
3. Aggregation
No
Yes
4. Export version
Non-Aggregated Flows—Export Version 5 or 9
e.g. Protocol-Port Aggregation Scheme Becomes
5. Transport protocol
Export Packet
Payload (Flows)
Aggregated Flows—Export Version 8 or 9
121919

SrcIf	SrcIPadd	DstIf	DstIPadd	Protocol	TOS	Flgs	11000	00A2	Src Msk	Src AS	00A2	Dst Msk	Dst AS	Next Hop	Bytes/Pkt	Active	Idle
Fa1/0	173.100.21.2	Fa0/0	10.0.227.12	11	80	10	11000	00A2	/24	5	00A2	/24	15	10.023.2	1528	1745	4
Fa1/0	173.100.3.2	Fa0/0	10.0.227.12	6	40	0	2491	15	/26	196	15	/24	15	10.023.2	740	41.5	1
Fa1/0	173.100.20.2	Fa0/0	10.0.227.12	11	80	10	10000	00A1	/24	180	00A1	/24	15	10.023.2	1428	1145.5	3
Fa1/0	173.100.6.2	Fa0/0	10.0.227.12	6	40	0	2210	19	/30	180	19	/24	15	10.023.2	1040	1745	14

- Inactive timer expired (15 sec is default)
- Active timer expired (30 min (1800 sec) is default)
- NetFlow cache is full (oldest flows are expired)
- RST or FIN TCP Flag

SrcIf	SrcIPadd	DstIf	DstIPadd	Protocol	TOS	Flgs	11000	00A2	Src Msk	Src AS	00A2	Dst Msk	Dst AS	Next Hop	Bytes/Pkt	Active	Idle
Fa1/0	173.100.21.2	Fa0/0	10.0.227.12	11	80	10	11000	00A2	/24	5	00A2	/24	15	10.023.2	1528	1800	4

Protocol	Pkts	SrcPort	DstPort	Bytes/Pkt
11	11000	00A2	DstPort	1528

Figure 3.1: Simple NetFlow overview

As for the NetFlow Export format, several versions spawned among the years of development, being the most up-to-date version 9, which was used as origin by IETF to design the industry standard IPFIX. There is numerous software, closed and even open-source (like http://www.ntop.org), that understands these export formats. An overview of the major versions used follows:

- Version 5: added Border Gateway Protocol (BGP) Autonomous System (AS) information and flow sequence numbers.
- Version 8: added aggregation caches data export (allows for exporting datagrams containing a subset of the usual Version 5 export data, if that data is valid for a particular aggregation cache scheme).
- Version 9: a flexible and extensible format, which provides the versatility needed for support of new fields and record types. This format accommodates new NetFlow-supported technologies such as multicast, Multiprotocol Label Switching (MPLS), and BGP next hop. The distinguishing feature of the NetFlow Version 9 format is that it is template based. Templates provide a means of extending the record format, a feature that should allow future enhancements to NetFlow services without requiring concurrent changes to the basic flow-record format. IPFIX was based on the Version 9 export format.

Being an embedded measuring tool, it is capable of fast operation, but obviously has limits, and it was tested for different products of different product lines in [16] to account for the overhead required vs. the normal routing operations. The following trend was observed: increasing the number of flows also increases CPU usage - which makes sense since more flows will be present in cache and thus more processing power will be required to sort packets through them. Another relevant observation concerns the different NetFlow features used (such as BGP next hop or enabling multiple destinations or even Version number) - they did not cause much impact on the CPU usage levels, so they should not be a decisive factor. Finally, the equipments suited with lesser powerful processors were the ones where the overhead was more noticeable.

Also subject to test is the (Random) Sampled NetFlow feature that implements random packet sampling, i.e., each packet has a fixed 1-in-n (n>1) probability of being sampled and thus inserted in the cache (or having the corresponding counters updated if it belongs to an existing flow). Cisco reports that using 1-in-100 samples, approximately 80% of the flows will be created in cache, which saves on CPU use and export traffic volumes (at the expense of having “lossy” data). The decrease in CPU use due to sampling is impressive for certain equipments, to the point Cisco openly advises in favor of using it in order to improve switching times. All the previous conclusions are also valid for Sampled NetFlow (an increase of flows provokes an increase of CPU usage, etc.).

NetFlow's ability to sample packets was first provided by a feature named Sampled NetFlow. The methodology that the Sampled NetFlow feature uses is deterministic sampling, which selects every k^{th} packet for NetFlow processing. For example, if the sampling rate is set 1-in-100 packets, then Sampled NetFlow samples the 1^{st}, 101^{st}, 201^{st}, 301^{st}, and so on packets. Sampled NetFlow does not allow random sampling and thus can make statistics inaccurate when traffic arrives in fixed patterns. Random Sampled NetFlow is more statistically accurate than Sampled NetFlow (though the nature of busy, highly-multiplexed backbones could probably make any possible difference insignificant).

NetFlow also has some basic IDS capabilities as shown in [14], with additional Layer-2 and Layer-3 information being exported as well as pre-filtered data (subset of NetFlow traffic aggregated by class). It is once again stressed the importance of Random NetFlow Sampling because it can be used to reduce the impact on the router using NetFlow to monitor traffic that might be a network threat, such as a DoS attack.

Packet SAMPling (PSAMP) is the perfect complement to IPFIX in terms of sampling framework and it is a work in progress by IETF whose draft can be seen in [21]. That document describes a standard set of capabilities for network elements to sample packets and report on them. This is poised to allow for network-based management and control of PSAMP equipped devices of different natures and vendors using standardized sampling schemes (the ones previously presented plus a few content-based ones).

The framework is made up of three main processes, namely: the packet selection algorithm, the creation and export of measurement reports and the protocol format for transmitting said reports. This means that PSAMP can serve not only local and remote collectors but even on-board measurement or IDS applications.

The introduction of Sampled NetFlow allowed for additional scalability and in [23] a proposal for improving it using adaptive sampling rates depending on traffic conditions was presented - an interesting feature to use with IDS: the normal usage would be to capture as many packets as possible to detect faint attack, but during massive DoS it would be wise to reduce the rate to avoid overflowing (and thus loosing other potential attack hidden amidst them).

The paper addresses the issues of sampling rate adaptation (native Sampled NetFlow is a case of systematic sampling) and flow entry renormalization (ability to gracefully degrade the accuracy level of the reported data) as well as implementing fixed time bins (native Sampled NetFlow may span to other time bins making analysis much more complex) and accurate flow counting (native Sampled NetFlow does not perform well with non-TCP flows).

The authors begin by pointing out that errors introduced by lost NetFlow packets (due to peaks of traffic intensity) are worse than those introduced by intentional sampling, so it is difficult for the network administrator to set a rigid sampling rate that provides enough accuracy during low/medium traffic periods but is also resistant to massive DoS attack. The heuristics used for flow termination are also incompatible with most network analysis tools, which are based on the concept of bins - fixed-length time periods - so the solution is to create a binned model for Sampled NetFlow as well. The problem with the lack of bins is that flows that overlap bins will get split (causing errors in counting). Also, to correctly report the increases on the number of flows typical of worms and scans, it was also necessary to implement reliable counters for flow aggregates, requiring additional protocol information not available with NetFlow.

The results presented confirm that the solution is effective and does not introduce additional bias, with the advantage of being easily deployed with a software update except the accurate flow counter that requires small hardware modifications. For "obvious" attacks causing a lot of traffic, this solution provides superior cost-benefit relationships, that requires no additional probes and the traffic statistics would be useful anyway for purposes other than intrusion detection.

A brief overview of the benefits and downfalls of each observation level is shown in Table 3.1:

Table 3.1: Different observation levels

Data Source	Description	Advantages	Disadvantages
Packet	lowest granularity; all raw packets with all fields	most detailed data and statistics; easier to obtain	scales poorly; protocols must be decoded
Flow	5-tuple + possible extras	scalable; uniform field format	might lack fields
IDS	alerts of different formats	tunable	resources intensive; false positives; false negatives

On per-flow state

In order to assess whether per-flow state is really required, one can look up to [38] where proofs regarding the detection of some types of attacks are devised. The motivation provided is that pushing IDS toward the center of the network (ISPs) would help mitigate the effects felt on all its clients, but to do so requires a big stretch in scalability. It is clearly indicated in that paper that most IDS implementations rely on per-flow state (maintaining information about every connection) which inhibits wire-speed analysis (the typical solution is to use load-splitters and various IDS). In that paper, a separation between ingress and egress packets (packets coming in and coming out) is exhibited because that can directly affect some of the assumptions made to characterize the analyzed attacks: TCP SYN flooding, port scans, TCP connection hijacking and payloads distributed among fragmented packets.

A TCP Synchronize (SYN) flood occurs when an attacker sends various SYN packets signaling the start of TCP connections, to which the victim responds with Acknowledge (ACK)+SYN and allocates resources to maintain the connection, but the attacker never sends the final ACK to complete the synchronization process.

The victim will quickly exhaust the maximum number of simultaneous half-open connections and stop responding to legitimate requests by third parties.

It is proved that detecting SYN floods through ingress packets does require per-flow state, or more practically, that detection requires the flood to be a sizable portion of all SYN traffic.

The best approach is then to randomly sample SYN packets and wait a reasonable Round Trip Time (RTT) for the ACK - if the unmatched SYN count rises, most likely it is a flood. A measure using the egress traffic is much more efficient: a simple comparison of SYN and FIN packets would account for all unclosed connections - but this required the internal network to be trustworthy - if the attacker had an insider accomplice, the latter could easily forge FIN packets.

As for port scans (horizontal: looking for the same service across various hosts or vertical: several services on the same host), it is also proved that per-flow state is required, and concluded that a simpler indicator such as a sudden increase in the number of target hosts/ports is not viable enough (e.g., possible route change). Again using egress traffic allows for much simplified detection: the TCP specification indicates that a RST packet should be sent whenever a port does not have a listening process running, or an ICMP "port unreacheable" response when using UDP, so an increase in this type of packets is a reliable indicator of port scans.

Session hijacking is the attempt by an attacker to inject crafted TCP packets into an existing connection by attempting to guess the right ongoing sequence number. This is seen by the receiver as an increase of the out-of-order packets, and this attack requires per-flow state as well. If egress data is available, such increase in the number of ACKs (which are the response to out-of-order packets: one ACK with the expected sequence number) would be a clear indication of an hijacking attempt.

Packet fragmentation is a known tool used by attackers to conceal malicious payloads that could be easily detected by IDS. A similar mechanism used in TCP called segmentation is analogous. To detect these packets, per-flow state is required and the NIDS must contain a normalization component. A simple workaround is to drop packets with small fragments, as there are few legitimate reasons to use them, and therefore larger fragments will typically hold enough of the payload to allow for identification. On the other hand, TCP segments can be very small and diluted among the stream, making them near impossible to detect without per-flow state.

The conclusions presented were always for the lower bound (or worst case scenario), so per-flow state may not be needed in all occasions. Also, the assumptions related to the network are crucial to determine if the proposed scalable alternatives will work, strengthening the idea that specific solutions for each problem will do fairly better than any general compromise.

Another paper voicing the need for scalable intrusion detection is [33], where the authors affirm that while some attacks cannot currently be detected without keeping per-flow state, several others can, even low traffic ones through the use of Partial Completion Filters which are a sort of hybrid solution between using signatures and statistical anomalies and can be implemented efficiently in FPGAs. They extract key fields from packet headers and they use different hashes to compare as their counts cross above certain thresholds. They are particularly effective at detecting SYN floods and portscans. Their tests on links with up to 40 Gbit/s showed good detection rates even for short-lived attacks and low false positives.

3.1.4 Impact of sampling

Sampling has been used with success in traffic measurement and various analysis and studies were made to ensure that the sampling methods do not add too much bias, but the application of sampled data to IDS needs more research. It is very important to know which metrics are of interest, from packet counts to flow counts or even distributions, in order to craft a sampling scheme appropriate to the problem.

To that effect, the authors in [39] have performed a series of tests intending to assess whether different sampling strategies can be used to detect volume anomalies such as DoS or portscans. This study also advocates the advantages of detecting intrusions inside the network near the ISP and thus high-speed monitoring is an absolute must. To that purpose, four sampling methods were tested: random packet sampling, flow sampling, smart sampling and sample-and-hold sampling. Traffic traces from Internet backbones were submitted to these sampling methods and then passed to IDS for detection of volume changes and portscans, consequences of DoS and/or flash crowds and worms and/or viruses, respectively. It should be noted that the selection of the algorithms present in the IDS was made taking into consideration the varied metrics used, so that a more general conclusion could be made.

Random packet sampling (behavior of Cisco's NetFlow random sampling) has the simplest implementation consisting on sampling a packet with a small probability. The packet is then classified in a flow. Random flow sampling initially classifies packets into flows as above then samples them with a small probability. More advanced techniques such as smart sampling have been devised to achieve lower variance rates while maintaining the sample size small; smart sampling takes flow size into account and was targeting the total byte count estimator. Finally, sample and hold sampling performs a look up for each packet to see if a flow entry already exists; if it does, the entry gets updated - if not, a new entry is created with a small probability that depends on the flow size.

The results show that random flow sampling achieves the best detection ratio on volume changes because it is the only one unbiased by flow sizes (packet sampling may miss small flows entirely), however, it still requires much more resources than packet sampling. The more advanced smart and sample and hold sampling methods achieve the worse results. Also, the decrease of the sampling rate also degrades the detection performance. Similarly, more portscans are sucessfully detected with random flow sampling, followed by packet sampling. Smart and sample-and-hold sampling are again inappropriate. The main downfall of packet sampling, however, is the increase of false positives due to flow shortening (the sampled packets provide bad estimates for the full flow length).

Another study ([8]) provides additional information concerning whether sampling, despite being a lossy process, still retains enough characteristics of the original data so that attacks can be detected. The paper begins by reiterating that most sampling studies are not focused on IDS and that random packet sampling, while simple and computationally efficient, is still biased. Splitting from other proposals, the approach presented focus not on the performance of a given set of anomaly detection methods, but on the impact of sampling in detection metrics. This allows for much broader conclusions: if a set of anomaly detection methods rely, for example, in the count of flows and it can be shown that said metric (count of flows) is degraded by sampling, then none of those anomaly detection methods will provide good results.

The metrics under investigation are the traditional number of bytes, packets and flows and a more recent feature entropy based one (entropy is a measure of how random a data-set is) which takes the packet or flow source IP as input. This investigation of entropy based anomaly detection stems from two previous studies ([36, 52]) whose preliminary results pointed out that these metrics were both better at detecting a larger number of high-traffic intrusions and more resilient to sampling, therefore being adequate for early-warning systems probing for new worm outbreaks of portscans. Back to [8], the methodology shown consists in recreating packet level traces from flow traces (because acquiring such detailed traces is difficult) obtained from real data containing a worm outbreak and then performing the different sampling rates. In the end, a comparison was drawn between the different metrics when subject to the different sampling rates.

Their findings corroborated the original results of ([36, 52]), as entropy proved to be a good metric for detecting the worm even at high sampling rates. In fact, this worm was undetectable using typical sampled packet counts. Other important result is the robustness of entropy vs. flow count regarding metric sensitivity variation due to anomaly intensity, meaning that even with fewer attacks entropy still allowed for the identification of the worm. All these conclusions are very promising and strengthen the idea that it is possible to devise high-speed IDS for at least some of these large-scale attacks.

A follow up can be found in [30], where a more definitive answer to the question "what is the maximum sampling rate that still allows for detection of a 10Kpps anomaly amidst normal traffic with baseline of 200Kpps?" is pursued. In fact, only packet-count based anomalies are addressed, but the goal of obtaining the relationships between the size of the traffic of the anomalies, regular traffic and optimal sampling rate is quite appealing.

Simply put, anomaly detection relies on finding differences between a baseline of what is normal and the set which contains the intrusions, so if sampling affects all traffic (normal and attacks) equally, detection becomes independent of sampling. The problem is that while mean rates of normal and anomaly traffic decrease linearly as sampling rates increases, variance does not decrease as fast for small sampling rates. This is due to sampling having a distortive effect on normal traffic, actively increasing its relative variance, thus making abnormal traffic harder to detect.

Equations for the false positive and false negative rates are indeed derived for random sampling, as well as the minimum granularities required.

While the tests were simulated (using real data), the conclusions seem valid: sampling rates lower than 1/100 give poor results, though it depends on how much variability did the normal original traffic had. However, with increases in granularity (the granularity in this context is how many samples of the determined type we can obtain in the time frame) the detection rate goes up, even with sampling. The problem is that some additional classification of packets is needed (for example, restricting to UDP).

3.2 Anomaly detection

3.2.1 Applications in IDS

Traditionally, network administrators relied on their experience and cunning to detect and identify abnormalities in the data presented by their monitoring systems. Some tools allowed certain thresholds to be defined, and upon crossing them an alert would be issued, but this is just a permutation of the initial problem because it is still up to the network administrator to somehow determine the appropriate thresholds and then to investigate the alert and try to interpret it. This dependence on the human factor is undesirable for two main reasons: requires highly qualified personnel with great knowledge of that particular system and looses efficiency as the network traffic grows and more events require his attention.

This ability to identify the typical values of the monitored variables and to establish reasonable assumptions about operating limits in real-time[1], adjusting them as the usage patterns change is the ultimate goal for IDS with anomaly detection capabilities. Such systems should be able to pinpoint all anomalous behavior and identify its cause, making adjustments to the network as needed, freeing the network administrator to perform supervision and administration duties.

Using network monitoring tools just to detect traffic volume changes is an ineffective approach because legitimate traffic can be very bursty and variable; besides, the network utilization patterns are continually changing. This is where anomaly detection systems shine by taking into account the behavior of the network and adapting themselves accordingly, being particularly effective at detecting DoS attacks and worm outbreaks. They provide an additional layer of security on top of the more traditional signature detection solutions.

There are two basic approaches to detecting anomalies: one using flow data as reported by most network elements such as routers and the other

[1]in this context, real-time refers to intervals of seconds or even a few minutes

using the packet data collected directly from the wire. Both are actively researched and each has its benefits and pitfalls.

3.2.2 Flow-based anomaly detection

Flow records are computed in network elements and then sent to a collecting station that uses that data as basis for analysis. This may put some stress on the infrastructure if the flow information is substantial, as it has to be transmitted over the network. It can also fail to reach the collector if an attack targets the router itself or causes congestion in the segment connecting to the collector.

Since flows are summaries of traffic, their analysis is much simpler than packets and they comprise much smaller amounts of information, often speeding its processing. However, that is also the most important flaw: the packets are no longer available for a more detailed analysis; there is a loss of granularity.

In [5], the authors propose that flow-level data is sufficient to expose a significant number of network anomalies. They cluster those anomalies in three groups: network operation anomalies, flash crowd anomalies and network abuse anomalies; their intention is to analyze and describe each type of anomaly rigorously, hoping that each will have some invariant characteristics. While most previous work focused on detecting deviations from a baseline assumed to represent normal use of the network, this approach targets specific statistical anomalies.

Network operation anomalies include physical link disconnections, hardware failures, service failures and network reorganization/reengineering. They are best distinguished by sudden and sharp changes in the bitrate followed by a stable bitrate that can be at a different level.

Flash crowd anomalies occur when there is a sudden rush of traffic to a particular service or content, usually sparked by mass media exposure, intentional or not. It is distinguished by a quick rise of the flow count for a particular service or destination with a gradual drop off over time.

Network abuse is a catch-all name for DoS, probes and portscans. These anomalies are typically not as obvious as the previous ones, but tend to appear as high flow counts with diverse source/destination addresses and ports.

They suggest simple statistics, time series analysis and wavelet analysis as methods to characterize the said anomalies and NetFlow as the tool to provide the raw data.

Another study [35] suggests using flow information from several points in the network and applying the subspace method to them. This method consists in retrieving the main temporal trends of each flow and decomposing them in the most common (normal) and less common (anomalous) eigenflows.

Their tests were conducted using sampled packets (1% of all traffic was selected using the uniform random method) that were aggregated into flows and binned in 5 minute intervals. While their method did not provide an automated classifier, the visualizations allow for intuitive manual classification, which yielded good results in terms of the variety of anomalies detected and low amount of false positives.

There are, however, some issues concerning flow-level anomaly detection that must be sorted, as described in [7], where the virtues (detecting a wide variety of DoS, worms or portscans) and faults (detecting benign changes like the introduction of a new application protocol, change of user behavior) are remembered and a methodology to aid distinguish them provided. Basically, it consists in determining good benchmarking tools that allow for correct characterization of malicious and benign traffic.

The idea is to collect real traffic considered to be "background" and then artificially inject anomalies to compare the differences. The main challenges are then to ensure simplicity and realistic behavior of the anomaly models, ensure that no artifacts are inserted by the injection method and that the resulting traffic is suitable to undergo different tests with different metrics. While the details of the framework are beyond the scope of this small review, the concept of having a flexible testbench capable of simulating the effects of varied intrusions in flow traces is surely interesting in the path to enlightenment about identifying the most prominent characteristics of such intrusions.

3.2.3 Packet-based anomaly detection

These solutions typically require specific hardware to capture the packets, often with the assistance of span ports in network devices to mirror traffic or using taps directly in the links.

Their mode of operation consist in capturing the packets traversing the link and keeping a variety of statistics that are constantly monitored for anomalies. They also have the ability to store or send batches of suspicious packets for further analysis.

Compared to flow-based methods, packet-based ones typically have greater potential because they access a larger amount of information and can extract, for example, signatures from new attacks and pass them on to firewalls. But the feasibility of realtime creation of said signatures is uncertain. Also, having access to the payload data may help later on to identify the source of the attacks or to use that information to create a security patch for that exploit.

Another advantage of packet-based anomaly detection is that it causes no overhead on network elements, allowing them to fulfill their primary role of switching or routing packets without interference.

The main disadvantage is the need for dedicated equipment, especially for high-speed scenarios. Scalability is also an issue, though sampling helps relieving it at the cost of reduced accuracy, as previously mentioned.

The metrics used by the detection algorithms vary according to the different implementations but tend to be based around entropy or directly related concepts. In [25], the authors use a pre-computed baseline distribution (obtained from packet data containing labeled attacks) of several classes or dimensions, each using a combination of traffic features. Then they apply an algorithm derived form the concept of relative entropy to detect the actual anomalies.

Therefore, the whole process is comprised of two-steps, the first of which is done offline and consists in finding the best combinations of features for each class by using what is described as maximum entropy estimation. The classes represent the probability space and represent possible protocols and port ranges; the objective is to find a set of functions and weights than when applied to the model distribution minimize the difference toward the empirical distribution.

Once the baseline is set, the packet stream is divided into slots of fixed duration and the relative entropy calculated, indicating how much does the traffic deviates from the expected behavior (baseline). By using thresholds and a sliding window mechanism it is possible to issue alerts when the anomalies are detected repeatedly over a time period. Their results show that the method is viable but the false positive rate is not as small as it should.

A different design, suited for deployment in multiple locations is presented in [55]. It is still based on entropy, but uses more efficient algorithms so that it can be implemented in commodity computers, while having the interesting "intersection measurable property", which means that it is possible to obtain the entropy of a stream passing in both point A and point B from the intersection of independent sketches collected at A and B.

This greatly simplifies the estimation of entropy in large traffic matrices, by simply selecting the ingress point A and egress point B and using the entropy at those points to calculate the entropy of the link between them.

The algorithm is quite complex but a number of approximations make it quite efficient. The results show reduced errors is estimating the entropy, but no application example for anomaly detection is given.

3.3 Signature detection

Since signature detection methods imply comparing every arriving packet to every attack in the database, they are inherently not scalable. Sampling will probably not help either because all signatures are either contained in a single packet (and most packets are discarded) or depend on a specific sequence of packets, which will never be seen by the detection engine because at least some packets of the sequence will be missing.

Nevertheless, they are powerful tools and studying them will certainly bring some added value and allow for a comparison against sampling with anomaly detection.

