Copyright Statement

The digital copy of this thesis is protected by the Copyright Act 1994 (New Zealand).

This thesis may be consulted by you, provided you comply with the provisions of the Act and the following conditions of use:

- Any use you make of these documents or images must be for research or private study purposes only, and you may not make them available to any other person.
- Authors control the copyright of their thesis. You will recognise the author's right to be identified as the author of this thesis, and due acknowledgement will be made to the author where appropriate.
- You will obtain the author's permission before publishing any material from their thesis.

To request permissions please use the Feedback form on our webpage. [http://researchspace.auckland.ac.nz/feedback](http://researchspace.auckland.ac.nz/feedback)

General copyright and disclaimer

In addition to the above conditions, authors give their consent for the digital copy of their work to be used subject to the conditions specified on the Library Thesis Consent Form.
Network Event Detection with Entropy Measures

Raimund E. A. Eimann

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Computer Science

The University of Auckland, 2008
Abstract

Information measures may be used to estimate the amount of information emitted by discrete information sources. Network streams are an example for such discrete information sources. This thesis investigates the use of information measures for the detection of events in network streams.

Starting with the fundamental entropy and complexity measures proposed by Shannon and Kolmogorov, it reviews a range of candidate information measures for network event detection, including algorithms from the Lempel-Ziv family and a relative newcomer, the T-entropy. Using network trace data from the University of Auckland, the thesis demonstrates experimentally that these measures are in principle suitable for the detection of a wide range of network events.

Several key parameters influence the detectability of network events with information measures. These include the amount of data considered in each traffic sample and the choice of observables. Among others, a study of the entropy behaviour of individual observables in event and non-event scenarios investigates the optimisation of these parameters.

The thesis also examines the impact of some of the detected events on different information measures. This motivates a discussion on the sensitivity of various measures.

A set of experiments demonstrating multi-dimensional network event classification with multiple observables and multiple information measures concludes the thesis.
Preface and Acknowledgments

This Ph.D. thesis applies T-entropy, a relatively new information measure, for network event detection for the first time. If the author is correct, it is also the first Ph.D. thesis to review the expected entropies of common network protocol fields.

Now looking back this project, I remember many occasions on which I could have gotten sidetracked. I would like to express my gratitude to colleagues and friends for their support during the sometimes quite challenging periods of this project.

My academic supervisor Ulrich Speidel made himself available at short notice for discussions and was always ready with good suggestions when I ran into difficulties. On countless occasions Ulrich helped me to ship around difficulties by providing new insights and ideas. Ulrich’s support was not limited to the project work itself: He also helped me come to terms with turbulent times in my personal life, the loss of my father being the most difficult.

Nevil Brownlee has been my co-supervisor during this project. At the beginning of the project I had the opportunity of visiting Nevil for three days at the Cooperative Association for Internet Data Analysis (CAIDA) in San Diego, USA. It was a privilege to meet many interesting people that work in the area of network measurement and data analysis face-to-face.

Both Ulrich and Nevil have provided essential help with the research papers that were published in the course of this project.

On the technical side, I owe thanks to the staff at ITSS and the Department of Computer Science. In particular, Russell Fulton and James Harper have greatly helped with data interpretation and acquisition.

I very much appreciated the feedback and suggestions from my fellow Ph.D. students Jia Yang and DongJin Lee.

With English being my second language, this thesis depended heavily on the watchful eyes of my proof-readers, Caroline and Peter Seddon, as well as Michael Taylor to whom I feel deeply indebted.

This project would not have been possible without a three year stipend from the Department of Computer Science at the University of Auckland provided. I would also like to thank Microsoft New Zealand for their interest in this project and their generous scholarship in support of it.

Last but not least I want to thank my family for their continuous love and support during this project. I am particularly grateful for my mother’s support with the household and her loving care for my daughters Melanie and Bianca while I was busy writing.