Snort

A popular open-source implementation of a NIDS is Snort[50]. It has evolved greatly and became a complete solution, relying on efficient articulation between modules to maximize performance.

Snort is an open-source (initially labeled as lightweight but now full-featured) NIDS that is widely deployed and has a modular architecture suitable to become a good testbench for characterizing the differences between different sampling methods. Alongside with the original paper ([46]) updated documentation is available online ([50]).

Snort does not require a great deal of resources, thus allowing Snort sensors to be deployed on practically any host. Being cross-platform, it is possible to use the same product and configuration sets consistently. Snort's ruleset language is fairly compact and easy to learn, especially compared to many other NIDS languages. This allows for simple modifications to existing rulesets or easy creation of locally maintained custom rulesets.

Snort uses the popular `pcap` library, the same library that `tcpdump` uses to perform its packet sniffing. Snort decodes all the packets passing by on the network to which it is attached by entering promiscuous mode. Recent addons include snort-inline which interacts with `iptables` based firewalls, changing those rules as required whenever attacks are found. In fact, Snort can run in the following modes:

- Sniffer mode, which simply reads the packets off the network and displays them in a continuous stream on the console (screen).
- Packet Logger mode, which logs the packets to disk.
- IDS mode, the most complex and configurable mode, which allows Snort to analyze network traffic for matches against a user-defined rule set and performs several actions based upon what it finds out.

- Inline mode, which obtains packets from `iptables` instead of from `pcap` and then causes `iptables` to drop or pass packets based on Snort rules that use inline-specific rule types.

Unlike a firewall, which is configured to allow or deny access to a particular service or host based on a set of access control rules (regardless of what the packet contains), NIDS captures and inspects all traffic, regardless of whether it is permitted or not. Based on the contents, at either the IP or application level, an alert is generated. Snort can perform real-time packet logging, protocol analysis, and content searching/matching. It can be used to detect a variety of attacks and probes such as stealth port scans, CGI-based attacks, Address Resolution Protocol (ARP) spoofing, and attacks on daemons with known weaknesses. Snort utilizes descriptive rules to determine which traffic it should monitor and a modularly designed detection engine to pinpoint attacks in real time. When an attack is identified, Snort can take a variety of actions to alert the systems administrator to the threat.

Several important components are at work behind Snort: the packet sniffer, preprocessors, detection engine, logging and alerting mechanisms, and output processors. The packet sniffer listens to all IP packets on the wire and decodes them. The preprocessors take the decoded packets and perform some preliminary tasks, such as defragmentation. The detection engine, the heart of the Snort IDS, performs pattern matches on the data stream to identify known attack signatures. The logging and alerting mechanisms define how recognized attacks will be handled. The output processors define how the output of the logging and alerting system looks.

For faster processing (like keeping up with a 1 Gbit/s connection), one should use unified logging and a unified log reader such as barnyard. This allows Snort to log alerts in a binary form as fast as possible while another program performs the slow actions, such as writing to a database.

Preprocessors allow the functionality of Snort to be extended by allowing users and programmers to drop modular plugins into Snort fairly easily. Preprocessor code is run before the detection engine is called, but after the packet has been decoded. The packet can be modified or analyzed in an out-of-band manner using this mechanism. Besides doing the packet defragmentation and normalization (to prevent IDS evasion techniques), they could be used to perform the different sampling algorithms.

Chapter 4

Implementation

4.1 Design goals

Recalling what was written in Chapter 2, IDS consist in mainly three phases: data collection, data analysis and alarm notification. The bottleneck for high-speed traffic stands somewhere between data collection and data analysis: while it is theoretically possible to "capture" every single packet (a router could do that), the computational resources required for real-time analysis of all that information are prohibitive, even when considering typical-case scenarios with links of a few Gbit/s. A solution is to use a personal computer using a dedicated high-throughput card such as a DAG from Endace [22], but that is yet another expensive equipment, which goes against the whole idea of keeping costs reduced by using the routers themselves for collection and a basic level of processing.

The usage of sampling produces two levels of benefit: the obvious one is that it reduces the sheer amount of data to be analyzed, the less obvious is by skipping some of the packets, computing resources are available for longer periods (until the next sample is picked), thus allowing for more complex analysis algorithms.

Based on the state of the art, sampling in IDS can be tackled from two different perspectives: a more practical one whose intention is to monitor traffic deeper into the core networks (e.g., at the ISP level) and thus requires very fast operation even under very large traffic loads and a more academic one where the goal is simply to devise sampling algorithms that can be run offline and provide unbiased metrics suitable of being used by IDS engines. This thesis addresses both, with the aid of the prototype framework created to allow a modular, yet vertically integrated solution. This framework stems from the lack of open-sourced tools oriented at testing intrusion detection algorithms and their effectiveness under different sampling schemes. To the best of our knowledge, this prototype is the first of its kind and was engineered as a proof of concept, to be refined and updated as new research takes place but also to support said research by providing verifiable results.

A separate program that performs sampling on a given pre-captured trace was used to test the effect of sampling in Snort. It is a simplified, standalone version of the sampling in the main framework that allows the usage of either the deterministic (counter-based) or probabilistic (uniform random) methods. The deterministic (timer-based) method could not be implemented due to the internals of `libpcap` that uses the dissector itself to dump the sampled packets to the destination file. Without the dissector, it is not possible to access the timestamp of each packet and therefore there are no time references available to decide when to sample.

4.2 Framework for NIDS

The framework is a sampling-capable packet analyzer that performs detection of high-volume anomalies in network traffic by using entropy-based metrics. It is not limited to anomaly detection nor to high-speed traffic, but other more specialized tools already exist for the remaining cases, so this prototype focus primarily on those two aspects.

Since the underlying intention is to compare the performance of different combinations of metrics/sampling methods, some care is needed to guarantee that results are comparable. To that end, a segmentation system based on time bins was implemented, being each bin an autonomous unit containing statistical data referring to all packets captured within its duration. An alternative binning strategy based on the count of packets is also implemented for comparison sake.

The example feature chosen for entropy calculation was the destination port in both TCP and UDP. This is calculated per bin and compared against an adaptive baseline based on a history of bins; if their difference exceeds a set threshold, the bin is flagged as anomalous and an alert is issued. An automated classifier then suggests likely classes of attacks that could cause the detected symptoms.

4.2.1 Architecture design

The basic architecture of the prototype framework is shown in Figure 4.1 and each component as well as the information flow is summarized in the following paragraphs. More detailed information is described in later sections of this chapter.

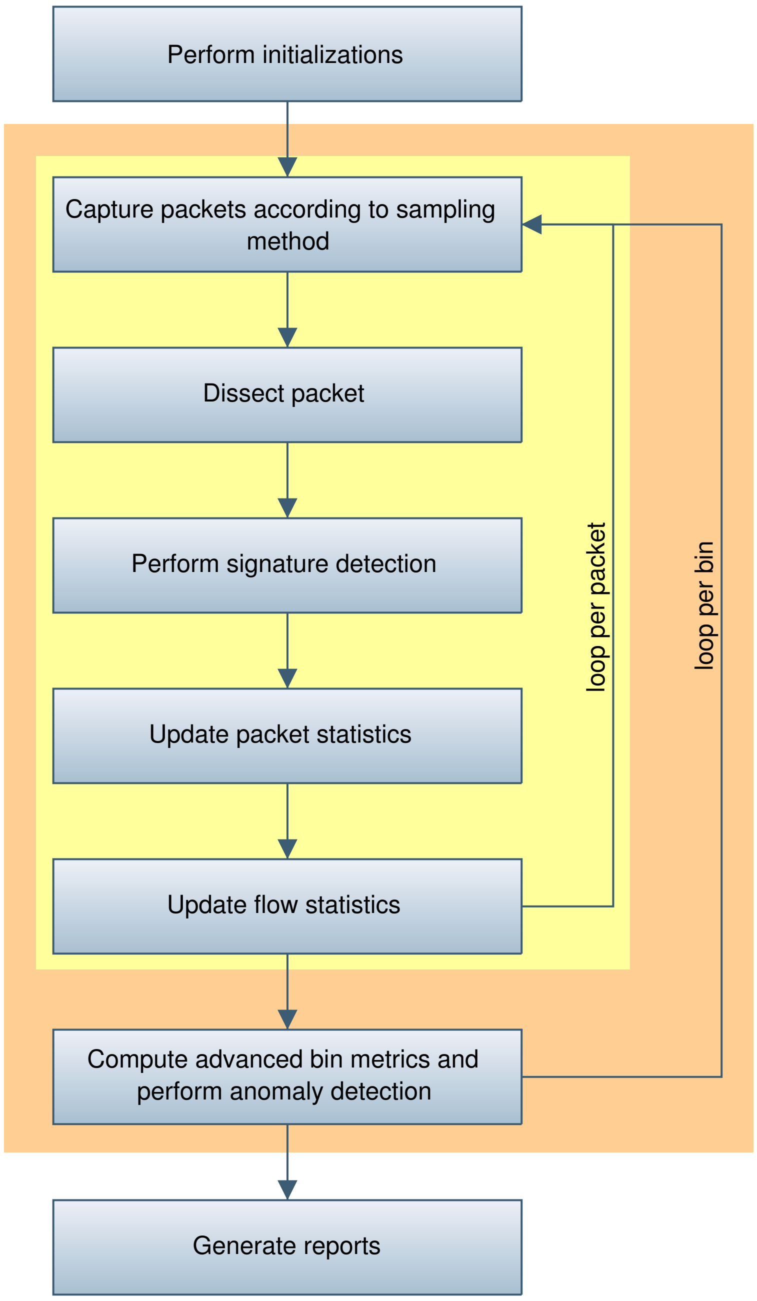

Perform initializations
Capture packets according to sampling method
Dissect packet
Perform signature detection
Update packet statistics
Update flow statistics
Compute advanced bin metrics and perform anomaly detection
Generate reports
loop per packet
loop per bin

Figure 4.1: Architectural diagram of the prototype framework

After performing the required initializations, the packet capture begins according to the desired sampling method, or without any sampling. This is achieved with the `pcap` library that transports the raw data of any incoming packet that passes through the BPF into userspace, where the framework decides whether to sample it or discard it. In a production scenario this sampling code should be moved in into `pcap` itself to avoid unnecessary overheads and create a monolithic application that provides maximum performance at the expense of loosing its modularity and portability. It would also be beneficial to move the main loop and enable support for PF_RING[43] sockets.

Next, the packet is dissected and the packet statistics updated to reflect the information contained within its header. Current statistics are grouped by protocol and include number of packets, number of bytes and source and destination port distribution. Information about the TCP flags, namely the number of ACK packets, is also maintained.

While specialized signature detection applications exist (such as Snort), simple ones can be implemented at this stage directly in the framework if they are needed to evaluate a particular threat. For example, detecting connections with unusual or impossible TCP flags. However, if said signatures are known in advance, they should be included directly in the Berkeley Packet Filter (BPF). But a real world use could be to only engage in such processing if there is an indication of an ongoing intrusion, i.e., some anomaly was detected and packets started being compared to signatures to classify the particular attack.

A form of aggregation into Origin - Destination (O-D) flows was implemented using hashing mechanisms. This is a simplified version of a full flow as it does not consider the ports and was chosen as a proof of concept to provide an estimate of the total flow counts in real time.

At the end of each bin, a more complex analysis on the stored data is performed. The current implementation calculates the destination port entropy and compares it with a baseline, in turn updated to take the current bin's information into account. If the difference between the entropy of the current bin and the one of the baseline crosses the defined threshold, an alarm is raised.

Finally, at the end of the capture, reports are generated and written to several logfiles containing a summary of the statistics per bin as well as flow information.

This framework can be extended by adding other, more complex, sampling methods or intrusion detection algorithms in a simple and straightforward way. What is truly important is to maintain a logical sequence of operations and allow alternatives to be tested under similar conditions.

4.2.2 Development

The framework is implemented in the C programming language on Linux kernel 2.6 series running on x86 architectures and requires `libpcap`[49] and `uthash`[27]. Portability has not been tested yet.

`pcap` library

The `pcap` library (also known as `libpcap` in the *NIX-like environments or `WinPcap` for the Microsoft Windows version) provides advanced packet capturing abilities in user-space. The Berkeley Packet Filter (BPF) [40] was introduced to simplify and fasten the filtering of network traffic and is already integrated into `libpcap`.

This prototype uses `libpcap` as the packet capture method due to the following reasons (by order or relevance):

- simple API;
- efficient, stable and portable implementation;
- widespread use;
- BSD-license

A limitation of this approach is the slight loss of performance when compared to using the OS native methods for raw packet handling (PF_PACKET in Linux) due to the sequence of various calls. But this would inhibit portability and the loading of pre-captured traces for testing. A solution could be to develop separate pipelines for processing: one generic and portable using `pcap` for testing and development and another, OS-specific and high-performance, for use in a production environment (compilation-time enabled). A more dramatic approach would be to put the entire framework into kernel-space, but that is not advisable for security reasons, especially during development, as even minor bugs can have disastrous consequences.

Sampling methods

Sampling should be performed as soon as possible (ideally in the NIC) in order not to waste resources by capturing a packet that is going to be discarded later on. Since retail-grade network adapters do not allow for such mechanisms, it is up to the software to keep track of when to actually sample a packet. Since `libpcap` does not provide sampling mechanisms either, this was done in the application. Editing the `libpcap` source code to implement the various sampling methods is clearly the way to go, but that would require knowledge of all the architectures and OSs in order not to break portability.

Due to `libpcap` being able to load pre-captured files and treating it similarly to live captures, and because of the above limitations in collecting high-speed traffic with the required efficiency, offline analysis posed challenges in the implementation of sampling.

The main loop of the prototype consists in calling the function of `libpcap` that delivers the next incoming packet to the dissecting function, regardless of whether that packet will actually be processed or not. For the deterministic counter-triggered and the uniform random sampling methods this is still easy to solve by using a branch instruction and a dummy dissector. But for the deterministic timer-triggered sampling method it requires a lot of overhead and modifications to the dissector itself. This is because when reading form the pre-captured file, one packet at a time, there is no notion of time, becoming necessary to "capture" the packet, read its timestamp and only then evaluate whether enough time has passed since the last sampling in order to continue with the analysis or to discard the packet.

This issue does not exist in live-capture, when access to system timers is available to keep track of passing time - but once again, some OSs might have problems with timestamp accuracy and require different approaches.

While the current implementation suffers from this performance penalty, it has accurate sampling using all three methods. But this accuracy could also be seen as a disadvantage, because no matter how many times a trace was analyzed using a certain sampling period, it would always select the same packets on every run. This may lead to biased indicators and inaccurate statistics. To overcome the problem, a random offset is introduced in the beginning of every run, as explained in Figure 4.2:

Bins

Statistics are processed per bin, and therefore the bin should be long enough so that a large number of packets arrive within, but not so long that all that is obtained is the average traffic level and thus accuracy becomes diminished. This is highly dependent on the traffic level, but it was already assumed that it would be high enough to ensure a decent minimum of sampled packets.

Binning the packets in offline mode suffers from the same problem as sampling: there is no notion of passing time, therefore, in the unlikely scenario when a link suffers a temporary disruption and no packets arrive, the program can only "close" the 'pending bin and possibly the following also empty bins when another packet arrives, even if that only happens a long time in the future, leading to those backlog tasks being performed before continuing

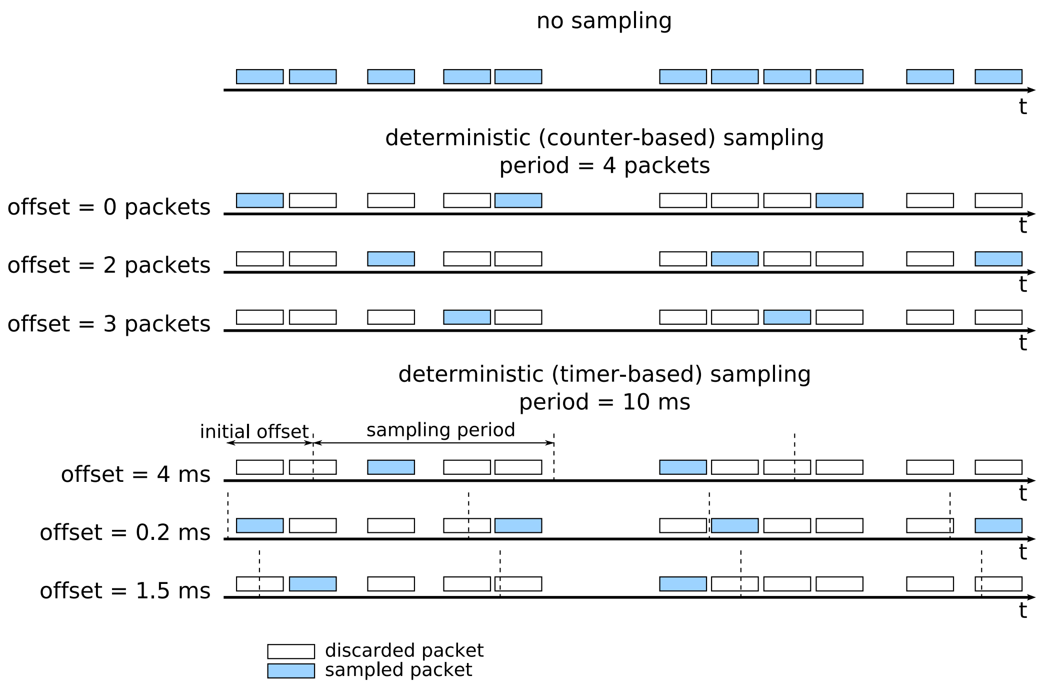


Figure 4.2: Examples of random initial offset

with the normal execution. Obviously, this is not a severe issue (because it is offline processing - no need for real-time performance), but may delay the alerts for an extended time.

If no more packets are found before the last bin is complete, it gets discarded and the program terminates showing the statistics up until the last complete bin.

This is particularly visible in aggressive sampling rates reporting less bins than the unsampled run because the traces contain few packets in those final bins, and thus the trace ends before reaching the packet that would be sampled (this does not happen in timer-based sampling methods).

Flows

Flows are identified by the origin and destination IP addresses and the next-layer protocol, either TCP or UDP. To increase performance and avoid implementing all the insertion and look up routines, hashing[27] has been used with good results in terms of processing speed. The actual hashing algorithm being used is the default implementation in `uthash` (Bob Jenkins's Hash Table Lookup) and is freely available in the source code or on his website `http://burtleburtle.net/bob/hash/index.html`.

When a new packet is sampled, it is checked to see if it matches a flow already present in cache, and if not, it is added. If if it does, the counter associated with that flow entry is increased by one. This is a simplified approximation of actual flows because no timeouts nor explicit flow terminations (FIN) are taken into account. This is not very problematic because it is unlikely that within the same bin, different flows are erroneously interpreted as being the same. However, the lack of overlap between bins prevents the flow count from being used as an accurate indicator, since elephant flows will be broken down and mouse flows will appear to have more relevance.

To develop a complete flow solution would require too much processing effort and should probably be targeted toward direct implementation on specific hardware (FPGA) for real-time or resort to NetFlow/IPFIX-like libraries already available for use in offline analysis with commodity computers.

Detecting anomalies

The detection engine is a comparator between the metrics of the current bin and the ones of the baseline - if the difference between them is above a certain threshold it is considered an anomaly. The most important metric is the destination port entropy, so the remainder of this section will focus on that. However, several anomalies can be detected with the current implementation, as listed in Table 4.1.

Table 4.1: Detected anomalies

Type	Criteria
portscan/ipsweep	entropy increase
worm	entropy decrease
network disruption	low packet count
network failure	near zero packet count
DoS,flash crowd or network reorganization	high flow count
SYN flood	high SYN percentage

The baseline is calculated from the entropies of past bins, currently the previous ten bins, using a simple average technique. This allows it to adapt naturally to ordinary, slow-moving changes in traffic while still attenuating sudden bursts of possibly benign origin. Since the calculation of entropy and comparison against the baseline are both done once per bin (at its end) their complexity can be quite higher than the current implementation; they could even be offloaded to a secondary processor and leave the main processor doing only the packet capture and statistical profiling because they are two independent functions. There is also no need for memory access controls (mutexes) because the source data is contained in a circular buffer with ten positions (one for each bin in the history), so as long as this execution takes less time than the duration of the bin, the still occupied buffer positions will never be overwritten. Figure 4.3 describes this mechanism.

The definition of empirical entropy is given in Equation (4.1), where S is the stream of packets containing n feature vectors (in this case, destination ports) with sizes (number of packets) $a_1, a_2, ..., a_n$ respectively, and $s = \sum_{i=1}^{n} a_i$ is the total number of packets in S.

$$H(S) = -\sum_{i=1}^{n} \frac{a_i}{s} \log_2 \left(\frac{a_i}{s}\right) \tag{4.1}$$

The implementation follows the definition, except it uses the natural logarithm (base e). This is not a problem because they are only different by a constant multiplicative factor and do not affect the performed analysis.

The idea is to develop a better mapping of packet characteristics to feature vectors in order to achieve a compact metric (one floating point number) that contains varied information. This is beyond the scope of this thesis but some of the papers studied and pointed out in Section 3.2.3 provide viable solutions. Actually, the framework facilitates the development and testing of said mappings, allowing for better tuning of parameters and comparison of results.

Reporting

The reports (logfiles) are in human-readable form and complement the on-screen alerts given when an attack is detected. They contain the information that is used to construct the graphs and represent what would be sent to a centralized collector using IPFIX/PSAMP templates.