Auckland, April 2008

Raimund Eimann
# Contents

1 Introduction ................................................. 13
   1.1 Structure of this thesis ................................. 15
   1.2 Other work in the area/literature ...................... 17

2 What’s in a string? - Information measures ................. 21
   2.1 General Remarks ......................................... 21
   2.2 Complexity measures ..................................... 23
      2.2.1 Kolmogorov-Chaitin complexity ..................... 23
      2.2.2 LZ production complexity ............................. 24
      2.2.3 LZ77 .................................................. 26
      2.2.4 LZ78 .................................................. 28
      2.2.5 T-complexity .......................................... 29
   2.3 Entropy measures .......................................... 34
      2.3.1 Entropy in Thermodynamics and Information Theory . . 34
      2.3.2 Shannon entropy ...................................... 34
      2.3.3 N-gram entropy ...................................... 35
2.3.4 Entropy estimators ........................................... 36

2.4 Commonalities and differences between the two approaches ............... 36

2.5 T-information and T-entropy ..................................... 38

2.6 Concluding remarks ............................................... 40

3 Network Events .................................................. 41

3.1 Possible information measurement points within a network ............... 41

3.2 Events and non-events ........................................... 42

3.3 Networks considered in this thesis ................................ 43

3.3.1 Trace files collected at the UoA border gateway .................. 44

3.3.2 Additional trace files collected at an internal campus gateway .......... 45

3.3.3 Third-party trace files: NLANR PMA Special Traces Archive .......... 46

3.4 Events considered in this thesis .................................. 46

3.4.1 Distributed Denial of Service (DDoS) attacks .................... 48

3.4.2 Spam .......................................................... 49

3.4.3 Windows Messenger spam ...................................... 50

3.4.4 Scans .......................................................... 51

3.4.5 Worms ......................................................... 52

3.4.6 SSH dictionary attack .......................................... 52

3.4.7 SYN flood attacks ............................................. 53

3.4.8 ICMP floods ................................................... 54

3.4.9 Failure of network equipment .................................... 54

3.5 Conclusions ....................................................... 55
## 4 Entropy of Network Traffic

4.1 Capturing network traffic ................................................. 58
4.2 T-entropy of raw IPv4 traffic .......................................... 59
4.3 T-entropy of filtered IPv4 traffic ..................................... 63
4.4 T-entropy of IPv4 traffic with added artificial events .......... 65
   4.4.1 Generating artificial events for injection into a trace file .... 65
   4.4.2 Example: injection of an artificial SYN flood .......... 68
4.5 Conclusions ................................................................. 69

## 5 Noise

5.1 Types of noise ............................................................... 71
5.2 Signal-to-noise ratio ...................................................... 73

## 6 Packet Mapping

6.1 Motivation ................................................................. 77
6.2 Field selection ............................................................ 80
6.3 Simple mapping .......................................................... 82
6.4 Fields available for mapping ......................................... 84
   6.4.1 PCAP packet header fields ................................. 85
   6.4.2 Ethernet header fields ...................................... 87
   6.4.3 IPv4 header fields .......................................... 88
   6.4.4 TCP header fields .......................................... 97
   6.4.5 UDP header fields .......................................... 100
Today, most organisations use IP networks connected to the Internet. Operators of such networks are generally interested in recognising network faults and attacks quickly. One rather narrowband approach in this area is to monitor the behaviour of individual hosts as discussed in Tan and Maxion [98], for example. Another more broadband approach is to monitor networks as a whole for anomalies by analysing the network traffic. This thesis follows the latter strategy. The recognition of patterns associated with faults and attacks is a challenging task, due to the highly discrete nature of network traffic. At least three different approaches can be distinguished:

1. **Signature-based analysis:** Traditionally, known attacks can be recognised easily with intrusion detection systems such as Snort [6]. This type of detection mechanism searches network traffic for signatures associated with attacks. One severe disadvantage of such intrusion detection systems is that new attacks cannot be recognised until their respective signature has been added to the signature database. Another problem is fault detection, as network equipment faults are not detectable
with signature-based approaches.

2. **Statistical analysis:** Another common approach to the detection of anomalies is the use of statistical tools, such as Bayesian methods (see [24] or [96], for example). Detectors based on such methods usually work in a proactive manner, because they do not depend on fixed signatures. Instead, the detection is based on data such as packet counts, packet sizes, and transmission errors. Statistical analysis of this data permits profile generation to describe normal network behavior. Advanced statistical detectors can adapt the profile to slow changes over time. A sudden and lasting deviation from such a profile may indicate an event. This includes network equipment faults, which is a strength of the statistical approach. One problem with statistical approaches is the narrow choice of observables which contribute to individual statistics. Events which do not affect a particular choice of observables may thus not be noticed.

3. **Information-theoretical analysis:** This is a relatively new approach that has gained popularity in recent years. Its detection strategy is to estimate the amount of information carried in a network stream: this approach takes samples from the stream (typically either in the form of flow records for TCP as described in Wagner et al. [109], [110], for instance, or as simple byte sequences for arbitrary protocol levels as described in Kulkarni et al. [60]) and estimates the information content of these samples by:

   - computing the number of steps required to build the sample string with elementary steps (complexity measures, see [67], for instance), or
   - computing the rate at which new patterns appear in the sample string (entropy measures, see [88], for instance).

The complexity measurement technique assumes that samples contain a roughly constant amount of information during normal operation. The entropy measurement technique assumes that the rate with which new patterns appear in the samples is roughly constant. The information-theoretical approach to network event detection interprets sudden changes in the complexity or the appearance rate of new patterns as indicators for network events. Both network attacks and network equipment faults may cause such changes. The range of detectable events is thus in principle at least as large as that of the statistical approach. However, the range of detectable events is likely to be even larger, because the choice of observables may cover entire packet signatures, permitting unanticipated correlations between observables to influence the result.

As the title suggests, this thesis attempts a contribution to the information-theoretical approach. It presents a number of event detection experiments with a relatively new measure, T-entropy. This measure
was proposed by Titchener in 1998 and is used here for the first time in the context of network event detection.

The thesis examines detection both with complete IPv4 packet signatures (limited by the capture length per packet only) as well as with narrower sets of observables. The latter required a study of the entropy behaviour of individual observables associated with common protocols such as IPv4 and TCP. One conclusion from this study is that individual observables may counteract each other in their behaviour under certain events, so that the footprint of one partially cancels the footprint of the other (cf. Section 6.1, Point 2).

In order to quantify the impact of network events on the information rate of a network stream, this thesis defines a signal-to-noise ratio measure. Among other applications, this measure is used to determine a suitable sliding window size for the detection of specific events.

The size of the sliding window has an influence on entropy/complexity estimation errors. The relative impact of window size reduction on the estimation error is also studied for different information measures.

The experiments in this thesis cannot cover all possible combinations of observables and sliding window sizes. Experimental setups thus often rely on heuristic guesses and not all experiments carried out have been included in this thesis.

The thesis concludes with a set of multi-dimensional experiments, using different measures and observables/sets of observables for the different dimensions.

The main results of this thesis were published at APNOMS’05 [33], IEEE ICICS’07 [93], in a special issue of the Mediterranean Journal of Computers and Networks [94], and as a presentation at NZNOG’08 [34].

A detailed roadmap of this thesis is provided in the following section.

## 1.1 Structure of this thesis

The following bullet list intends to give the reader an overview of the thesis. As readers probably come from different backgrounds, some chapters in the thesis may be more appealing to one reader and less appealing to another reader. This section may be helpful to decide which chapters are relevant to individual readers.

- The remainder of Chapter 1 reviews literature in the area.
Chapter 2 forms the theoretical foundation of this thesis. The Kolmogorov-Chaitin complexity is presented as an ideal information measure, which unfortunately cannot be used for practical purposes, as it is not computable. Other information measures discussed in this chapter include the Lempel-Ziv family of measures, Shannon entropy, and the measures associated with T-information theory.

Network events are the topic of Chapter 3. The chapter starts out considering which observation points in a network topology are suitable for event detection. A discussion of what one might want to consider a network event follows. As the network environment is an important factor in this discussion, a brief description of the specific network environments considered in this thesis is presented subsequently. The chapter finishes with a description of several network events that may be detectable with information measures.

Chapter 4 is a first stab at network stream data analysis with T-entropy. A simple first experiment exhibits several steep entropy drops, which also appear with established measures. The traffic samples related to these drops contain unusual traffic patterns. The chapter concludes with two further experiments, which establish a causal relationship between these traffic patterns and the entropy drops.