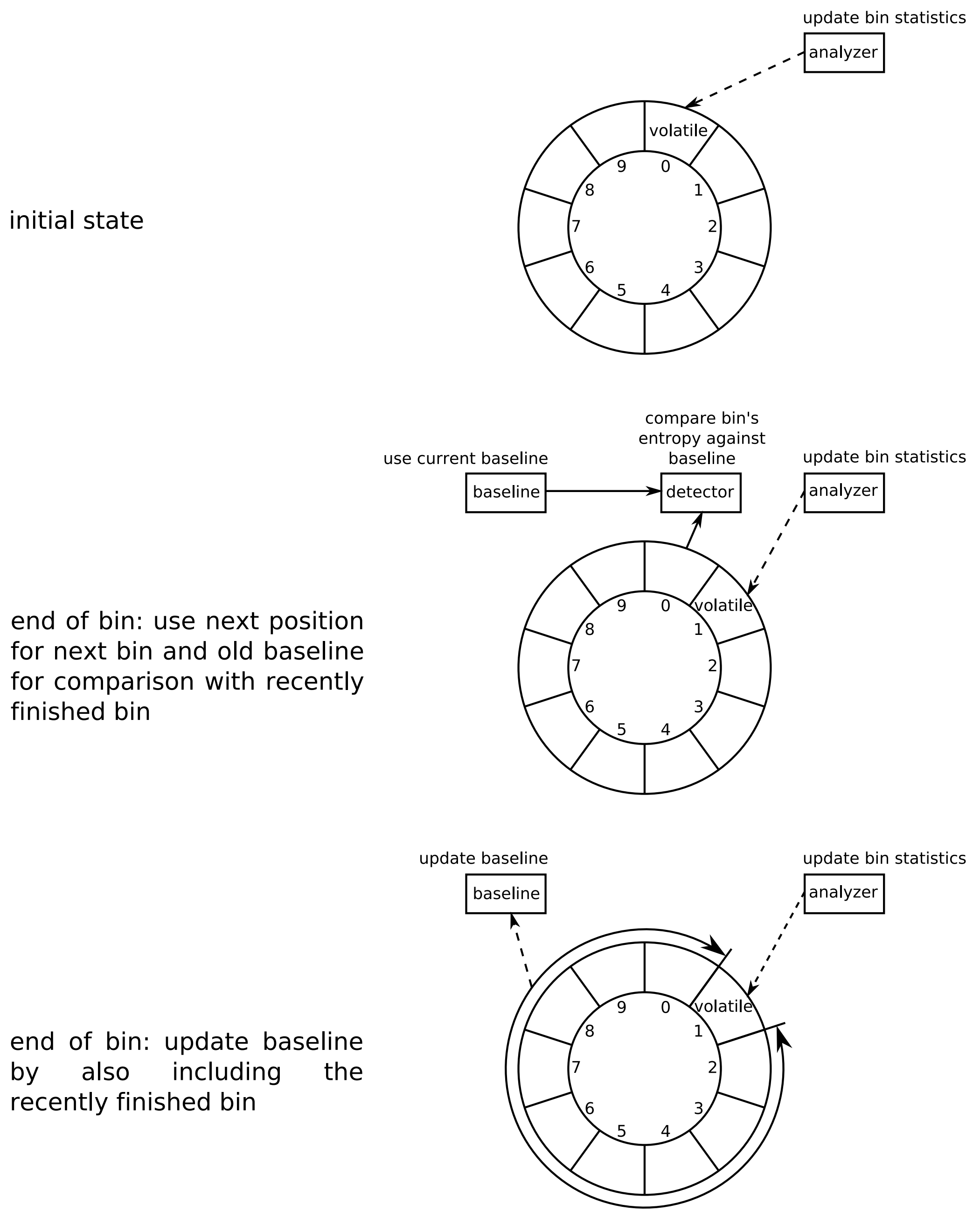

initial state
update bin statistics
analyzer
volatile
9
0
1
2
3
4
5
6
7
8
end of bin: use next position for next bin and old baseline for comparison with recently finished bin
use current baseline
baseline
compare bin's entropy against baseline
detector
update bin statistics
analyzer
volatile
end of bin: update baseline by also including the recently finished bin
update baseline
baseline
update bin statistics
analyzer
volatile

Figure 4.3: History circular buffer being accessed in different ways

4.2.3 Implementation highlights

The most important features already implemented in the prototype are listed here:

- three sampling methods available; two deterministic and one probabilistic
  - systematic counter-based with random initial offset
  - systematic timer-based with random initial offset
  - uniform random
- all header fields are available and can be used for subsequent calculations
- analysis and graphical reporting of major traffic parameters per protocol
  - counters for packets, bytes, SYNs (TCP only) and flows
  - distribution and entropy of packets per destination port and protocol
- O-D flow lookup and aggregation per protocol
- processes live captures or pre-captured `pcap` traces
- adaptive computation of entropy baseline
- real-time detection and automated classification of anomalies

4.2.4 Assumptions and limitations

This framework is a work in progress and still lacks some features, despite the main architecture being well-defined. This section describes the current limitations or caveats associated with it and workarounds or suggestions for future improvements.

Link speed and traffic volume

Since high-speed links with high traffic volume are the main target of this thesis, the prototype was designed with that in mind. While it does work with any speed and any level of traffic, it does not perform very well with higher sampling periods (because few packets are actually sampled). Sporadic periods of inactivity will be considered a loss of connectivity and get reported as an anomaly.

Protocols

This initial release assumes all network traffic to be TCP or UDP over IPv4 over Ethernet, though this is only for simplicity as there is no architectural obstacle to prevent its use with IPv6 or over any other Data Link Layer. If such additional support is required, it will suffice to update the BPF rule and add a packet dissecting function capable of identifying the key fields (addresses, ports, etc.) and payload offsets, not unlike the ones existing in Wireshark[54].

No provision for IP fragmentation has been developed so far; not only it is such a rare occurrence (according to [31], only 0.06% of packets are fragmented) but reconstruction would also add a considerable overhead, assuming that all fragments were even routed through the monitored link to make it possible.

There is some support for Internet Control Message Protocol (ICMP), but only for preliminary testing purposes because it accounts for very little traffic outside of LANs.

Encryption

Because no DPI is performed, the prototype is unaffected by the encryption of the actual payload; however, if the encryption is applied at the IP level, the underlying TCP or UDP headers cannot be processed. In fact, the default BPF rule only allows “plain” TCP or UDP (implicitly over IP) packets and will silently drop everything else. However, support for IP security (IPsec)’s Transport Mode using Authentication Header (AH) or unencrypted Encapsulating Security Payload (ESP) is trivial to implement (again, it is a matter of taking the right offsets into account during the dissection of the raw packet). Obviously, it cannot possibly work with Tunnel Mode (only the encapsulating packet will be seen and discarded, as there is no way to retrieve the encapsulated encrypted packet’s original TCP or UDP header).

Sampling accuracy

The current implementation has microsecond accuracy, but for multi Gbit/s links there is a need for more refined time granularities. Since this is NIC and OS dependent, does not impact the architecture in any way, and no tests at such speeds were going to be performed, it was relegated for a future improvement.

4.2.5 Deployment scenario and integration

The proposed application of this framework is to monitor a backbone link and report the alerts to a centralized collector. This can be achieved with the current prototype by piping the output to a socket connected to the management station, but it is planned to create an export template for use with NetFlow and achieve more proper integration. This would also allow to export not only the alerts but even copies of packets found in suspicious bins. The deployment diagram is shown in Figure 4.4.

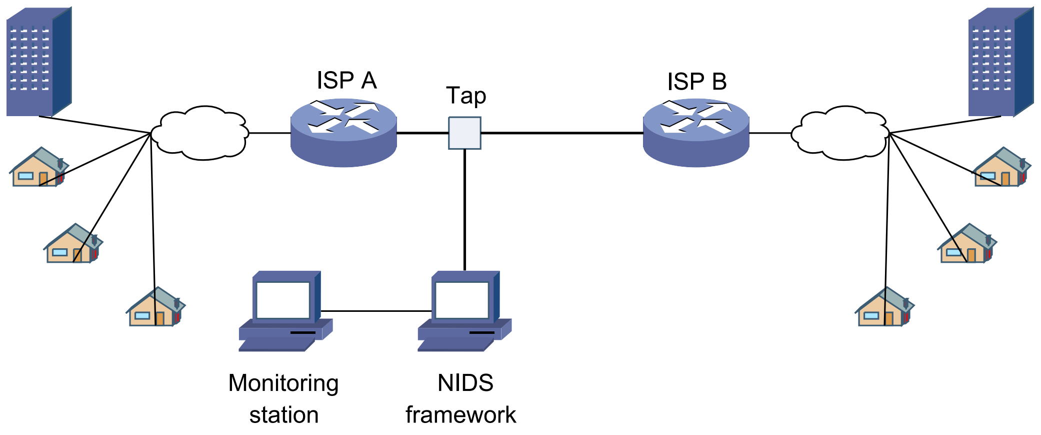


Figure 4.4: Possible deployment scenario

Chapter 5

Tests and results

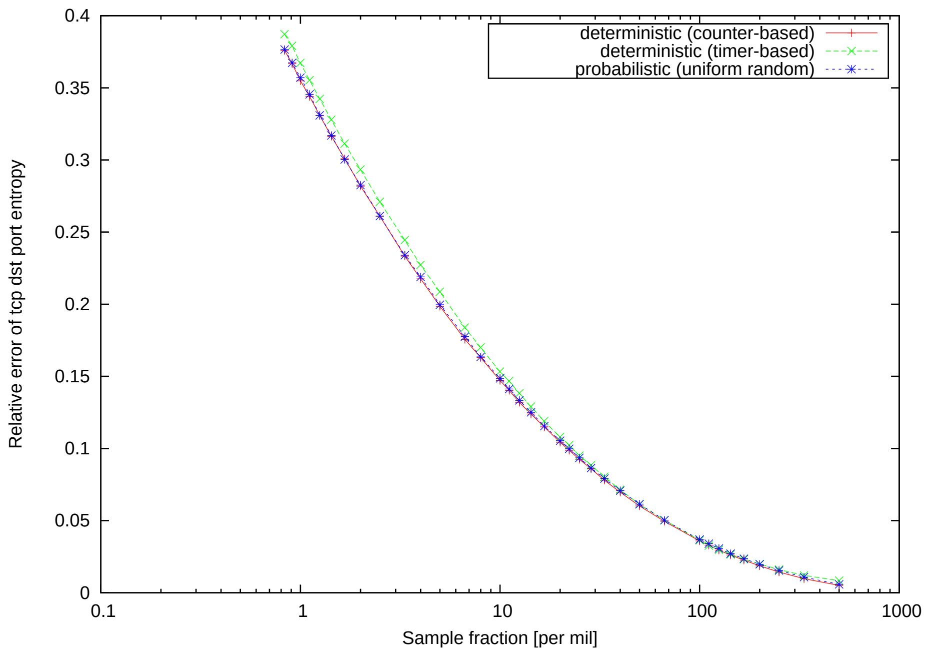
5.1 Comparison of sampling methods

In order to compare sampling methods regarding their effect on anomaly detection, a test was conducted in order to quantify the error of the estimated metric vs. the actual metric obtained without sampling. This was performed using pre-captured traces and running them several times with different sampling methods and rates, comparing the metrics per bin (using a bin duration of 5s) and normalizing the difference in order to obtain a relative error.

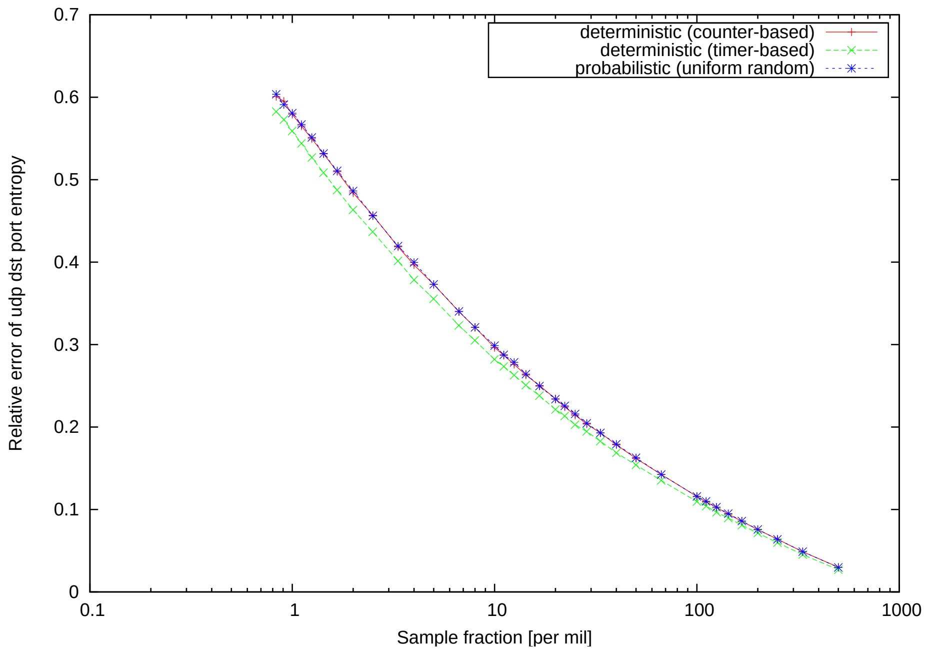
Results for each of the two traces are presented in in Figures 5.1 and 5.2. More information can be found in Section 5.4.1. Each point consists in the combination of a sampling method and a sampling rate, and was repeated twenty times for improved confidence and the relative errors per bin of each run were averaged to provide a single value for that specific point.

The sample fraction is presented per mil and reflects the inverse of the target sampling period in packets for the deterministic (counter-based) and uniform random methods. For the deterministic (timer-based) method, there is no accurate way to map the sampling time intervals to a fraction of sampled packets, so an approximation was used supposing that the packet interarrival times are constant during the trace and therefore the number of packets arriving in each sampling period can be determined. If c is the total number of packets received and t is the duration of the trace, the target sampling interval s is derived from the target sampling fraction in packets f, as given by $s = \frac{f\,t}{c}$. The actual sample fraction (in packets) resulting from this sample interval is reported by the prototype and used in the graph.

There are no results for sampling rates above 500 per mil (50%) because that is the minimum distance between consecutive packets ("pick one, drop one, pick another"). Also, such high rates are of little interest.

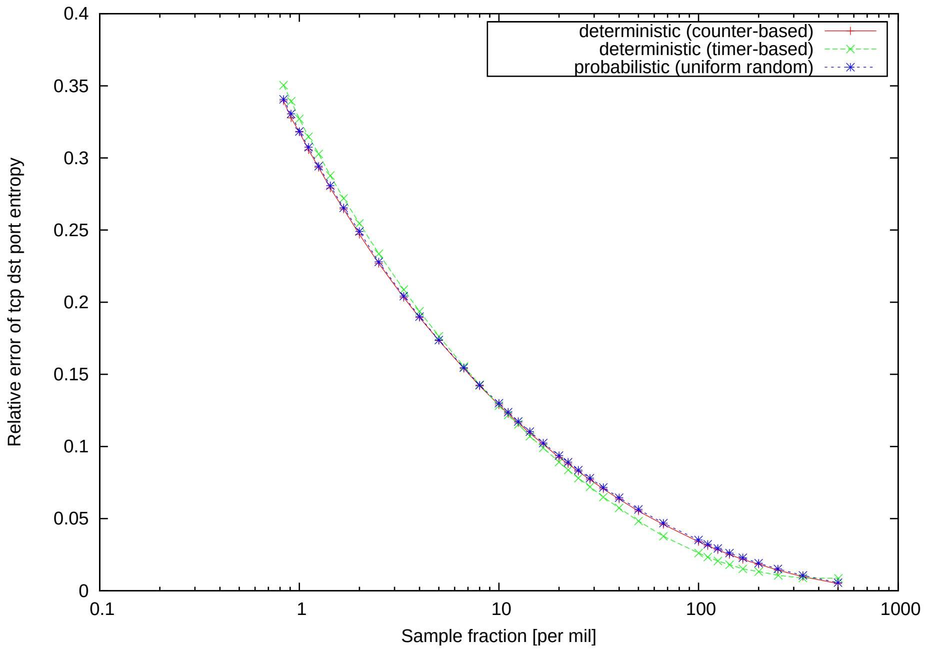


(a) TCP

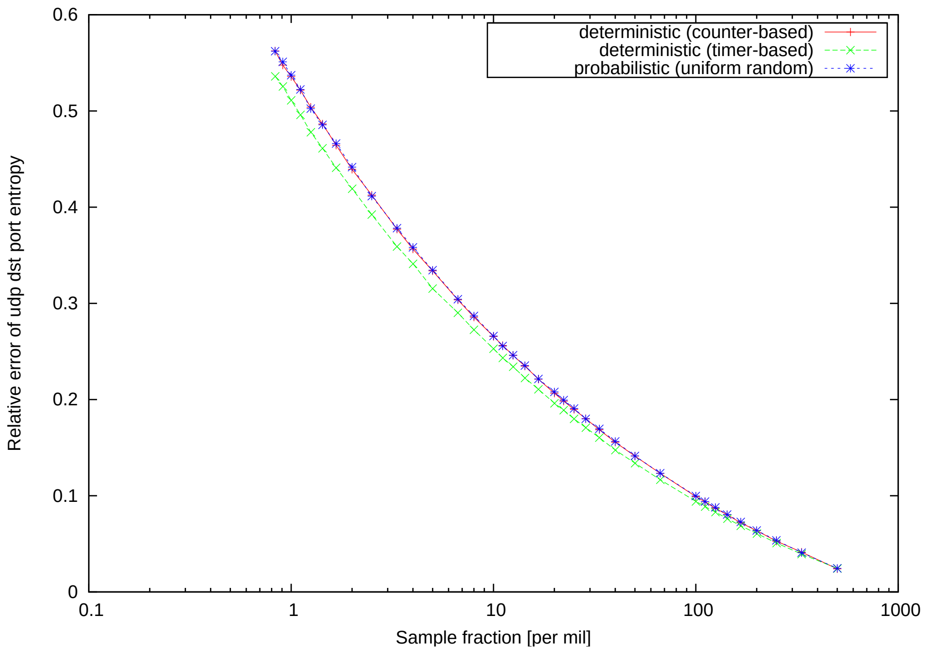


(b) UDP

Figure 5.1: Relative sampling error (MAWI 200803172345)



(a) TCP



(b) UDP

Figure 5.2: Relative sampling error (MAWI 200803201630)

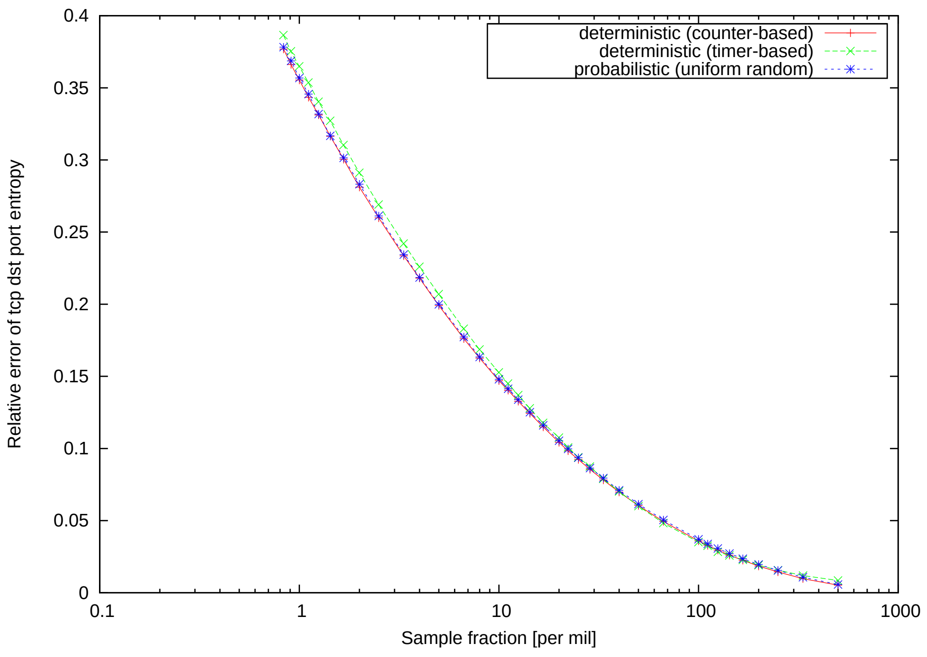
As expected, a decreasing exponential behavior is observed, with typical values of <5% error for 500 per mil sampling and >30% for 1 per mil sampling in the case of TCP or >50% in the case of UDP. All three sampling rates exhibit similar performance, with deterministic (counter-based) and uniform random overlapping and deterministic (timer-based) within a 1% boundary. The second trace (MAWI 200803201630) has ≈ 5% less error for the lower, more aggressive, sampling rates likely due to an higher average load as detailed in Table 5.2, which means that more packets are actually sampled in each bin.

Because the sampling mechanism is unaware of the protocol type, the UDP traffic, which only accounts for about 8% of the total, might have been impaired when compared with TCP. To check that out, the BPF filter was adjusted to admit only one of those protocols and simulate an architecture where each protocol has its own pipeline, a mix of content-based sampling with the previously mentioned methods. The results are shown in Figures 5.3 and 5.4.

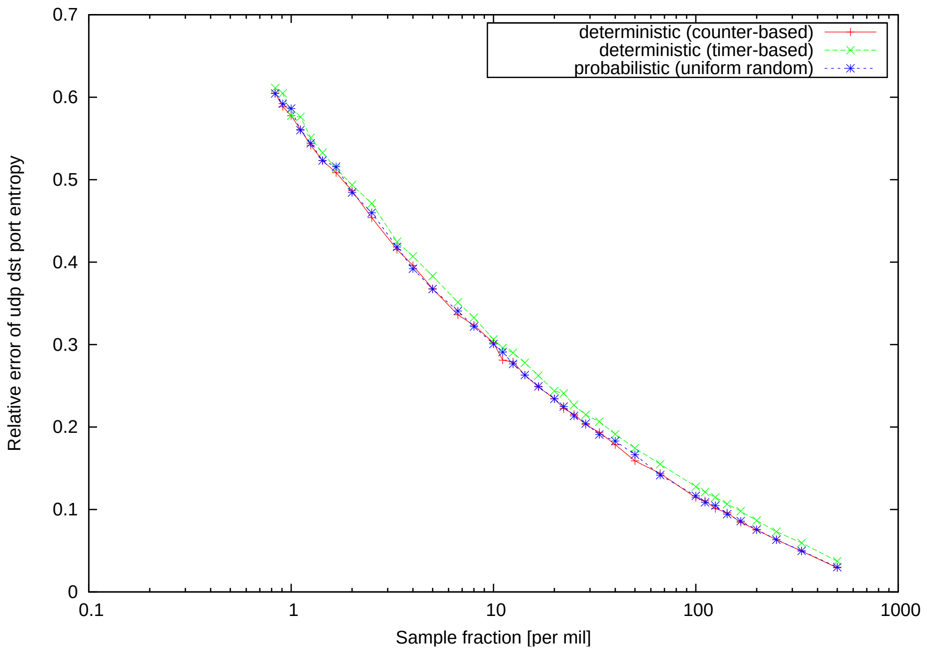
The values show that the error levels only decreased by less than 1%, both in TCP and UDP. This means that the overall number of sampled packets within each bin is also important, and not just the sampling rate. Particularly in the UDP case, when the sampling was unaware of the protocol, the fact that UDP comprises only a small portion of total traffic was mitigated because most of the dropped packets were therefore TCP. When using protocol aware sampling, all sampled packets are UDP but so are all dropped packets. That is why the results are so close.

5.2 Detecting anomalies

To attempt detecting real intrusions, another set of pre-captured traces was used, as detailed in 5.4.2. These contained some attacks that affect the distribution of packets and the results of various monitored metrics are displayed in Figures 5.5 through 5.9 for the first trace and Figures 5.16 through 5.20 for the second trace. Screenshots of the execution are also shown in Figures 5.10 and 5.21, where the alerts are displayed. Finally, the evolution of the computed adaptive baseline is shown in Figures 5.11 through 5.15 for the first trace and Figures 5.22 through 5.26 for the second trace. The duration of each bin was 10000s and the history from where the baseline is calculated was comprised of the latest 10 bins. The sampling method used was probabilistic (uniform random).