One observation from the experiments in Chapter 4 is that the entropy samples contain residual noise. Chapter 5 examines possible causes of this noise. The noise level determines the minimum entropy deviation required for events to be detectable. To quantify that, the second section of the chapter introduces a signal-to-noise ratio that is frequently applied in subsequent chapters.

Chapter 6 examines the influence of individual fields in IPv4 traffic on the entropy of network traffic. The concept of packet mapping is introduced to improve the specificity of the entropy measures to network traffic by considering only certain fields of packet records (or combinations thereof) for entropy analysis. A large section is dedicated to the information-theoretic properties of those fields commonly found in network traffic today.

Network traffic may be sampled in a variety of ways. Chapter 7 presents three sliding window methods for traffic sampling and discusses their individual advantages and disadvantages. The remainder of the chapter discuss the influence of the window size on entropy-based network event detection.

Chapter 8 acts as the experimental proof of the hypotheses about entropy-based network event detection proposed in Chapter 3. A variety of events found in the experimental data are discussed in detail. Readers primarily interested in the theoretical side of entropy-based network event detection might want to skip this chapter.
Chapter 9 investigates the sensitivity of various information measures to network events. In particular, the sensitivity of T-entropy to events with different magnitudes is examined. The second half of this chapter compares the sensitivity of T-entropy to the LZ production complexity.

Chapter 10 studies how different information measurements of the same traffic samples may be combined for multi-dimensional analysis. The first part of the chapter looks at multi-dimensional analysis with respect to different mappings. Analyses with multiple information measures are the topic of the second part.

Chapter 11 presents conclusions that can be drawn from this thesis and proposes areas that may be interesting for further studies.

The Appendix, starting from page 159, contains details on the software developed for this project and an overview of the mappings used in the thesis.

1.2 Other work in the area/literature

Network event detection with information measures is not new. In recent years, a considerable amount of research has been done in this area. This section reviews a number of relevant publications.

The initial motivation for this Ph.D. project were two remarks in an article published by Kulkarni, Bush, and Evans [60] which reviews the detection of DDoS attacks with Kolmogorov complexity measures: “The complexity estimation technique used here is not the best because empirical entropy is actually a very poor method of complexity estimation. For example, the estimate for the string $1010101010101010101$ and a completely random string with equal numbers of 1s and 0s is the same under empirical entropy.” and “Obviously, good estimation of Kolmogorov Complexity is key to its usefulness in identifying correlation between attack flows. Although our simple entropy calculation technique served as a useful metric and it is computationally efficient, we are investigating and benchmarking more estimators for $K(x)$.”

Mark Titchener proposed T-entropy in 1998 as a new entropy measure [102]. T-entropy can be computed efficiently and it exhibits a close relationship to known entropies such as the Kolmogorov-Sinai entropy of the logistic map (in Ebeling, Steuer, and Titchener [32]). These properties and the comments in Kulkarni, Bush, and Evans [60] prompted experimentation with this measure on network streams.

In an earlier publication, Kulkarni and Bush [61] describe how complexity measurement could be used for network event detection, with some detail left out, due to page restrictions for their publication. This

---

1Many authors use the term *empirical entropy* for 1-gram Shannon entropy (cf. Section 2.3.2 and 2.3.3) if the associated probabilities are derived from measured symbol frequencies.
paper also describes sample clustering in multi-dimensional observation spaces as a result of network events.

Xu, Zhang, and Bhattacharyya [117] published a seminal paper on profiling Internet backbone traffic. In their paper they use Shannon entropy in conjunction with typical flow observables, such as source/destination IP address and source/destination ports for the identification of “interesting host behaviours”. The experiments are based on 5-minute sampling intervals, which is relatively long compared to the experiments carried out in this thesis. Similar to the experiments in Chapter 10 of this thesis, Xu, Zhang, and Bhattacharyya associate sample clusters in a multi-dimensional observation space with events.

While this thesis measures the global entropy/complexity at a single monitoring point, Zhao, Lall, Ogihara, Spatscheck, Wang, and Xu [120] propose the use of Shannon entropy to determine the entropy of the traffic between every origin-destination pair, where origin and destination are defined as ingress and egress points, respectively.

Chakrabarti, Ba, and Muthukrishnan [27] propose the use of empirical entropy for entropy estimation of general data streams, thus covering a wider spectrum of data streams than Kulkarni et al. [60], who use the same measure. Their approach is characterised by tight time and space constraints. Chakrabarti, Ba, and Muthukrishnan also published [28] as a follow-up paper of [27].

Lall, Sekar, Ogihara, Xu, and Zhang [64] propose two algorithms for the fast estimation of the empirical entropy of certain fields such as the IPv4 source/destination address on high speed links.

Feinstein, Schnackenberg, Balupari, and Kindred [36] propose the use of empirical entropy in combination with chi-square tests for the detection of DDoS attacks. Like many other authors, their focus with respect to observables appears to be on the address fields in IPv4 and the port fields in transport layer protocols.

Burgess, Haugerud, Straumsnes, and Reitan [23] use Jaynes’ maximum entropy principle [49, 50] for the calculation of statistical system behaviour in order to describe the normal behaviour of hosts. Among others, their work includes studies on normal SSH connection behaviour and sliding window size considerations, as well as changes in the entropy during attacks – all being topics which will also be discussed in the thesis at hand.

John and Tafvelin [52] examine properties of fields in common protocols such as IPv4 and TCP, along with value distributions for some of these fields, which are largely consistent with the findings presented in this thesis. The trace file data collected for their work originates from an OC-192 backbone link.
Another publication [53] from the same authors examines updated statistical data, such as transport protocol and packet size distributions from the same OC-192 link. A third publication [54] proposes heuristics for the classification for backbone flows. A publication by John, Tafvelin, and Olovsson [55] examines the connection establishment and termination behaviour of different classes of backbone traffic, such as P2P, HTTP and malicious traffic. While information measurement is not a topic in these four publications, they largely confirm statistical observations made here. The research papers mentioned are also published as part of John’s Licentiate of Engineering thesis [51].

Ziviani, Gomes, Monsores and Rodrigues [121] propose the use of Tsallis entropy for the detection of anomalies in autonomous systems. According to this publication, Tsallis entropy permits higher numbers of detected traffic patterns than previous entropy-based detection approaches. For the evaluation of this approach, the authors use reference data from Moore, Shannon, Brown, Voelker, and Savage [78].

Zhang, Roughan, Lund, and Donoho [119] propose the use of Shannon entropy for traffic matrix estimation. The information gained from such matrices can be used to predict component failure impact, for example. The detection of component failures is also studied in this thesis.

Gianvecchio and Wang [38] propose the use of entropy measures for the detection of covert timing channels. Covert timing channels may be used to obscure illegitimate data transfers from intrusion detection systems. Like other publications mentioned above, the fundamental entropy measure this publication is empirical entropy.

Gu, McCallum and Towsley [40] propose a combination of profile- and information-based schemes for network event detection: a maximum entropy baseline distribution derived from training data is compared with a distribution derived from network monitoring, using relative entropy. Differences between the two distributions are interpreted as indicators for network events.

Liu, Towsley, and Ye [74] estimate the overall information carried in full header traces with data compression. The high compression ratios described in their publication are consistent with the low entropies observed in this thesis, arising from the highly repetitive nature of packet headers. They further examine the amount of information associated with three monitoring paradigms: full packet header capturing, flow-level capturing and SNMP statistics. The fundamental information measure of this publication is once again Shannon entropy.

Lakhina, Crovella, and Diot [63] examine observables that are typically affected by common network events, such as IPv4 addresses and transport protocol ports. With a focus on these observables, their paper compares classical volume-based detection methods with entropy-based (here: Shannon entropy)
approaches. Their findings may be seen as further justification for the work carried out in this thesis project: “In summary, the results in this section are encouraging for the use of entropy as a metric for anomaly detection. We find that entropy-based detection exposes a large number of anomalies that cannot be detected using volume-based methods”. The authors further examine entropy clustering concerning addresses and ports as observables in multi-dimensional observation spaces, similar to the experiments carried out in Section 10.2 of this thesis.