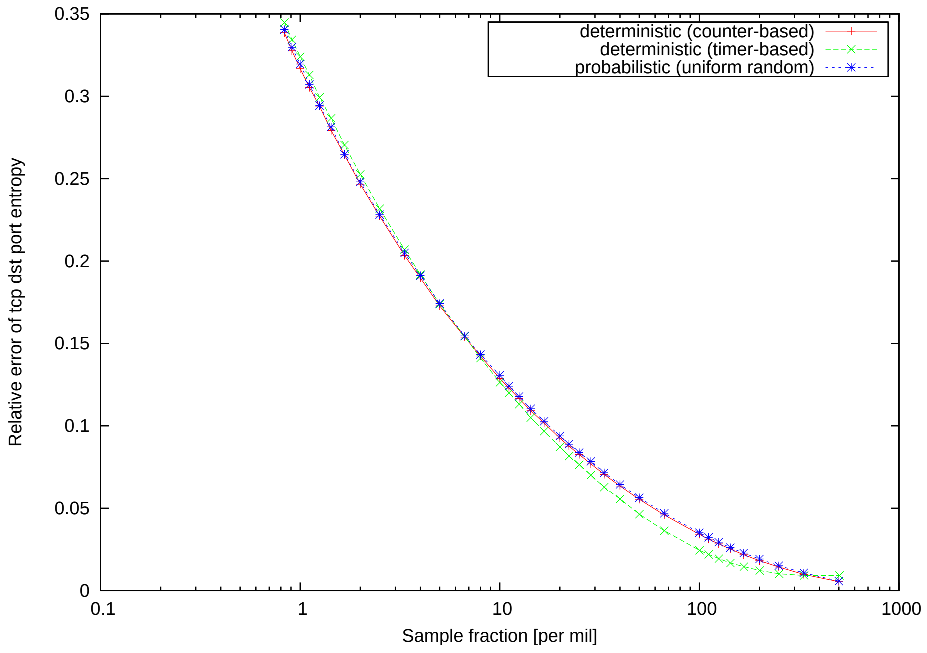


(a) TCP

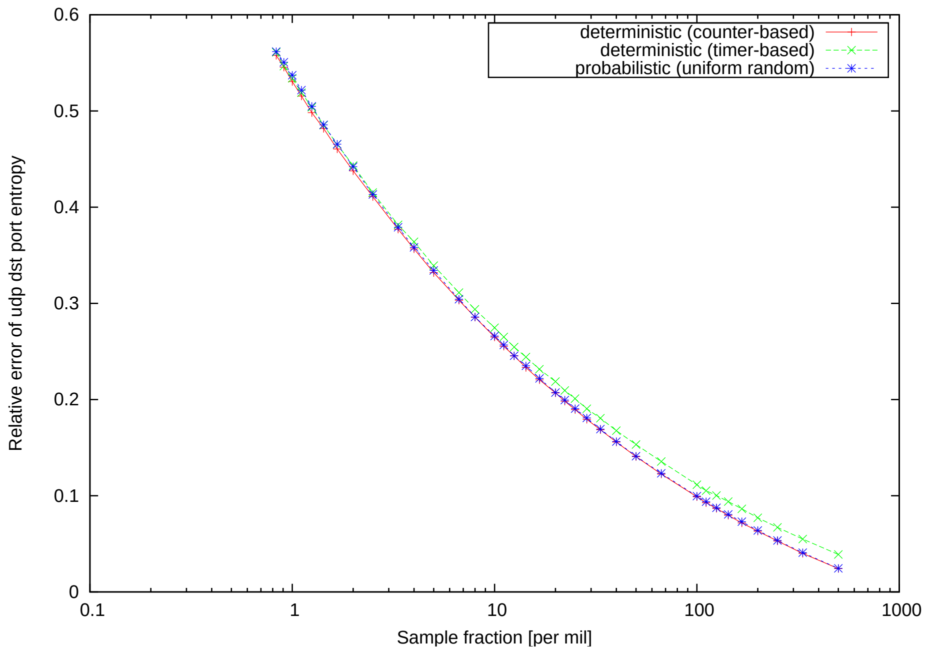


(b) UDP

Figure 5.3: Relative sampling error - protocol aware (MAWI 200803172345)

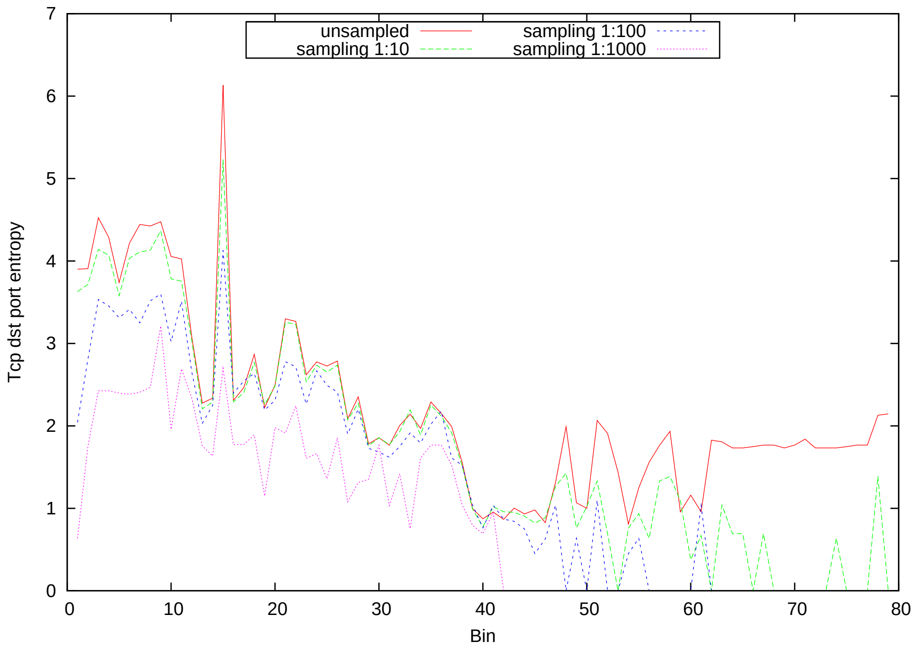


(a) TCP

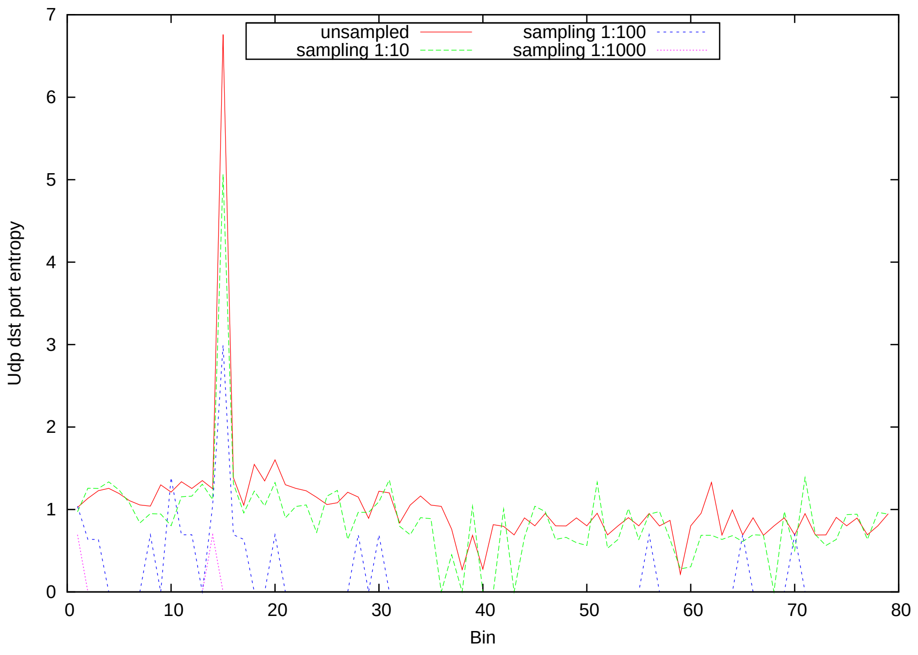


(b) UDP

Figure 5.4: Relative sampling error - protocol aware (MAWI 200803201630)

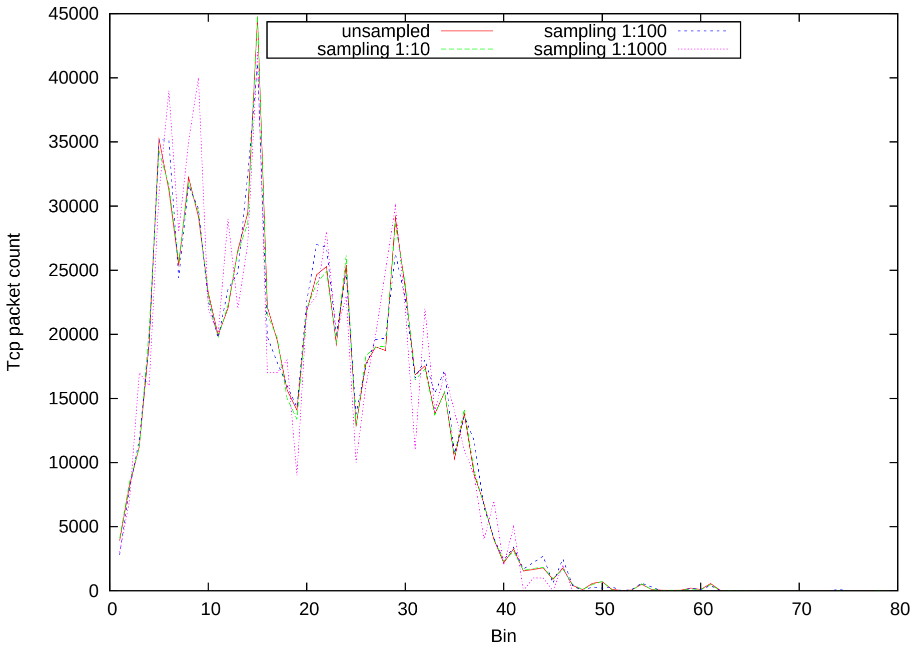


(a) TCP

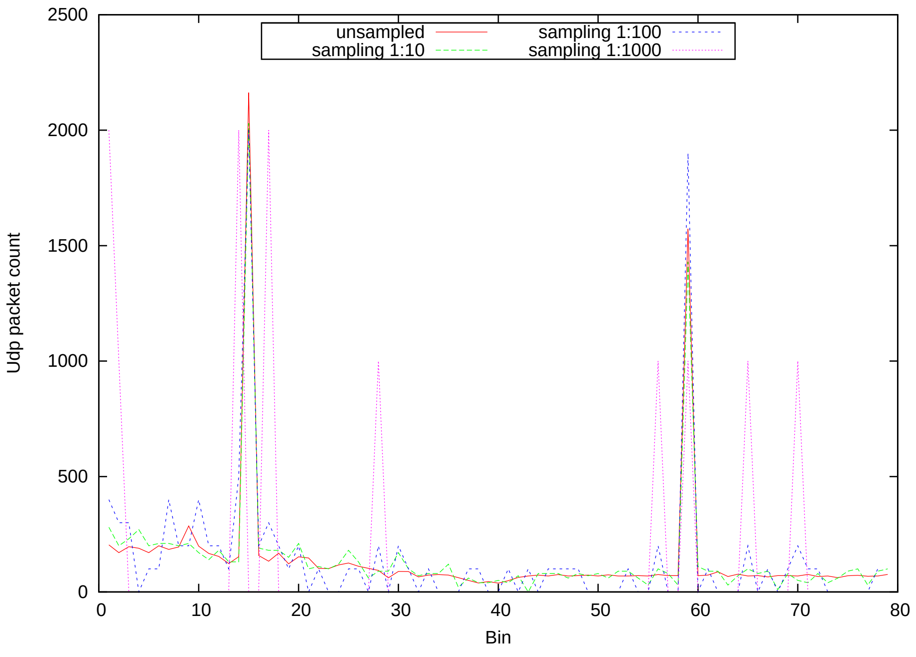


(b) UDP

Figure 5.5: Destination port entropy - DARPA MIT 1999 week 2, Wednesday

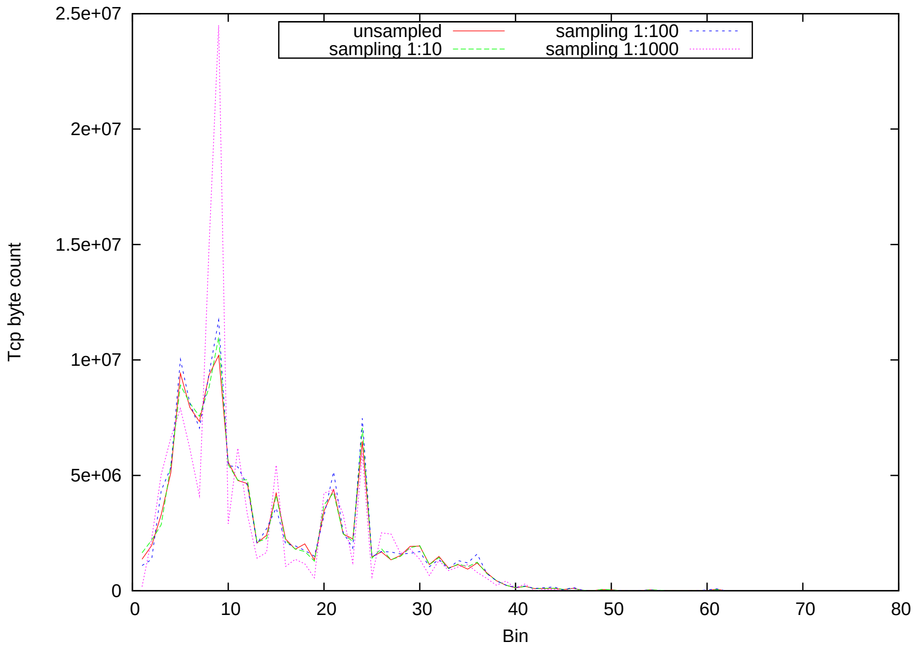


(a) TCP

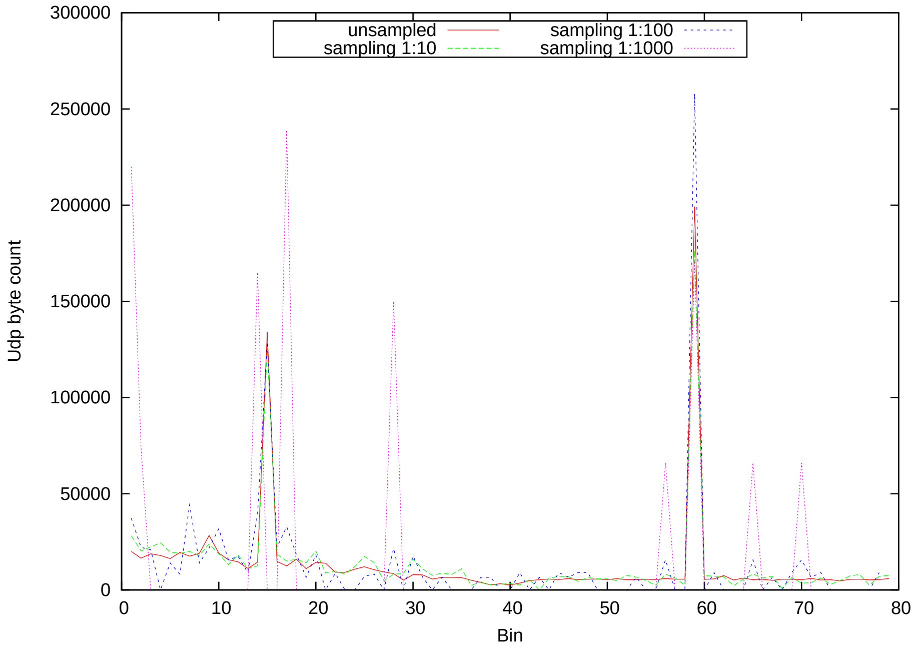


(b) UDP

Figure 5.6: Packet count - DARPA MIT 1999 week 2, Wednesday

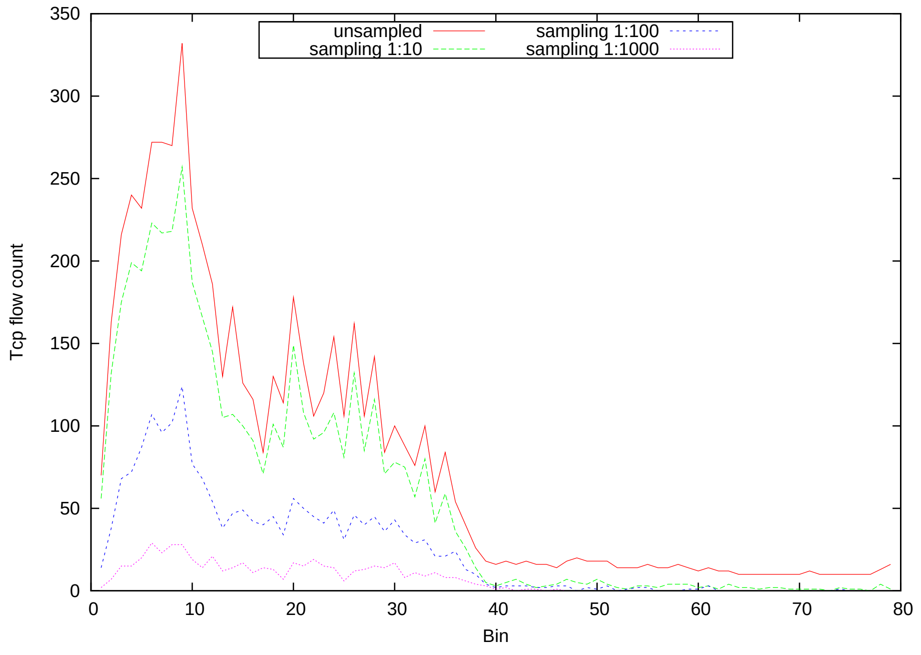


(a) TCP

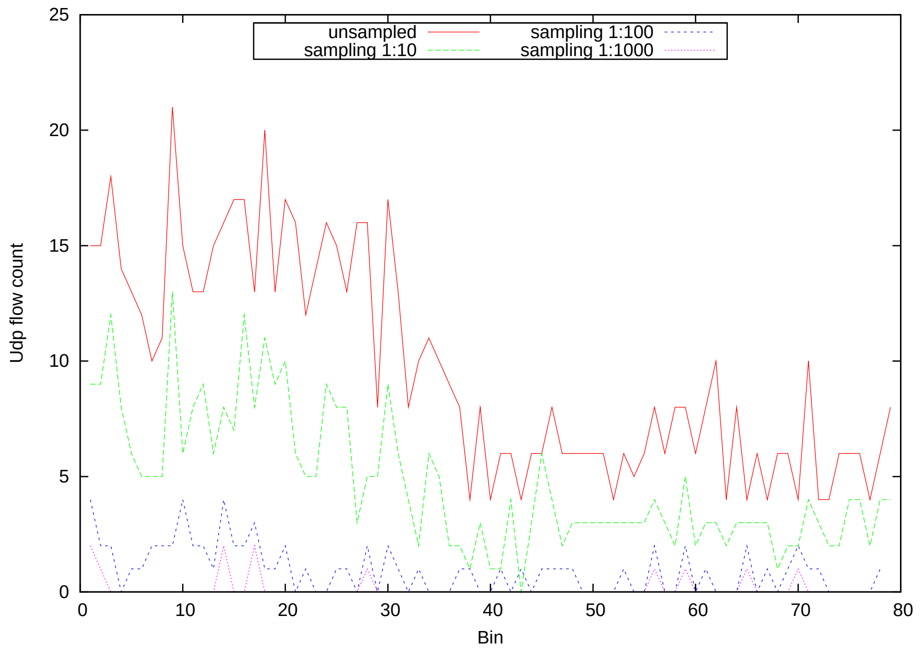


(b) UDP

Figure 5.7: Byte count - DARPA MIT 1999 week 2, Wednesday



(a) TCP



(b) UDP

Figure 5.8: Flow count - DARPA MIT 1999 week 2, Wednesday

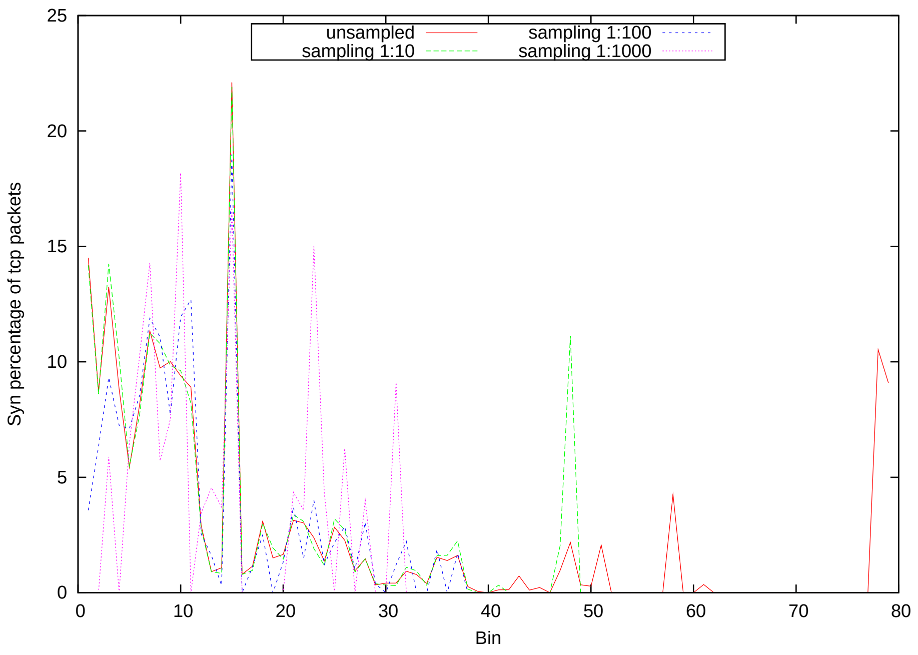
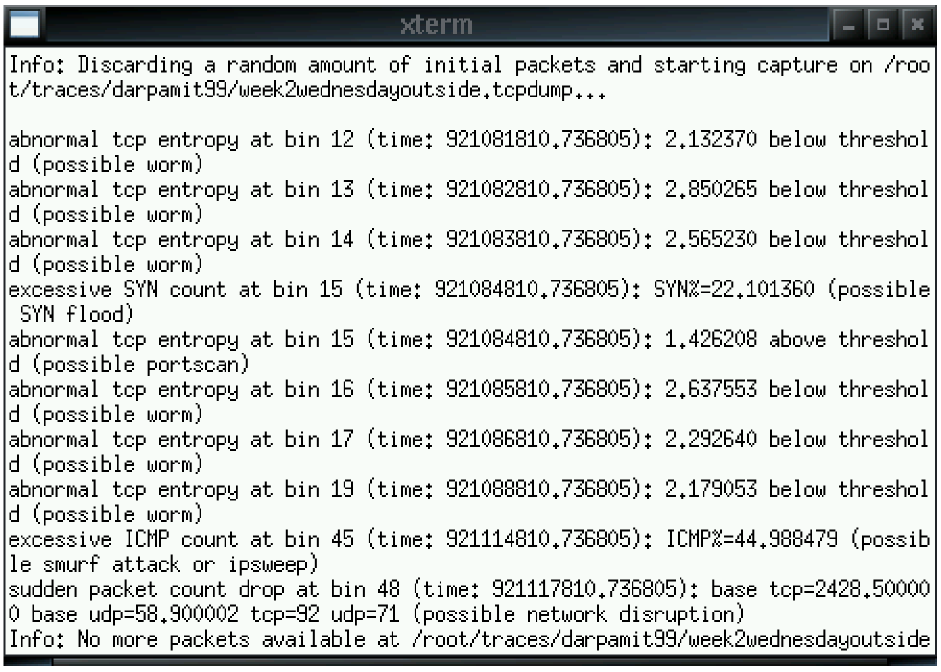


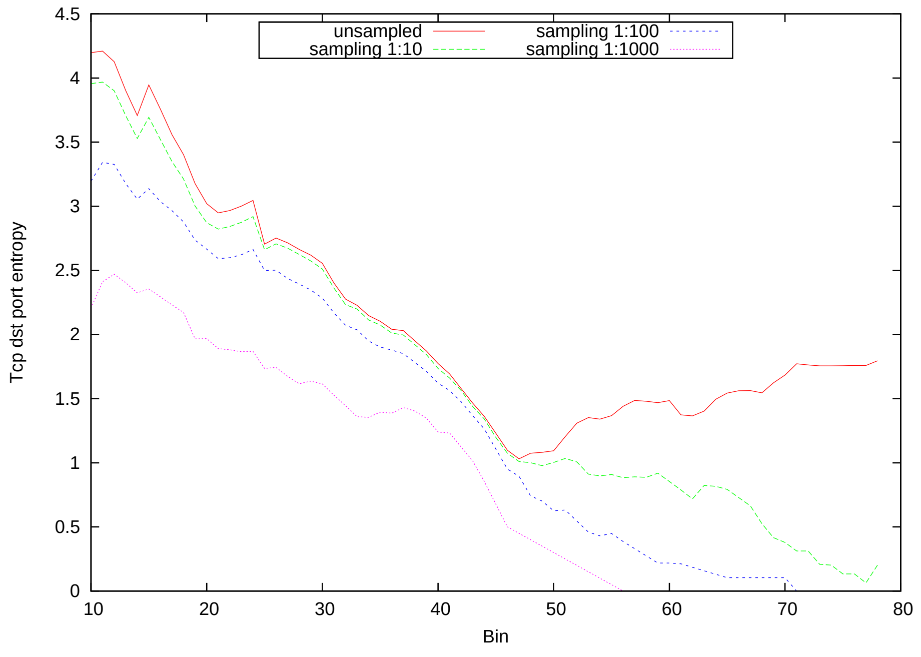
Figure 5.9: SYN percentage - DARPA MIT 1999 week 2, Wednesday