A comparably early publication by Lee and Xiang [66] examines the applicability of various information measures, such as conditional entropy and relative conditional entropy for the purpose of network event detection.

Wagner and Plattner [110] use common compression algorithms such as \texttt{gzip}, \texttt{bzip2} and LZO on flow data to approximate the entropy of the associated data. A later publication by Wagner [109] considers only LZO for its relatively high speed and low memory requirements. Like Xu, Zhang, and Bhattacharyya [117], Wagner uses 5 minute sampling intervals, which again is relatively long compared to the approach presented in this thesis. Wagner’s and Plattner’s entropy estimation strategy via compression ratios is similar to Wehner’s approach [112]. However, Wehner appears to consider both entire packet signatures as well as flow data.

Evans and Barnett [35] use Unix \texttt{compress}, a derivative of LZ78 (cf. Section 2.2.4), as an estimator for Kolmogorov-Chaitin complexity in network event detection.

Cilibrasi and Vitányi [29] also define a similarity metric based on compression. Unlike the other compression-based approaches just cited, Cilibrasi and Vitányi’s publication is not intended to be subject-specific.
2

What’s in a string? - Information measures

2.1 General Remarks

This thesis examines the usability of information measures on network stream information for the purpose of detecting network events. Nowadays, most networks use digital technology, so the information transported across them is encoded in strings of symbols. Therefore, measurement of information in strings is a central topic in this thesis. The question of measuring information in strings of a general nature has already attracted much research interest in the past, which largely rests on the fundamental works of Kolmogorov [58], Chaitin [25, 26], Shannon [88, 89], and Levenshtein [70]. Some of the concepts proposed are revisited in this chapter.

One problem with the measurement of information in strings is the term information itself, as the exact
meaning of this term is difficult to define. Two widely used approaches to this problem exist:

- **Complexity approach:** Here, the information carried in a string $x$ is regarded as the minimum number of elementary steps required to construct $x$. The central concept in this area is the Kolmogorov-Chaitin complexity\footnote{In the literature often only referred to as *Kolmogorov complexity*.}, which forms the theoretical foundation of this approach. A large part of Section 2.2.1 deals with the circumstance that Kolmogorov-Chaitin complexity is not computable, however. For this reason, alternative computable complexity measures have been developed. An example is the LZ production complexity, which is closely related to the compression algorithms of the Lempel-Ziv family.

- **Entropy approach:** Entropy has two origins, one of which lies in thermodynamics in physics where it is used as a measure for order/chaos and for the predictability of the state of a physical system. Entropy was also independently discovered by Shannon in the area of information theory where it is used as a measure for the predictability of an information source. Von Neumann alerted Shannon [84] to the fact that both concepts are mathematically identical, so Shannon also named his measure *entropy*. Absolute entropy is generally not studied in either of the two realms, physics or information theory. Rather, the objects of study are changes in entropy. For this reason, the terminology used in the literature can occasionally be confusing as entropy rates are often simply referred to as entropy. An information source is said to have maximum entropy in the Shannon sense if it generates all symbols and symbol patterns with identical probability.

The structure of this chapter reflects these two predominant approaches to information measurement. Section 2.2 revisits various complexity measures such as Kolmogorov-Chaitin complexity, LZ production complexity (as well as two measures derived from it), and T-complexity. The subsequent section discusses the origin of entropy as an information measure and continues with brief descriptions of contributions to this area by Shannon. At the end of the chapter T-entropy is introduced, an information measure which combines elements from both realms, complexity as well as entropy.
2.2 Complexity measures

2.2.1 Kolmogorov-Chaitin complexity

At its core, Kolmogorov-Chaitin complexity deals with descriptions of strings in some fixed description language $L$. Properties which $L$ has to satisfy will be discussed shortly. In essence, the Kolmogorov-Chaitin complexity $C(x)$ of a string $x$ is the length of the shortest description $d(x) = y \in L$ for $x$ with $d^{-1}(y) = x$:

$$C(x) = |y| : \forall y' \in L : d^{-1}(y') = x \land |y| \leq |y'|$$

(2.1)

According to Li and Vitányi [71], a description language $L$ is suitable if the mapping from $x$ to $d(x)$ can at least in principle be executed efficiently for any given $x$, i.e., it can be carried out by humans or machines. This concept is formally described by partial recursive functions.

An earlier result, the Church-Turing thesis [46], says that the class of partial recursive functions coincides with the class of functions which can be computed by a Turing machine. A universal Turing machine can simulate any other (fixed purpose) Turing machine [46] and can therefore compute any partial recursive function. The Kolmogorov-Chaitin complexity of a string $x$ is thus independent of the chosen description language $L$: apart from an additive constant contributed by the finite control of the universal Turing machine, it only depends on the inherent complexity of $x$ itself. Without any loss of generality, one
may therefore use any Turing-complete [21] language as string description language \( L \). All popular programming languages are Turing-complete, for example.

If the Kolmogorov-Chaitin complexity of a string \( x \) is less than \( |x| \), we say that \( x \) is *compressible*. In the opposite case, we say that \( x \) is *incompressible*. For incompressible strings \( x \), the shortest description for \( x \) is \( x \) itself. Each set of all strings of length \( n \) over an alphabet of size \( \#A \) contains at least one element that is not compressible. This may be seen by simple counting: there are \((\#A)^n\) strings of length \( n \), but there are only \( \sum_{i=0}^{n-1} (\#A)^i = \frac{1}{n-1}(\#A)^n - 1 \) shorter strings to which these may be compressed.

For practical purposes like the one proposed in this thesis, a computable complexity measure is required. Unfortunately, Kolmogorov-Chaitin complexity is not computable, see Kolmogorov [58] and Li and Vitányi [71]. The proof is briefly outlined below in a form that is commonly used in the literature, for example in the corresponding Wikipedia article [115].

1. Assume that Kolmogorov-Chaitin complexity is computable by some function \( C(x) \) of length \( m \).

2. It is therefore possible to compute the most complex string \( s \) for any given string length \( l \) (with an algorithm of fixed length \( n \)). Because of the counting argument mentioned earlier, it follows that \( C(s) \geq |s| \).

3. We now choose \( l > m + n \). Our algorithm computes the most complex string among all strings of size \( l \) and we know that the shortest possible description for it is \( l \). However, we just described that string with an algorithm which is \( m + n < l \) characters long, which is a contradiction.

Because Kolmogorov-Chaitin complexity is not computable, we cannot know the shortest possible description for a given string. However, there is a number of computable complexity measures and compressors that permit *estimations* of the Kolmogorov-Chaitin complexity. Such computable complexity measures tend to overestimate the complexity of strings. In the following sections, computable complexity measures from the family of Lempel-Ziv algorithms and the newer, less well-known T-complexity measure will be examined.

### 2.2.2 LZ production complexity

In 1976, Abraham Lempel and Jacob Ziv proposed a computable string complexity measure [67], [118] which they called *production complexity*. In this thesis, it will be referred to as Lempel-Ziv production complexity, LZ production complexity, or simply LZ76. Algorithm 2.2.1 outlines its computation. Given
an alphabet $A$ and a string $x \in A^n$ of length $n$, this measure parses $x$ – broadly speaking – successively into a list $S$ of substrings of $x$, such that each substring consists of a previously parsed part of the string followed by an “innovation” symbol. Each list item in $S$ is called a production step in the production (of $x$). The number of production steps (i.e., $\#S$) is the LZ production complexity. Algorithm 2.2.1 makes no provision for storing elements of $S$, however.

Algorithm 2.2.1 LZ production complexity

Require: String $x$ of length $n > 0$

1. Initialise a position counter $p = 1$ and a production step counter $c = 0$.
2. while $p \leq n$ do
3.     $i = 0$
4.      while $x(p, p + i) = x(j, j + i)$ for some $1 \leq j < p$ do
5.          $i = i + 1$
6.      if $p + i > n$ then break
7.          end while
8.     $c = c + 1$
9.     $p = p + i + 1$
10. end while
11. return $c$

In accordance with the notation introduced above, $x(p, p + i)$ denotes the substring of $x$ from position $p$ to position $(p + i)$ inclusive. This is only considered a production step if one of the following conditions is met:

1. No copy of $x(p, p + i)$ starts in $x(1, p - 1)$, or
2. $x(p, n)$, i.e., the end of $x$ is reached.

In the literature about LZ production complexity, production steps satisfying the first condition are usually referred to as exhaustive production steps. Lempel and Ziv showed that every non-empty finite string can be parsed into $c$ production steps of which the first $c - 1$ are exhaustive.