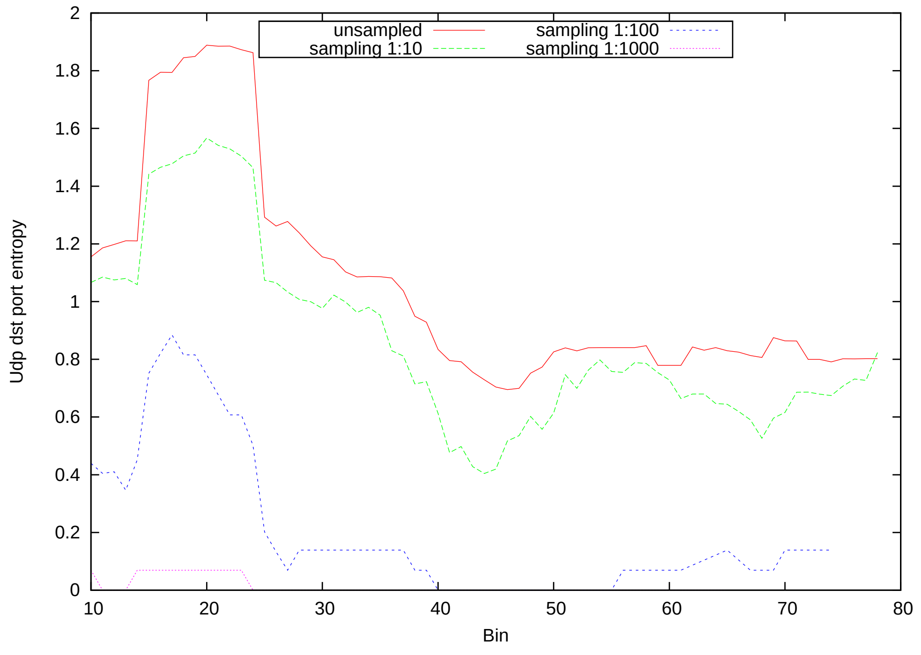
```
Info: Discarding a random amount of initial packets and starting capture on /roo
t/traces/darpamit99/week2wednesdayoutside.tcpdump...

abnormal tcp entropy at bin 12 (time: 921081810.736805): 2.132370 below threshol
d (possible worm)
abnormal tcp entropy at bin 13 (time: 921082810.736805): 2.850265 below threshol
d (possible worm)
abnormal tcp entropy at bin 14 (time: 921083810.736805): 2.565230 below threshol
d (possible worm)
excessive SYN count at bin 15 (time: 921084810.736805): SYN%=22.101360 (possible
 SYN flood)
abnormal tcp entropy at bin 15 (time: 921084810.736805): 1.426208 above threshol
d (possible portscan)
abnormal tcp entropy at bin 16 (time: 921085810.736805): 2.637553 below threshol
d (possible worm)
abnormal tcp entropy at bin 17 (time: 921086810.736805): 2.292640 below threshol
d (possible worm)
abnormal tcp entropy at bin 19 (time: 921088810.736805): 2.179053 below threshol
d (possible worm)
excessive ICMP count at bin 45 (time: 921114810.736805): ICMP%=44.988479 (possib
le smurf attack or ipsweep)
sudden packet count drop at bin 48 (time: 921117810.736805): base tcp=2428.50000
0 base udp=58.900002 tcp=92 udp=71 (possible network disruption)
Info: No more packets available at /root/traces/darpamit99/week2wednesdayoutside
```

Figure 5.10: Screenshot - DARPA MIT 1999 week 2, Wednesday

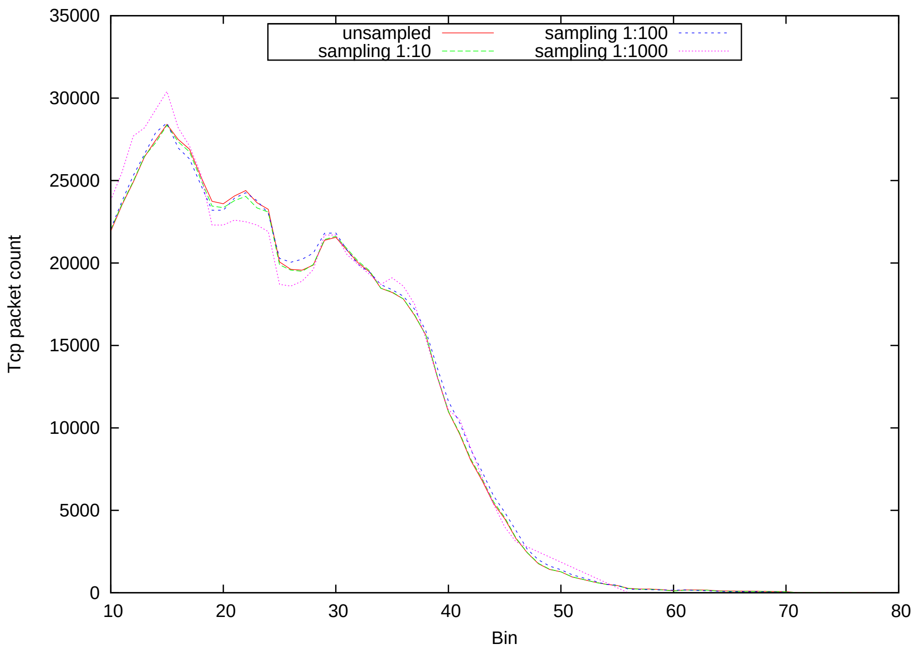


(a) TCP

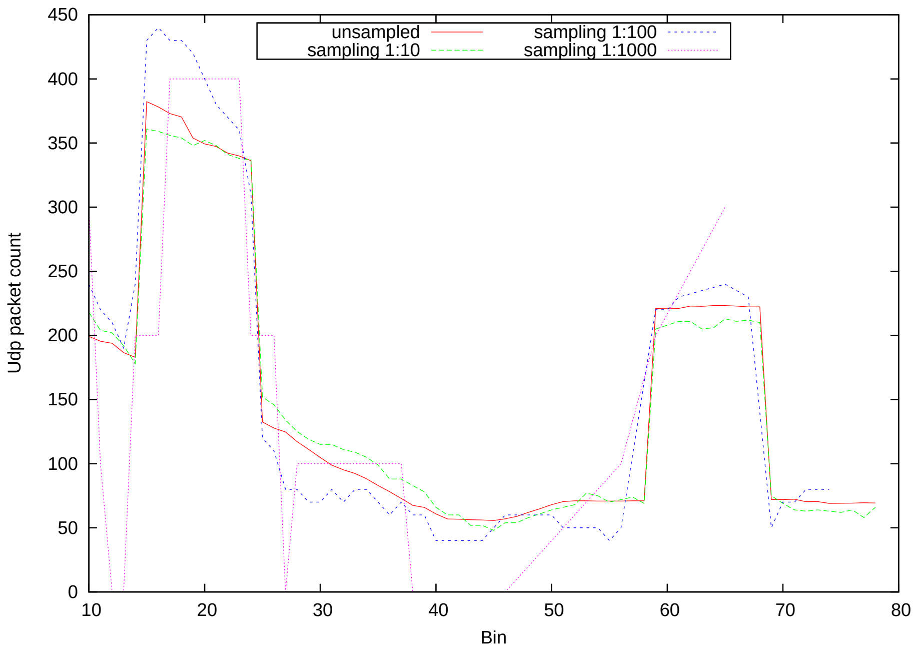


(b) UDP

Figure 5.11: Destination port entropy (baseline)- DARPA MIT 1999 week 2, Wednesday

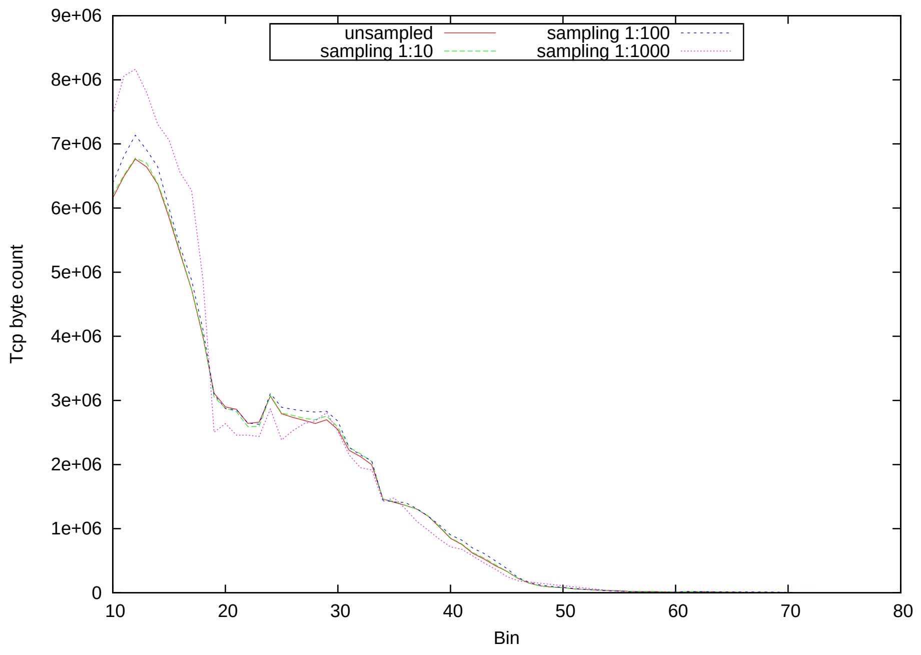


(a) TCP

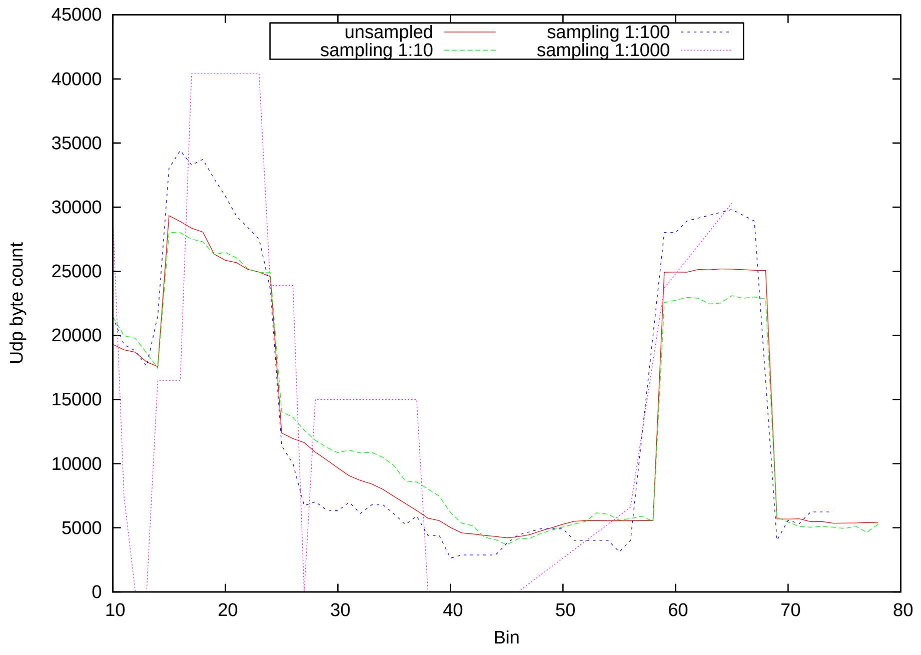


(b) UDP

Figure 5.12: Packet count (baseline) - DARPA MIT 1999 week 2, Wednesday

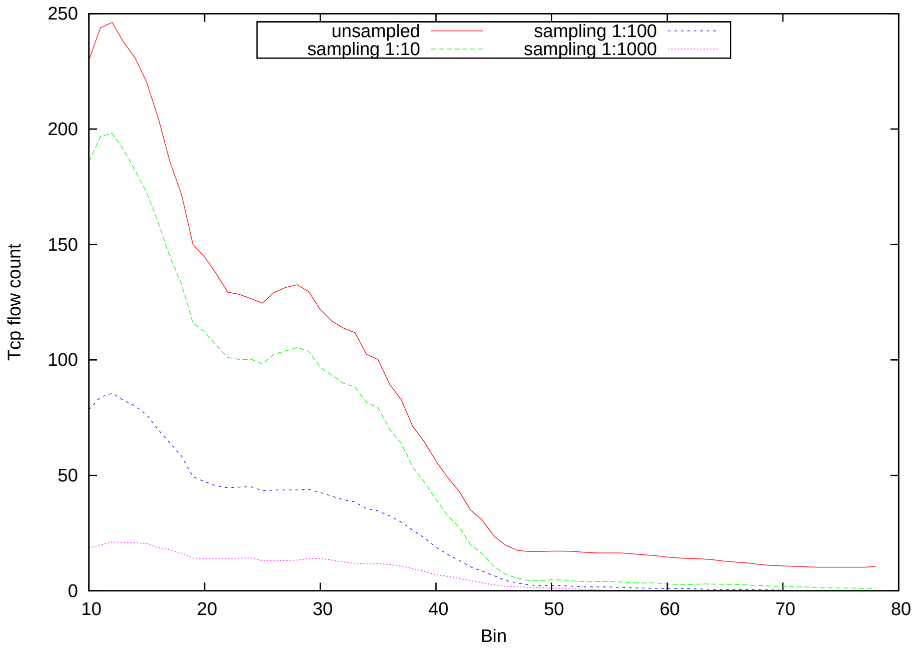


(a) TCP

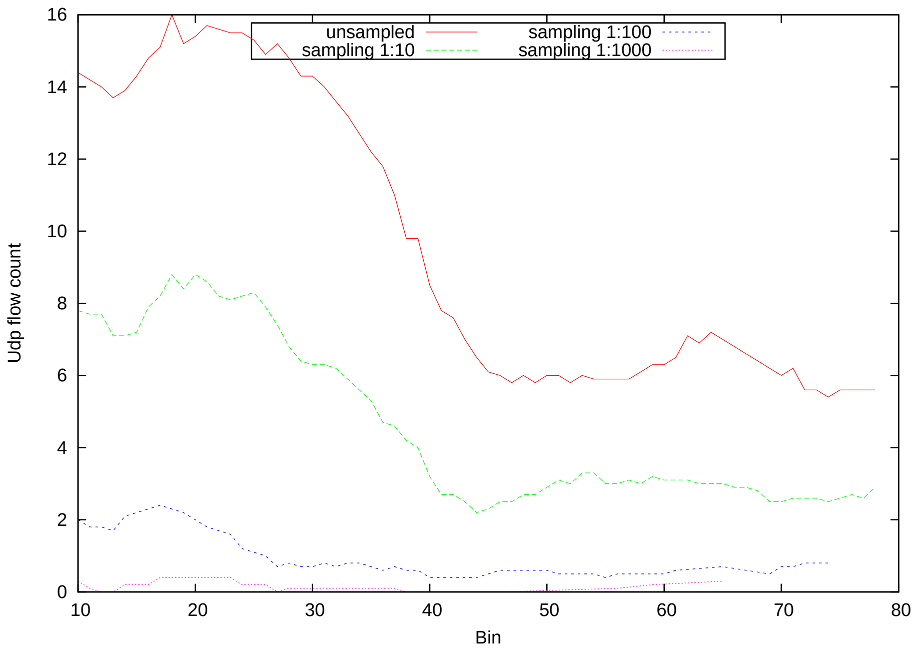


(b) UDP

Figure 5.13: Byte count (baseline) - DARPA MIT 1999 week 2, Wednesday



(a) TCP



(b) UDP

Figure 5.14: Flow count (baseline) - DARPA MIT 1999 week 2, Wednesday

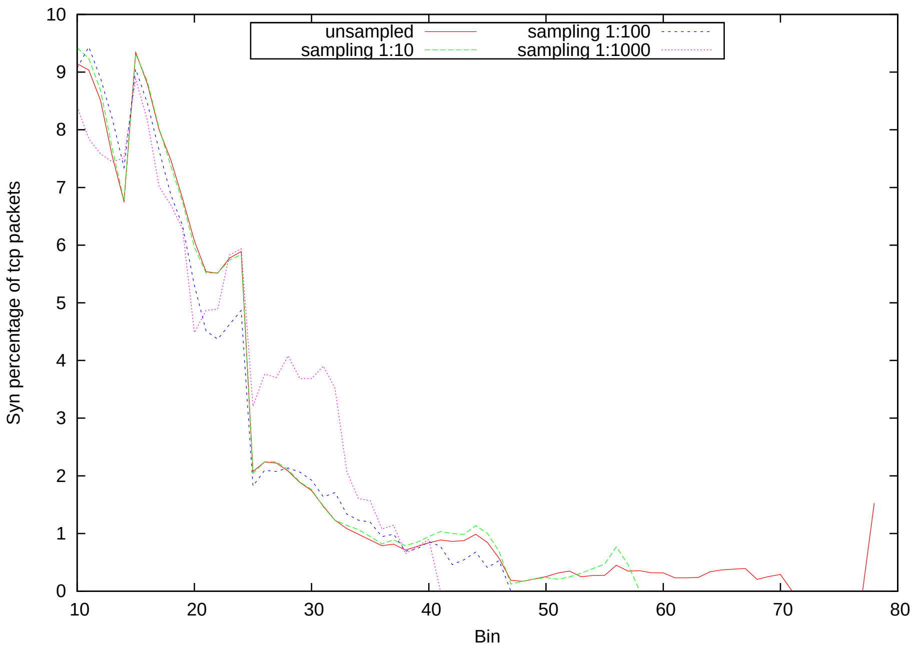
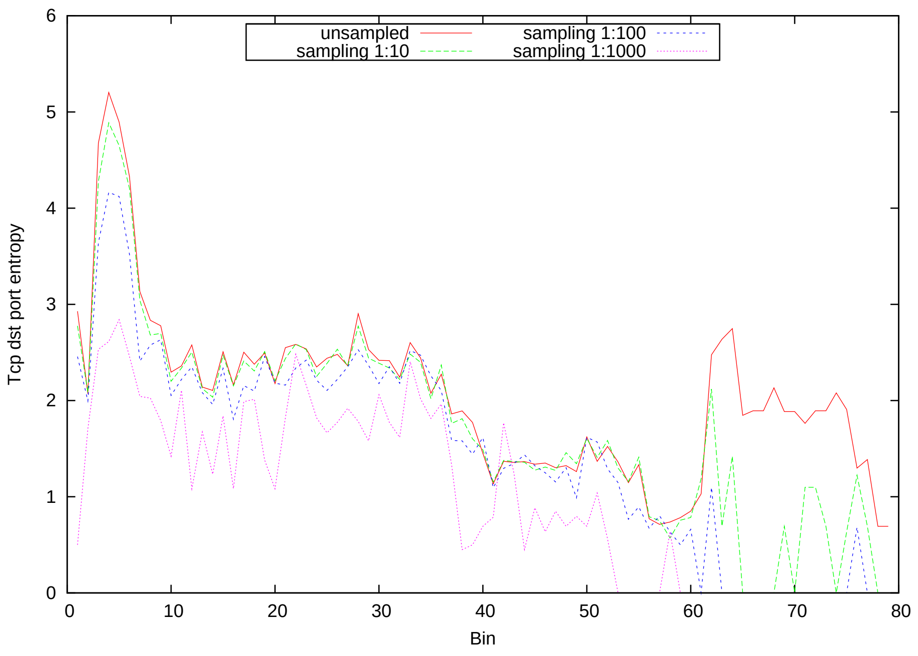
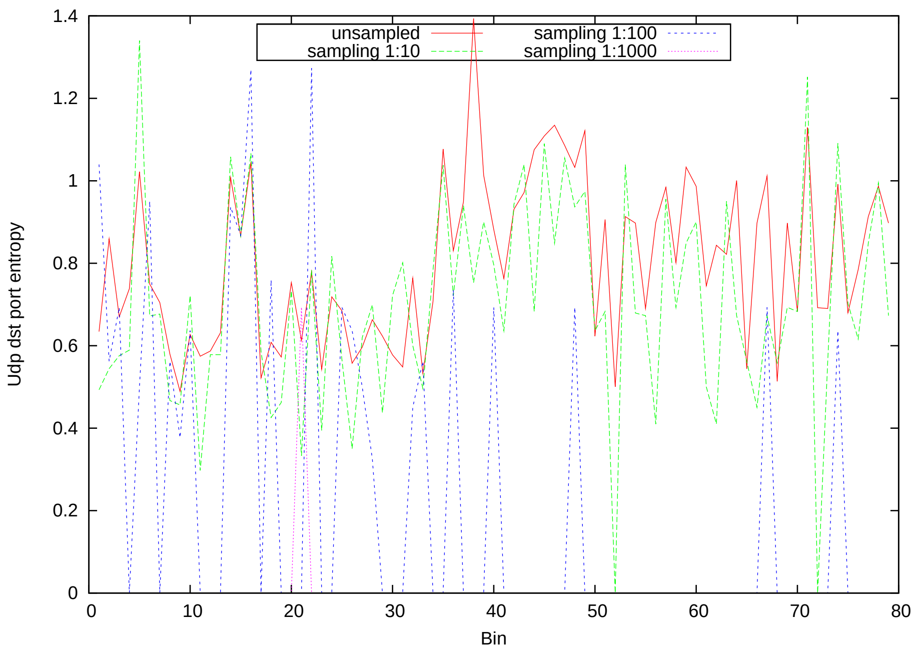


Figure 5.15: SYN percentage (baseline) - DARPA MIT 1999 week 2, Wednesday

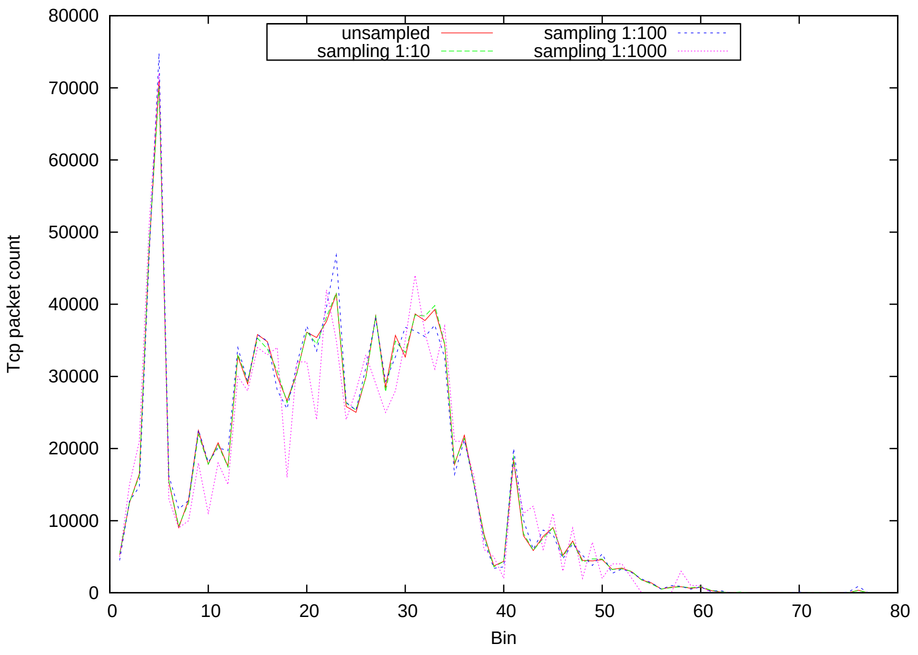


(a) TCP

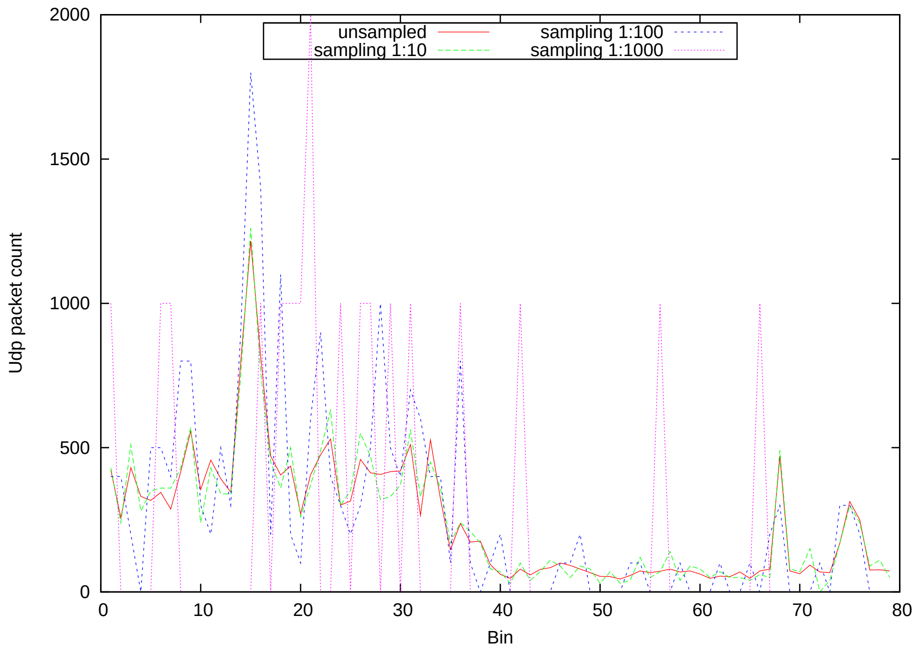


(b) UDP

Figure 5.16: Destination port entropy - DARPA MIT 1999 week 4, Monday

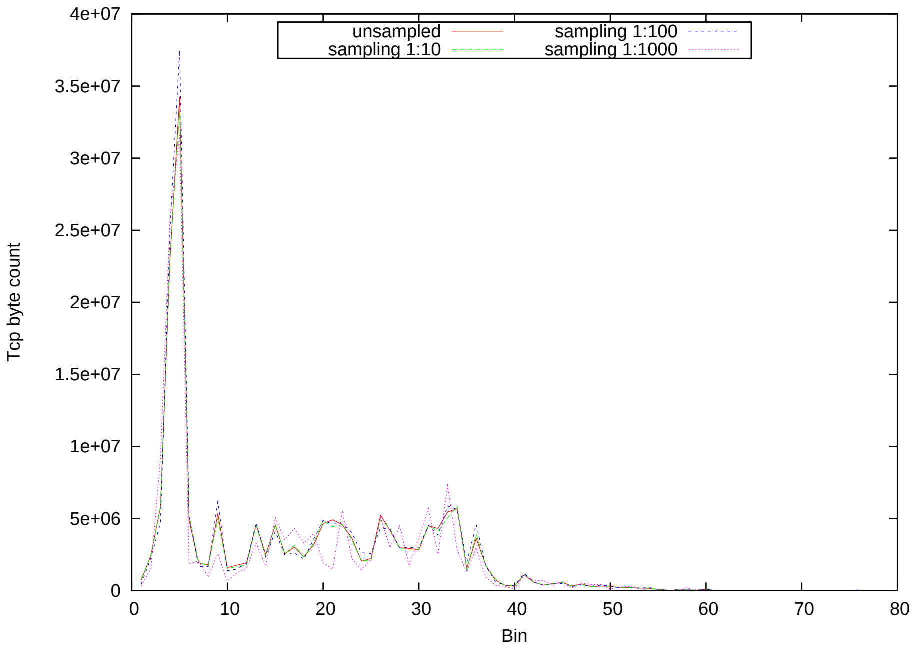


(a) TCP

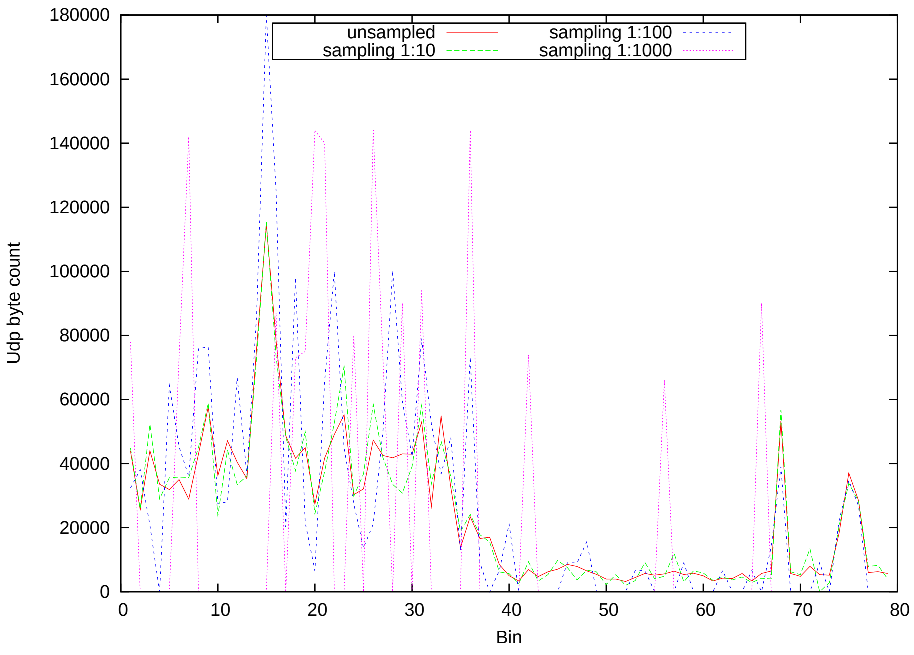


(b) UDP

Figure 5.17: Packet count - DARPA MIT 1999 week 4, Monday

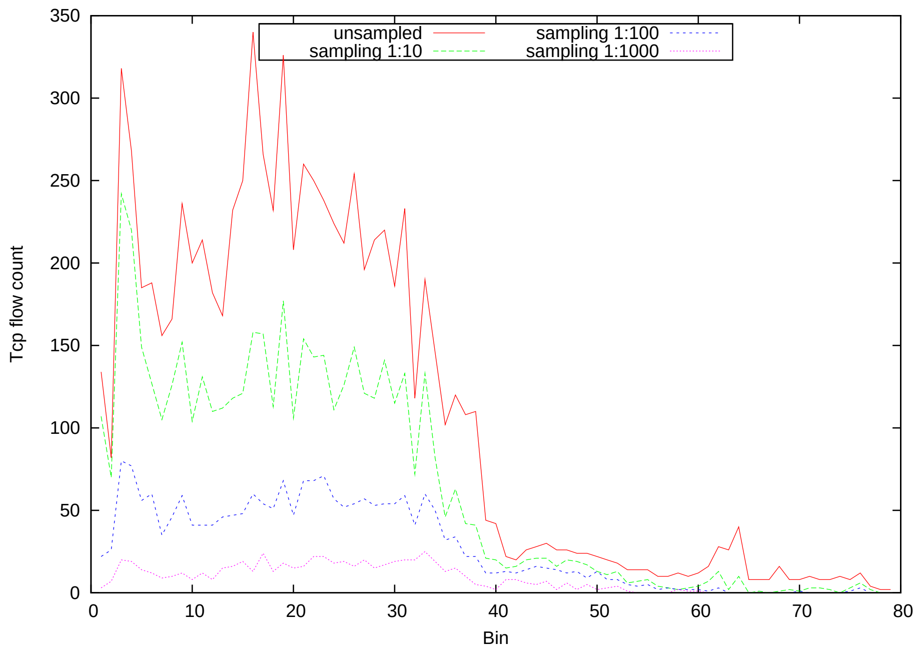


(a) TCP

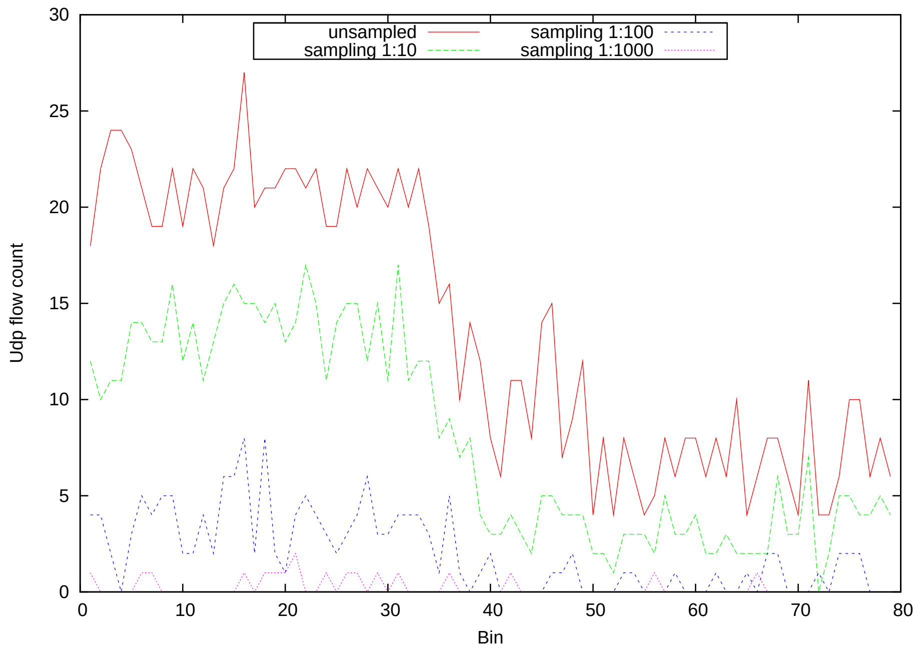


(b) UDP

Figure 5.18: Byte count - DARPA MIT 1999 week 4, Monday



(a) TCP



(b) UDP

Figure 5.19: Flow count - DARPA MIT 1999 week 4, Monday

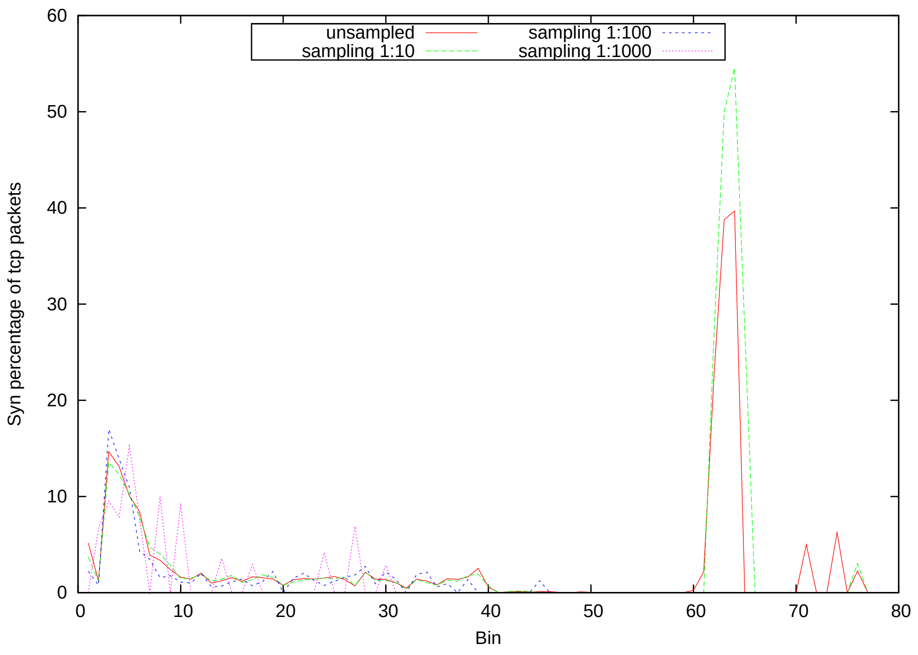
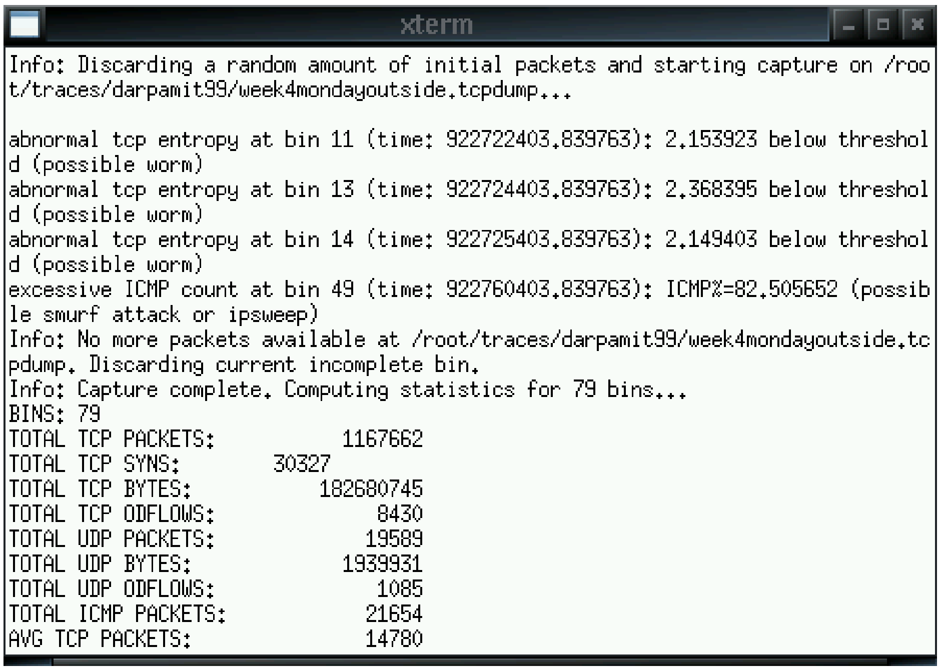


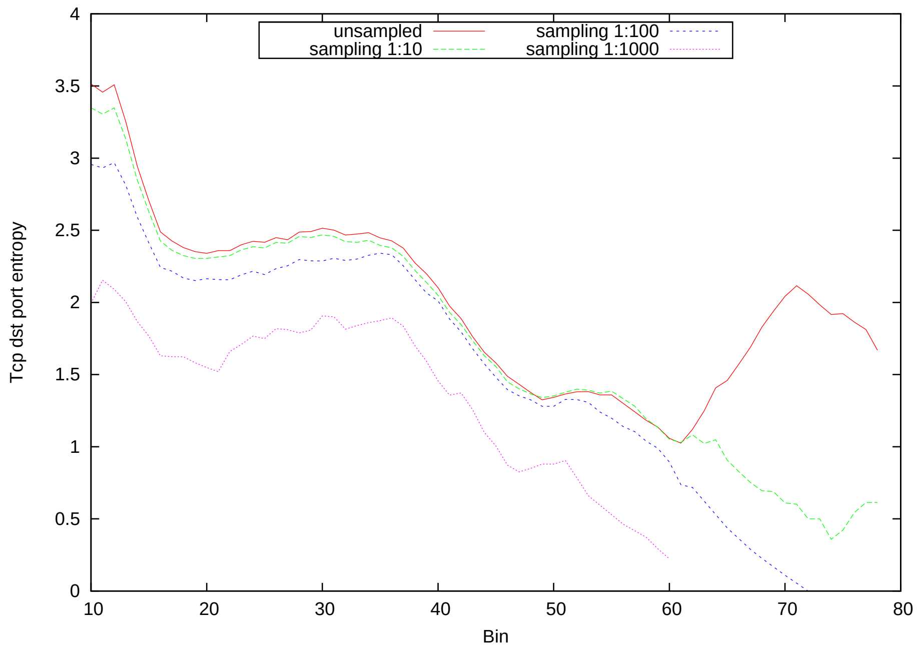
Figure 5.20: SYN percentage - DARPA MIT 1999 week 4, Monday



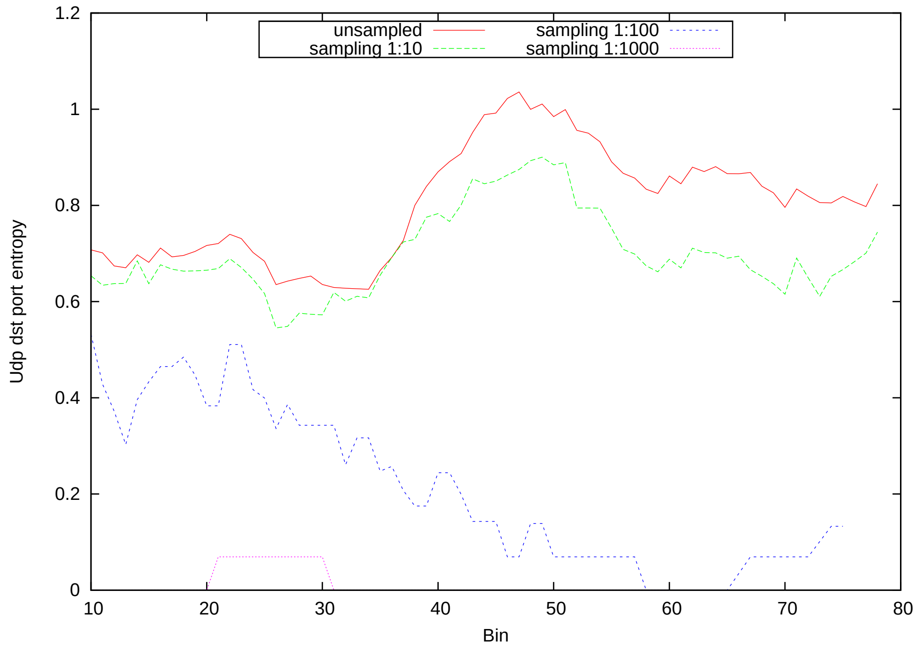
```
Info: Discarding a random amount of initial packets and starting capture on /roo
t/traces/darpamit99/week4mondayoutside.tcpdump...

abnormal tcp entropy at bin 11 (time: 922722403.839763): 2.153923 below threshol
d (possible worm)
abnormal tcp entropy at bin 13 (time: 922724403.839763): 2.368395 below threshol
d (possible worm)
abnormal tcp entropy at bin 14 (time: 922725403.839763): 2.149403 below threshol
d (possible worm)
excessive ICMP count at bin 49 (time: 922760403.839763): ICMP%=82.505652 (possib
le smurf attack or ipsweep)
Info: No more packets available at /root/traces/darpamit99/week4mondayoutside.tc
pdump. Discarding current incomplete bin.
Info: Capture complete. Computing statistics for 79 bins...
BINS: 79
TOTAL TCP PACKETS:           1167662
TOTAL TCP SYNS:        30327
TOTAL TCP BYTES:           182680745
TOTAL TCP ODFLOWS:              8430
TOTAL UDP PACKETS:             19589
TOTAL UDP BYTES:             1939931
TOTAL UDP ODFLOWS:              1085
TOTAL ICMP PACKETS:            21654
AVG TCP PACKETS:               14780
```

Figure 5.21: Screenshot - DARPA MIT 1999 week 4, Monday

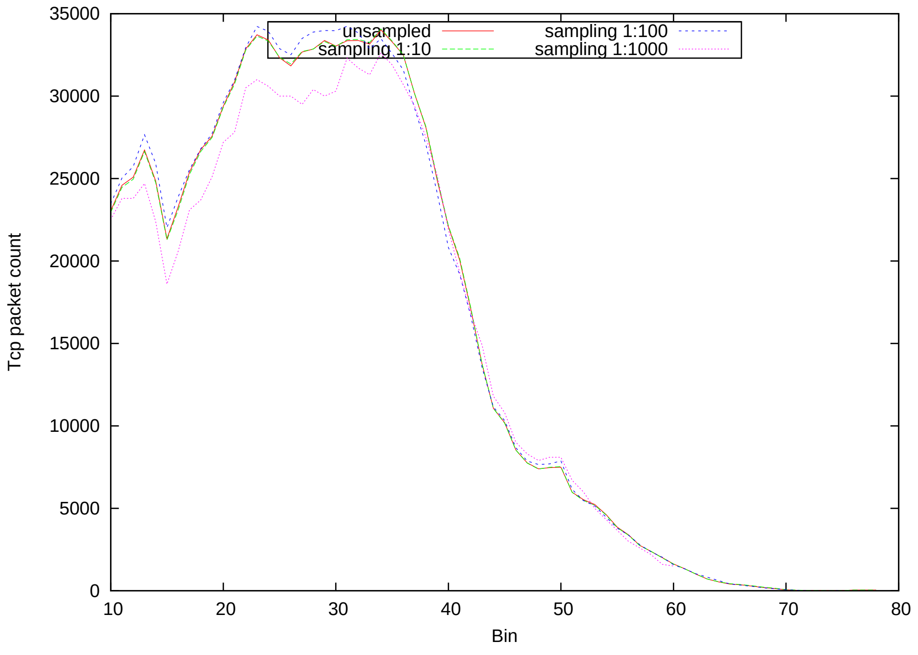


(a) TCP

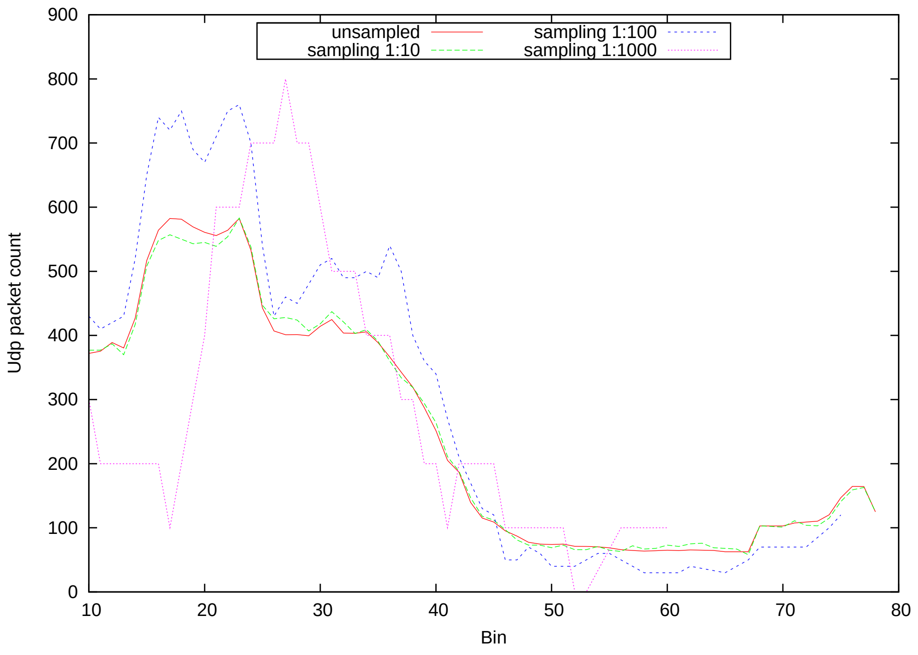


(b) UDP

Figure 5.22: Destination port entropy (baseline) - DARPA MIT 1999 week 4, Monday

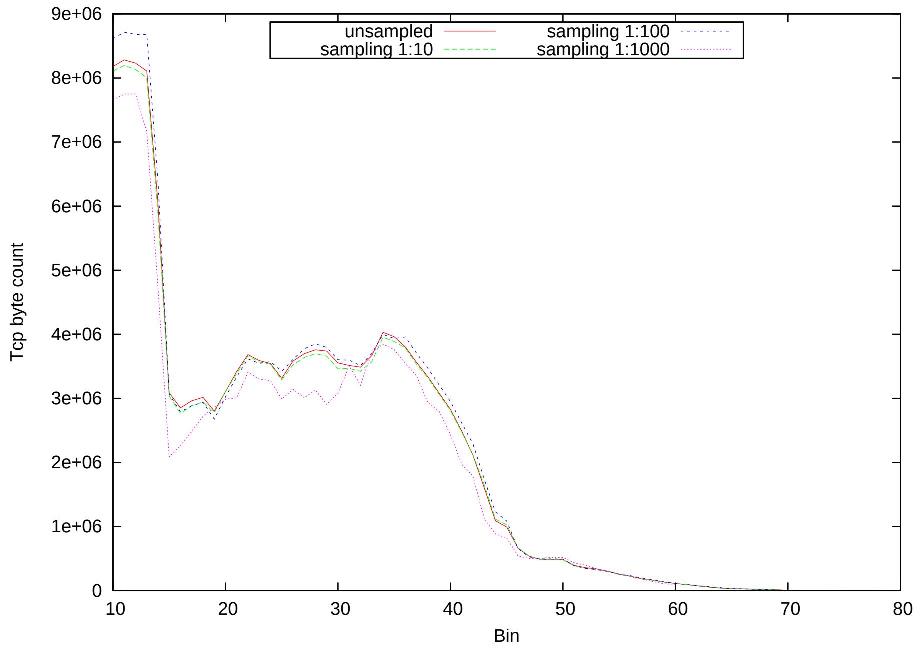


(a) TCP

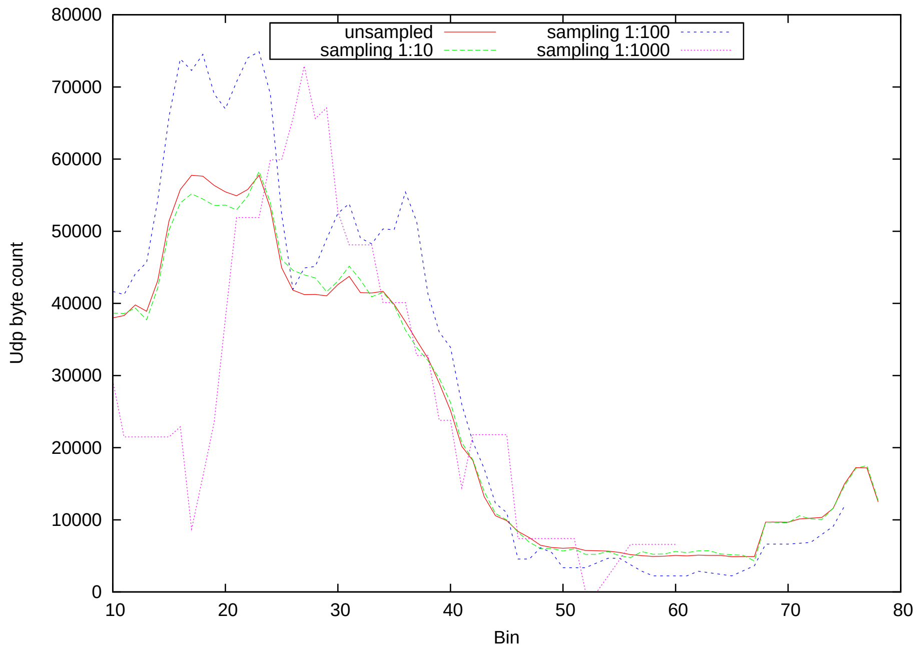


(b) UDP

Figure 5.23: Packet count (baseline) - DARPA MIT 1999 week 4, Monday

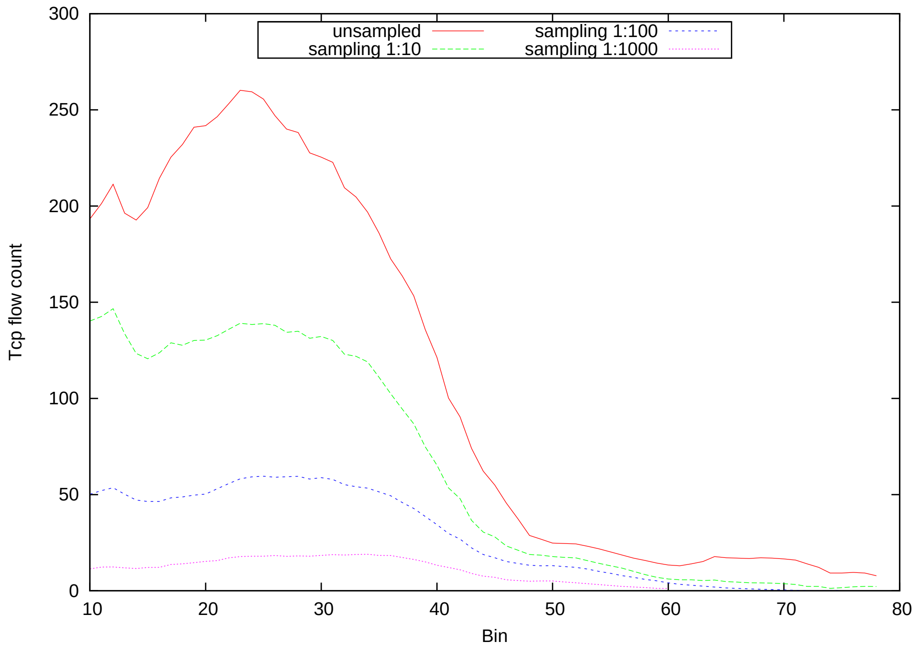


(a) TCP

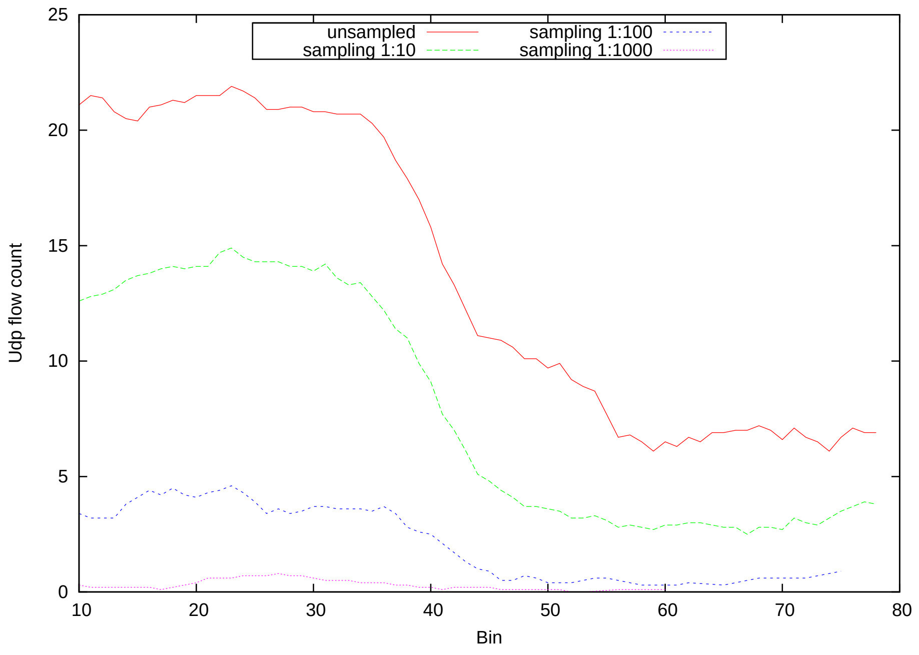


(b) UDP

Figure 5.24: Byte count (baseline) - DARPA MIT 1999 week 4, Monday



(a) TCP



(b) UDP

Figure 5.25: Flow count (baseline) - DARPA MIT 1999 week 4, Monday

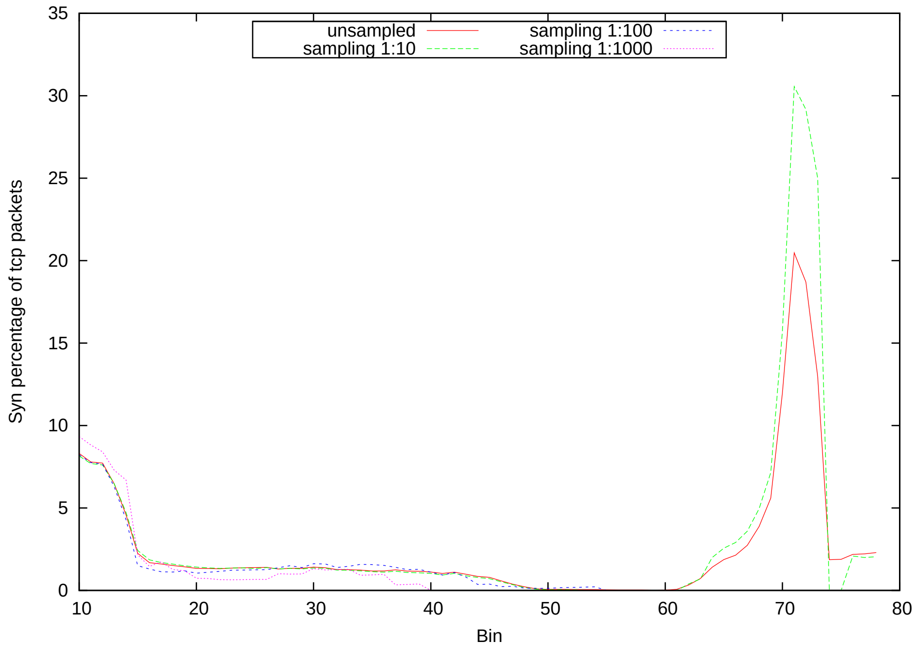


Figure 5.26: SYN percentage (baseline) - DARPA MIT 1999 week 4, Monday

As expected, entropy provided a good measure of the variability of destination port addresses and thus allowed to detect large probes/portscans, such as the satan attack clearly visible as a spike at bin 15 in Figures 5.5a and 5.5b. There are also spikes in the packet counts, especially in UDP, but they do not show anything by themselves, the spike could be benign and due to a sporadic large file transfer. Incidentally, the TCP byte count does not show signs of anomalies, while the UDP again reports a spike. The flow counts are also next to useless because they naturally exhibit a lot of variation throughout the trace duration. Another indicator that something strange happened at bin 15 is the high SYN percentage; it was not a true SYN flood however and this is a good example of a misclassification.

The screenshot of Figure 5.10 shows the detection of the probing attack, but since it is preceded by a sudden drop of the number of connections and traffic, it also shows an erroneous worm detection, a case of a false positive.

Finally, the low usage of network after the work hours (past bin 40) renders detection ineffective because even a small alteration of a couple packets causes a substantial change in its distribution. To avoid more false positives, the alerts are only issued when the traffic levels are high. Therefore, the entropy peaks between bin 45 and 65 are completely meaningless.

As for the effects of sampling, they only become noticeable for sampling fractions in the range of 1:100 packets. For 1:1000 the values no longer follow the unsampled behavior at all. But for UDP, even 1:100 is sometimes too aggressive, because there are extremely few packets to sample in first place. This effect does not occur when UDP traffic is significant, i.e., at least one order of magnitude superior to the sampling period.

The challenge for the second trace was to see if the stealthy portscans could be detected, since they consist in single attempts to connect to a very small group of well-known services. It turned out to be impossible to detect them because they simply lack volume to stand out in the entropy graphs of Figure 5.16. These types of stealthy attacks would require to track that the same origin IP address was used to establish connections to various services within a relatively small timeframe and that such connections contained no actual data. This only works in small networks when the IDS knows beforehand which services exist in the network.

Other two distinct issues are present in that trace, namely the initial spike in bins 4-6 that is just ordinary, albeit unusually high, traffic and that even if it was a real intrusion it would still go undetected because it occurred in the beginning of the capture while the initial baseline was still being computed. It also has the undesired side effect of creating an abnormal baseline (see Figure 5.22a), meaning that the following traffic, which is normal, will cause a false positive because there appears to be a sudden drop in the entropy, similar to the effect of worms. This is the explanation for the screenshot in Figure 5.21.

Sampling degradation in TCP is small until 1:100 sampling rates but UDP already shows serious problems at 1:100 due to low traffic. Again, this is a consequence of the trace used where only one bin contains more than 1000 UDP packets and the average is 500, which means that when subject to a 1:100 sampling rate, most bins collect no more than 5 packets.

The final bins corresponding to the late afternoon and night again contain little traffic and thus are inconclusive and disregarded.

While having timer-based binning is much more convenient for comparisons, counter-based binning was also implemented and tested, having obtained similar results. This can be seen by comparing Figure 5.27 and Figure 5.5. Since most of the literature examples and routers themselves use time-based binning, no further tests were conducted.

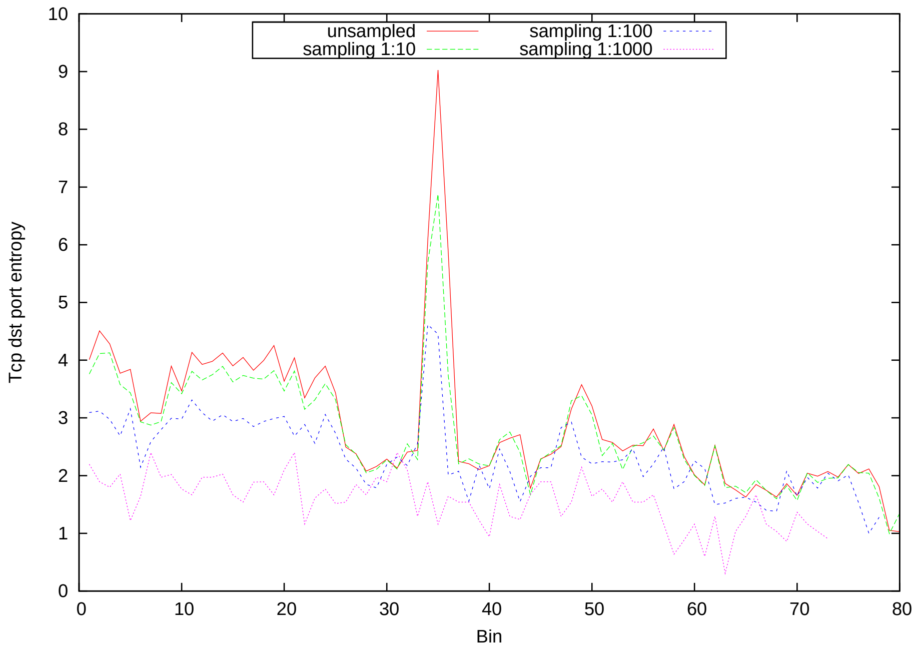
5.3 Sampling with Snort

The methodology consisted in using Snort to analyze the DARPA MIT 1999 traces (`week2wednesdayoutside.tcpdump/week4mondayoutside.tcpdump`) as they are - unsampled - and then repeat the procedure for the sampled counterparts. Only the number of alerts detected in the unsampled trace was considered; afterall the object of study is not Snort's ability per se, but the impact of sampling on its efficacy. The version of Snort used was 2.8.0.1 with the default configuration. Table 5.1 contains the averaged results after 20 runs each. As explained in the end of Section 4.1, deterministic (timer-based) sampling could not be implemented.

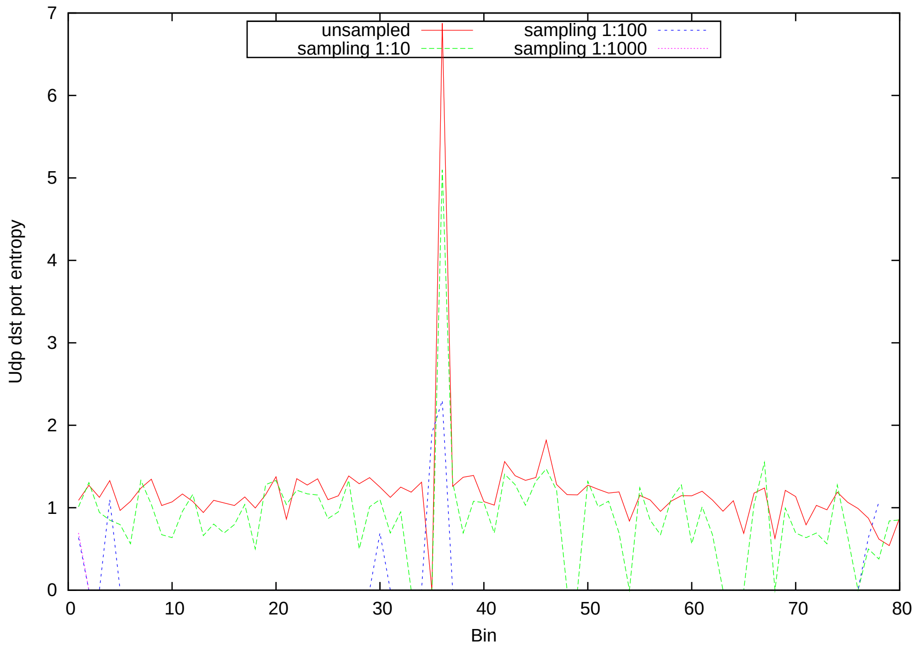
Table 5.1: Snort alerts

	week2 trace	week4 trace
unsampled	137	52
determinisitic (counter-based) 1:10	4	0
probabilistic (uniform random) 1:10	6	1
determinisitic (counter-based) 1:100	1	0
probabilistic (uniform random) 1:100	1	0
determinisitic (counter-based) 1:1000	1	0
probabilistic (uniform random) 1:1000	1	0

Snort was clearly affected by sampling, even at low rates. Which sampling method was used did not make much difference either.



(a) TCP



(b) UDP

Figure 5.27: Use of counter-based binning - DARPA MIT 1999 week 2, Wednesday

5.4 About the traces

It proved impossible to find traces with all the desired qualities: from high-speed links, in `pcap` format with full headers, with labeled intrusions that are large enough to affect global metrics. This posed a major problem to test the prototype but was partially overcome by using different traces for different tests.

Most IDS related traces consist on particular packets containing malicious payloads, suitable for signature detection. For anomaly detection the payloads are ignored, and attacks are only spotted if there is a change over time on the traffic patterns. In particular, the implemented detection method based on the entropy of the destination port per protocol can only target worms that are propagating quickly, DoS attempts or portscans, since all these affect the traffic distribution heavily.

This problem was also found in the literature and the common solution is to use artificially modified traces, either by reconstructing packets from flow summaries, inserting additional traffic (typically malicious) or using simulations. Since the purpose of this thesis is not to implement an actual IDS system but to study its performance when sampling is used, using such approximations is an expedient way to obtain valuable results. This does not eliminate the need for validation in a real scenario, later on.

5.4.1 MAWI traces

These traces were obtained through `http://www.DatCat.org` and represent A Day in the Life of the Internet as seen by MAWI/WIDE in Japan[9]. From the collection, two data objects representing peak and low traffic hours, repectively, were randomly selected: `200803172345.dump` and `200803201630.dump`; their summaries are shown in Table 5.2.

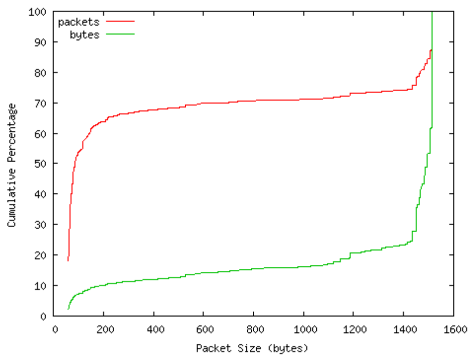
These recent traces represent typical Internet traffic passing through a 150Mbps link and contain enough quantity and diversity of packets and flows to make them interesting for use in testing the different sampling methods.

Table 5.2: Summary of MAWI traces

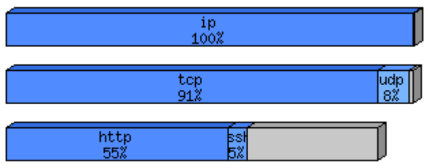
Traffic Trace Info
DumpFile: 200803172345.dump
StartTime: Mon Mar 17 23:45:00 2008
EndTime: Tue Mar 18 00:00:00 2008
TotalTime: 899.70 seconds
TotalCapSize: 805.40MB
CapLen: 96 bytes
# of packets: 14405984 (6932.26MB)
AvgRate: 64.63Mbps stddev:9.81M

IP flow (unique src/dst pair) Information
# of flows: 758317 (avg. 19.00 pkts/flow)
Top 10 big flow size (bytes/total in %):
8.0% 3.2% 1.3% 1.2% 1.0%
0.9% 0.9% 0.8% 0.8% 0.7%

Packet Size Distribution (including MAC headers)



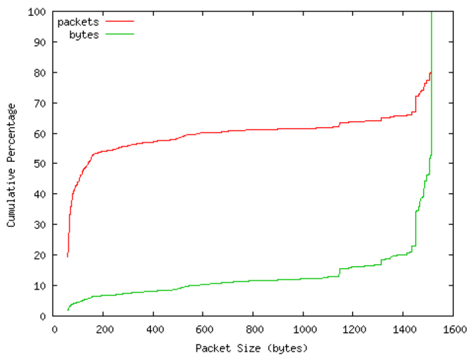
Protocol Breakdown



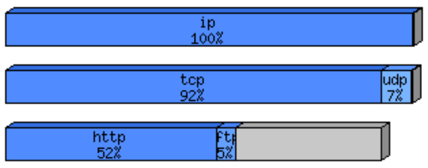
Traffic Trace Info
DumpFile: 200803201630.dump
StartTime: Thu Mar 20 16:30:01 2008
EndTime: Thu Mar 20 16:45:00 2008
TotalTime: 899.41 seconds
TotalCapSize: 828.74MB
CapLen: 96 bytes
# of packets: 15345258 (9389.90MB)
AvgRate: 87.59Mbps stddev:17.35M

IP flow (unique src/dst pair) Information
# of flows: 681777 (avg. 22.51 pkts/flow)
Top 10 big flow size (bytes/total in %):
10.9% 3.2% 3.1% 2.3% 1.7%
1.7% 1.6% 1.6% 1.6% 1.5%

Packet Size Distribution (including MAC headers)



Protocol Breakdown



5.4.2 DARPAMIT99 traces

These traces were used in 1999 for the DARPA/MIT Intrusion Detection Evaluation[41] and contain a wide variety of both labeled and unlabeled attacks, though they are primarily designed to test signature-based NIDS and HIDS. The majority of anomalies contained pertain to software exploits and erratic usage patterns of network services, and are unsuitable for the prototype at hand. However, some of the attacks are so prominent that can be effectively detected, as shown in Figure 5.10 which displays the output of the prototype during one of the executions.

The traces used for testing were `week2wednesdayoutside.tcpdump` and `week4mondayoutside.tcpdump`, consisting in nearly one day worth of traffic in a 100Mbit/s link, with little average traffic.

A summary of the attacks as provided by authors of the traces is shown in Tables 5.3 and 5.4. The attacks that are of interest for this thesis consist in portscans and probes. An extended description of these and the other attacks can be found online[1]. A special mention goes to the smurf and ipsweep attack because they use ICMP packets which are discarded by the BPF filter, and thus is not normally detected. This type of attack does not occur normally in large networks because broadcast addresses are normally inaccessible from the outside preventing smurf attacks and most firewalls already possess mechanisms that detect such simple ipsweeps but, for completion sake, an exception to the assumption that all packets are either TCP or UDP was made and one test run performed whose results are displayed in Figures 5.28 and 5.29 and Figures 5.30 and 5.31.

Table 5.3: List of existing attacks - DARPA MIT 1999 week 2, Wednesday

Time	Name/Description	Detectable without DPI/signatures
12:02:13	satan	yes
13:44:18	mailbomb	no
15:25:18	perl (failed)	no
20:17:10	ipsweep	yes
23:23:00	eject (console)	no
23:56:14	crashiis	no

[1]`http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/docs/attackDB.html`

Table 5.4: List of existing attacks - DARPA MIT 1999 week 4, Monday

Time	Name/Description	Detectable without DPI/signatures
08:18:35	ps	no
08:48:12	sendmail	no
09:36:23	sshtrojanInstall	no
11:22:43	xsnoop	no
11:45:12	snmpget	no
11:47:16	guesstelnet	no
12:22:15	stealthy portsweep	maybe
13:58:16	ftpwrite	no
16:27:10	stealthy portsweep	maybe
18:24:19	secret	no
21:34:16	smurf	yes

The ipsweep occurs at bin 45, which is visible as a spike in Figure 5.28 and correctly reported as shown in the screenshot of Figure 5.29. It is detected across all sampling rates.

The smurf attack causes the spike at bin 49 in Figure 5.30. Again it is correctly detected across all sampling rates.

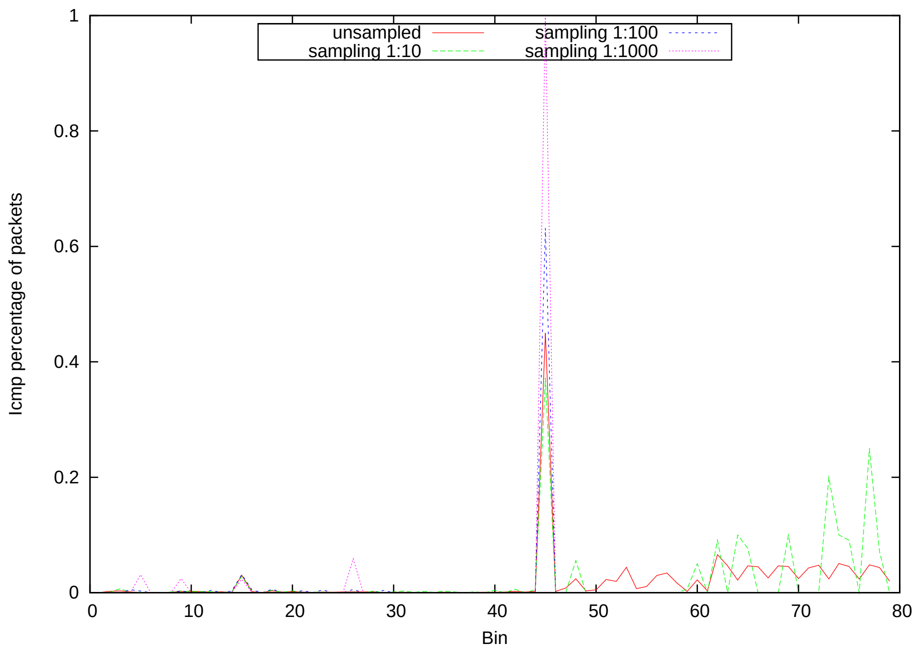
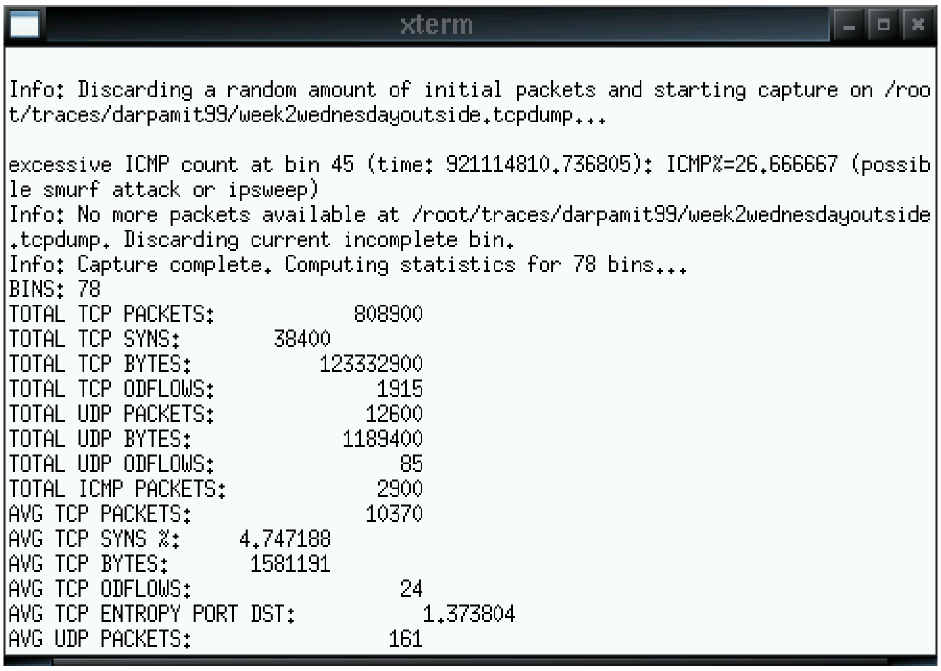


Figure 5.28: ICMP - DARPA MIT 1999 week 2, Wednesday



```
Info: Discarding a random amount of initial packets and starting capture on /roo
t/traces/darpamit99/week2wednesdayoutside.tcpdump...

excessive ICMP count at bin 45 (time: 921114810.736805): ICMP%=26.666667 (possib
le smurf attack or ipsweep)
Info: No more packets available at /root/traces/darpamit99/week2wednesdayoutside
.tcpdump. Discarding current incomplete bin.
Info: Capture complete. Computing statistics for 78 bins...
BINS: 78
TOTAL TCP PACKETS:          808900
TOTAL TCP SYNS:      38400
TOTAL TCP BYTES:          123332900
TOTAL TCP ODFLOWS:            1915
TOTAL UDP PACKETS:           12600
TOTAL UDP BYTES:           1189400
TOTAL UDP ODFLOWS:              85
TOTAL ICMP PACKETS:           2900
AVG TCP PACKETS:             10370
AVG TCP SYNS %:     4.747188
AVG TCP BYTES:      1581191
AVG TCP ODFLOWS:                24
AVG TCP ENTROPY PORT DST:         1.373804
AVG UDP PACKETS:               161
```

Figure 5.29: Screenshot (ipsweep) - DARPA MIT 1999 week 2, Wednesday

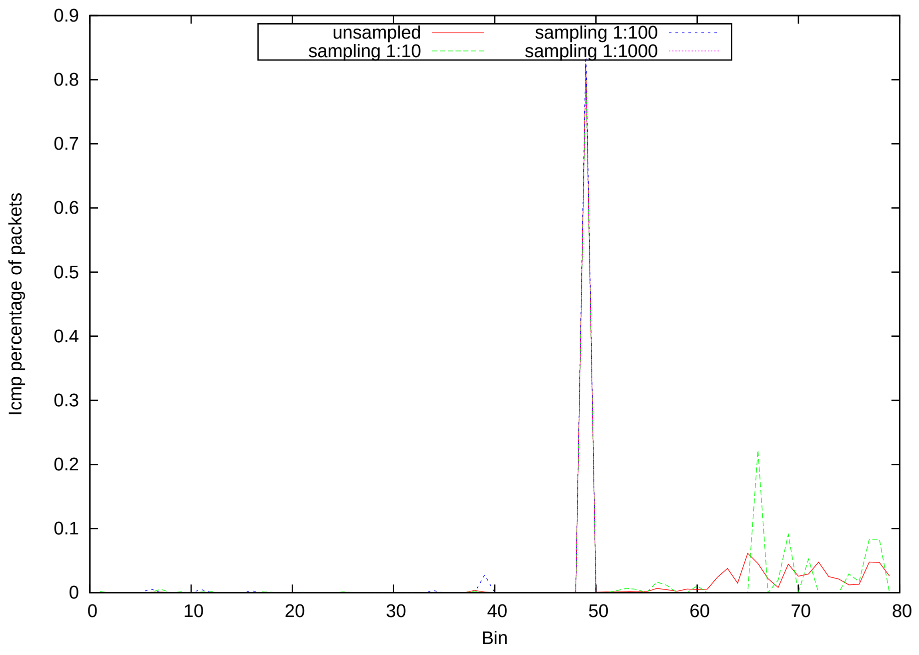
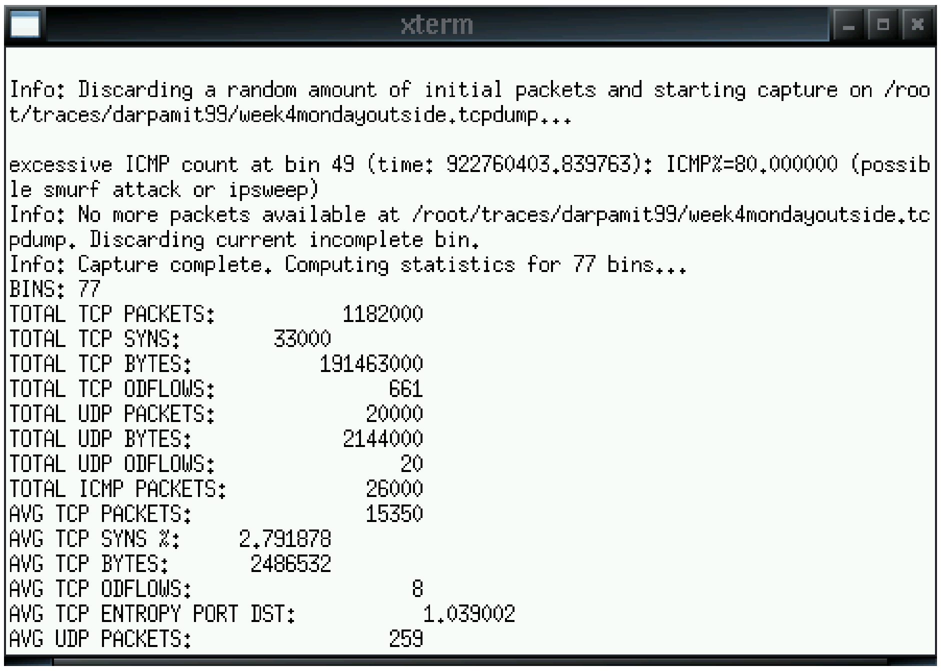


Figure 5.30: ICMP - DARPA MIT 1999 week 4, Monday



```
Info: Discarding a random amount of initial packets and starting capture on /roo
t/traces/darpamit99/week4mondayoutside.tcpdump...

excessive ICMP count at bin 49 (time: 922760403.839763): ICMP%=80.000000 (possib
le smurf attack or ipsweep)
Info: No more packets available at /root/traces/darpamit99/week4mondayoutside.tc
pdump. Discarding current incomplete bin.
Info: Capture complete. Computing statistics for 77 bins...
BINS: 77
TOTAL TCP PACKETS:          1182000
TOTAL TCP SYNS:        33000
TOTAL TCP BYTES:          191463000
TOTAL TCP ODFLOWS:              661
TOTAL UDP PACKETS:            20000
TOTAL UDP BYTES:            2144000
TOTAL UDP ODFLOWS:               20
TOTAL ICMP PACKETS:           26000
AVG TCP PACKETS:              15350
AVG TCP SYNS %:     2.791878
AVG TCP BYTES:       2486532
AVG TCP ODFLOWS:                  8
AVG TCP ENTROPY PORT DST:           1.039002
AVG UDP PACKETS:                259
```

Figure 5.31: Screenshot (smurf) - DARPA MIT 1999 week 4, Monday

Chapter 6

Conclusions and future work

6.1 Conclusions

In order to deploy an efficient IDS in the backbone of the Internet, better metrics that are robust to sampling must be found. Entropy provided good results, but calculating it by the definition and only applying it to the destination ports leaves a lot of information out, thus making the classification of the anomaly very difficult and prone to error. This conclusion is backed up by the false positives found during the tests, where no simple way could be found to differentiate between a change in the usage of the network services involved or an attack.

The other limitation is the actual types of attack that can be detected; they need to be large enough to affect the overall distribution of traffic, meaning that a lot of smaller attacks can go unnoticed. However, these smaller attacks are typically less dangerous and with less consequences for the network itself - and this is the primary goal of moving IDS upstream, to protect the core from collapse - and can be detected using conventional IDS on the gateways of the fringes.

As for sampling, the results are virtually independent of the method used, which was expected due to highly random nature of high-level traffic aggregated from various sources. Possible correlations upon one source get dissipated as soon as traffic from other sources gets interleaved. From the three methods tested, deterministic (counter-based) is the simplest one to implement and performs well even in its most basic form of constant sampling period. This requires a single counter and can be easily implemented on any computer system; in fact, it is already present on most routers.

There is, however, one aspect that can be improved regarding deterministic sampling which is to include some form of protocol awareness. It has been shown during tests that having a separate pipeline for each protocol does not improve the error for a given sampling rate, so the solution is to adjust it independently for each protocol, namely to adopt a lower rate for UDP since it comprises much less traffic than TCP. This gains importance when moving into a distributed system made up of probes that report to a central collector: each probe can be set up individually and target a specific protocol.

Regarding the sampling rates themselves, it becomes clear that, using a simple entropy metric, anything with a period above 1:100 will introduce a substantial error, which added to the proneness for false positives discourages its use. On the other hand, a more conservative sampling period of 1:10 can have errors as low as 4%, meaning that there is no reason not to use it, even with simple metrics. The only caveat is that the total amount of packets traversing the link during each bin duration should be at least one order of magnitude higher than number of packets discarded due to sampling. For example, if using 1:100 sampling (where 1 packet is sampled and the next 99 discarded) there should be at least 1000 packets available per bin, or there may not be enough samples to make them representative.

Finally, applying sampling to signature detection systems also proved unfruitful, as expected. The number of false negatives increased drastically even with low sampling rates. In order to improve the scalability of these systems, solutions other than sampling must be found.

6.2 Summary of contributions

The main contribution of this thesis is the development of an extensible framework for the evaluation and testing of different sampling methods and different intrusion detection algorithms. Most of the drawbacks related to false negatives and false positives faced by current IDS and confirmed through testing with the current implementation could be overcome with more intelligent algorithms that use more of the information provided by the framework, such as flows.

The prototype is fully functional and can perform live captures or read pre-captured traces. It can detect most traffic distributions anomalies and classify them. It can also easily be expanded and improved in a modular fashion. Finally, it is freely released under a Berkeley Software Distribution (BSD) license.

Also worth of notice is the summary of the state of the art in 2008 regarding the applicability of IDS to high-speed networks and the advantages and limitation of using sampling. This would allow any interested person to quickly get a detailed view and move on into putting their ideas to practice.

6.3 Future work

The directions for future improvements are endless, but some of the more immediate ones include integration with NetFlow/IPFIX/PSAMP to allow for a distributed system to be implemented. Standardizing the information transmitted in templates would also contribute to further simplify the comparison of results.

Another consequence of using flow information would be different granularity levels for different deployment scenarios or different alert states.

Integrating Snort would also boost the framework capabilities on detecting the majority of threats, even if such low-level analysis could only be performed when the higher-level indicators point toward suspicious activity.

Another important step would be to simulate worm propagation patterns and inject the generated packets into real-world traces to tune the detection parameters.

Finally, implementing some of the more complex entropy-based algorithms described in the papers composing the state of art would certainly reduce errors.

Bibliography

[1] Mark Allman, Vern Paxson, and Jeff Terrell. A brief history of scanning. In *IMCONF 2007 Proceedings*, 2007.

[2] James P. Anderson. Computer security threat monitoring and surveillance. Technical report, James P. Anderson Co, Box 42 Fort Washington, PA 19034, 1980.

[3] Stefan Axelsson. Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Dept. of Computer Engineering, Chalmers University of Technology, Sweden, Department of Computer Engineering Chalmers University of Technology Gteborg, Sweden, 2000.

[4] Rebecca Bace and Peter Mell. Intrusion detection systems. Technical report, Infidel, Inc and National Institute of Standards and Technology, 2001.

[5] Paul Barford and David Plonka. Characteristics of network traffic flow anomalies. In *IMCONF 2001 Proceedings*, 2001.

[6] James R. Binkley and Suresh Singh. An algorithm for anomaly-based botnet detection. In *USENIX SRUTI 06: 2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet*, 2006.

[7] Daniela Brauckhoff, Martin May, and Bernhard Plattner. Flow-level anomaly detection - blessing or curse? IEEE INFOCOM 2007, Student Workshop, May 2007.

[8] Daniela Brauckhoff, Bernhard Tellenbach, Arno Wagner, Martin May, and Anukool Lakhina. Impact of packet sampling on anomaly detection metrics. In *IMCONF 2006 Proceedings*, 2006.

[9] Kenjiro Cho. WIDE-TRANSIT 150 Megabit Ethernet Trace 2008-03-18 (Anonymized) (collection).

[10] Cisco Systems, Inc. http://www.cisco.com/en/us/prod/collateral/iosswrel/ps6537/ps6555/ps6601/prod_white_paper0900aecd80406232.html.

[11] Cisco Systems, Inc. http://www.cisco.com/en/us/docs/ios/12_0s/feature/guide/nfstatsa.html, 2008.

[12] Cisco Systems, Inc. http://www.cisco.com/en/us/products/ps6350/products_configuration_guide_chapter09186a00805e387d.html, 2008.

[13] Cisco Systems, Inc. http://www.cisco.com/en/us/products/ps6350/products_configuration_guide_chapter09186a00805e7c53.html, 2008.

[14] Cisco Systems, Inc. http://www.cisco.com/en/us/products/ps6441/products_configuration_guide_chapter09186a008055404a.html, 2008.

[15] Cisco Systems, Inc. http://www.cisco.com/en/us/products/ps6601/products_ios_protocol_group_home.html, 2008.

[16] Cisco Systems, Inc. http://www.cisco.com/en/us/technologies/tk543/tk812/technologies _white_paper0900aecd802a0eb9.html, 2008.

[17] Kimberly C. Claffy, George C. Polyzos, and Hans-Werner Braun. Application of sampling methodologies to network traffic characterization. *SIGCOMM Comput. Commun. Rev.*, 23:194–203, 1993.

[18] Herv Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 31(8):805–822, April 1999.

[19] D. E. Denning, D. L. Edwards, R. Jagannathan, T. F Lunt, and P.G. Neumann. A prototype IDES: A real-time intrusion detection expert system. Technical report, Computer Science Laboratory, SRI International, Menlo Park, California, 1987.

[20] Dorothy E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, SE-13(2):222–232, February 1987.

[21] Nick Duffield. http://www.ietf.org/internet-drafts/draft-ietf-psamp-framework-13.txt, June 2008.

[22] Endace Limited. http://www.endace.com/our-products/dag-network-monitoring-cards/ethernet, 2008.

[23] Cristian Estan, Ken Keys, David Moore, and George Varghese. Building a better NetFlow. In *SIGCOMM '04: Proceedings of the 2004 conference*, pages 245–256, 2004.

[24] Anup K. Ghosh and Aaron Schwartzbard. A study in using neural networks for anomaly and misuse detection. In *Proceedings of the 8th USENIX Security Symposium - SEC1999*, 1999.

[25] Yu Gu, Andrew McCallum, and Don Towsley. Detecting anomalies in network traffic using maximum entropy estimation. In *IMCONF 2005 Proceedings*, 2005.

[26] Mark Handley, Vern Paxson, and Christian Kreibich. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *Proceedings of the 10th USENIX Security Symposium*, AT&T Center for Internet Research at ICSI (ACIRI), International Computer Science Institute, Berkeley, CA 94704 and Institut für Informatik - Technische Universität München, 80290 München, Germany, 2001.

[27] Troy D. Hanson. http://uthash.sourceforge.net/, 2008.

[28] L. Todd Heberlein, Gian V. Dias, Karl N. Levitt, Biswanath Mukherjee, Jeff Wood, and David Wolber. A network security monitor. In *Proceedings of the IEEE Computer Society Symposium, Research in Security and Privacy*, pages 296–304, May 1990.

[29] Gianluca Iannaccone, Christophe Diot, Ian Graham, and Nick McKeown. Monitoring very high speed links. In *IMCONF 2001 Proceedings*, November 2001.

[30] Keisuke Ishibashi, Ryoichi Kawahara, Mori Tatsuya, Tsuyoshi Kondoh, and Shoichiro Asano. Effect of sampling rate and monitoring granularity on anomaly detectability. In *IEEE Global Internet Symposium, 2007*, pages 25–30, May 2007.

[31] Wolfgang John and Sven Tafvelin. Analysis of internet backbone traffic and header anomalies observed. In *IMCONF 2007 Proceedings*, August 2007.

[32] Srikanth Kandula, Dina Katabi, Matthias Jacob, and Arthur Berger. Botz4sale: Surviving organized ddos attacks that mimic flash crowds. In *Proceedings of the USENIX NSDI 05: 2nd Symposium on Networked Systems Design & Implementation*, pages 287–300, 2005.

[33] Ramana Rao Kompella, Sumeet Singh, and George Varghese. On scalable attack detection in the network. In *IMCONF 2004 Proceedings*, pages 187–200, 2004.

[34] Jack Koziol. *Intrusion Detection with Snort*. Sams Publishing, 2003.

[35] Anukool Lakhina, Mark Crovella, and Christophe Diot. Characterization of network-wide anomalies in traffic flows. In *IMCONF 2004 Proceedings*, pages 201–206, 2004.

[36] Anukool Lakhina, Mark Crovella, and Christophe Diot. Mining anomalies using traffic feature distributions. In *SIGCOMM '05: Proceedings of the 2005 conference*, pages 217–228, 2005.

[37] John Zhong Lei and Ali Ghorbani. Network intrusion detection using an improved competitive learning neural network. In *Second Annual Conference on Communication Networks and Services Research (CNSR'04)*, pages 190–197, Faculty of Computer Science University of New Brunswick Fredericton, NB, E3B 5A3, Canada, 2004.

[38] Kirill Levchenko, Ramamohan Paturi, and George Varghese. On the difficulty of scalably detecting network attacks. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 12–20, 2004.

[39] Jianning Mai, ChenNee Chuah, Ashwin Sridharan, Tao Ye, and Hui Zang. Is sampled data sufficient for anomaly detection? In *IMCONF 2006 Proceedings*, 2006.

[40] Steven McCanne and Van Jacobson. The BSD packet filter: A new architecture for user-level packet capture. In *1993 USENIX Winter Conference Proceedings*. USENIX, January 1993.

[41] MIT Lincoln Laboratory. 1999 DARPA MIT intrusion detection evaluation data sets, 1999.

[42] Biswanath Mukherjee, L. Todd Heberlein, and Karl N. Levitt. Network intrusion detection. *Network, IEEE*, 8:26–41, May-June 1994.

[43] ntop.org. http://www.ntop.org/pf_ring.html, 2008.

[44] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Comput. Networks*, 31(23-24):2435–2463, 1999.

[45] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. A multifaceted approach to understanding the botnet phenomenon. In *IMCONF 2006 Proceedings*, 2006.

[46] Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of LISA '99: 13th Systems Administration Conference*, November 1999.

[47] Joseph S. Sherif and Tommy G. Dearmond. Intrusion detection: Systems and models. In *Proceedings of the Eleventh IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE02)*, California Institute of Technology, JPL, Pasadena, CA 91109 and California State University, Fullerton, CA 92834, 2002.

[48] Stephen. E. Smaha. Haystack: an intrusion detection system. In *Aerospace Computer Security Applications Conference, 1988., Fourth*, pages 37–44, December 1988.

[49] tcpdump.org. http://www.tcpdump.org, 2008.

[50] The Snort Project. http://www.snort.org/docs/snort_manual/2.8.0/snort_manual.pdf, 2008.

[51] USENIX ID99. http://www.usenix.org/publications/library/proceedings/detection99/, 1999.

[52] A. Wagner and B. Plattner. Entropy based worm and anomaly detection in fast IP networks. In *14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise*, pages 172–177, June 2005.

[53] Ping Wang, Sherri Sparks, and Cliff C. Zou. An advanced hybrid peer-to-peer botnet. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, 2007.

[54] wireshark.org. http://www.wireshark.org/docs/, 2008.

[55] Haiquan Zhao, Ashwin Lall, Mitsunori Ogihara, Oliver Spatscheck, Jia Wang, and Jun Xu. A data streaming algorithm for estimating entropies of od flows. In *IMCONF 2007 Proceedings*, 2007.

[56] Tanja Zseby. Deployment of sampling mehtods for SLA validation with non-intrusive measurements. In *PAM 2002 Proceedings*, MArch 2002.

[57] Tanja Zseby. Comparison of sampling methods for non-intrusive SLA validation. In *E2EMON 2004 Proceedings*, October 2004.