Example 2.2.1 (LZ production complexity)

Given a string $x = abacabcacabcabc$, algorithm 2.2.1 parses $x$ into $S = \{a, b, c, ac, bcab, cababcabc\}$.

The LZ production complexity of $x$ is therefore $\#S = 6$.

The computational time complexity of Algorithm 2.2.1 is $O(n^2)$, because the substring search for each production step starts from the first symbol of $x$. This constitutes a limitation on the practical use of LZ76. Lempel and Ziv subsequently developed the LZ77 algorithm, which addresses this problem.
A number of publications, such as Rodeh et al. [87] and Gusfield [45], claim that the computational complexity of LZ production complexity could be reduced to $O(n)$. This claim rests on works by Weiner [113], McCreight [76], and Ukkonen [108], which propose the use of suffix trees as data structure. Although some of these publications, Rodeh et al. [87] for example, were more than two decades old at the time of writing, no usable implementation of an efficient algorithm for LZ production complexity was available for experimentation. The author of this thesis had some difficulties understanding the description of the efficient LZ production complexity algorithm given in [87] and was therefore unable to implement it himself. Descriptions of the algorithm seem to indicate that pointers are used in the suffix tree. This, in turn, seems to imply that the actual computational complexity would have to be at least $O(n \log n)$. This would be identical to the ftd algorithm for T-entropy computation discussed later in Section 2.2.5 on page 31.

In principle, LZ production complexity could be used as a front-end for a compression algorithm. Such an algorithm would work similarly to LZ77, which will be discussed in the following section, albeit with an unlimited window size. This algorithm would recognise all the patterns LZ77 recognises, plus additional patterns which require larger windows. Hence, such a compression algorithm should yield higher compression rates than LZ77. This poses the question why LZ77 still forms the basis for many popular compression algorithms if a better algorithm would actually have been available for more than two decades.

### 2.2.3 LZ77

LZ77 (cf. Lempel and Ziv [68] or Bell, Cleary, and Witten [17]) is a variant of the LZ production complexity algorithm and is almost exclusively known for its use as a data compression algorithm. However, it can also be used to estimate the LZ production complexity. It limits the substring search in step 5 of Algorithm 2.2.1 to a sliding window of fixed size $m$. This way, the computational time complexity of LZ77 is $O(mn)$ which permits the practical use of this algorithm on large $x$. Data compression is possible, because LZ77 outputs codewords which can be uniquely recombined into $x$. These codewords consist of three components which will be discussed in more detail shortly.

The sliding window used in LZ77 is split into two sub windows, a *dictionary window* $W_d$ of size $m_d$ and a *lookahead window* $W_l$ of size $m_l$. Thus $m = m_d + m_l$. Initially, $W_d$ is empty and $W_l$ contains the first $m_l$ characters of the input. Each iteration of the algorithm consists of two phases, a matching phase and a window shift.
During its matching phase, the algorithm searches for the longest matching sequence at the beginning of \( W_l \) which starts in \( W_d \). Note this match may straddle the boundary of \( W_d \) with \( W_l \) and read into \( W_l \) (this happens in the last two lines of Example 2.2.2 below).

Once the match is complete, an output codeword \((o,\ell,\kappa)\) is generated. Its three components are the offset \(o\) of the match in \( W_d \), the length \(\ell\) of the match, and the first character \(\kappa\) in (or after) \( W_l \) that does not belong to the match. \(\kappa\) is usually referred to as \textit{innovation}. Dots (.) denote unoccupied offsets in \( W_d \) and \( W_l \). \(\kappa = -\) indicates that there is no match in \( W_d \), which consequently is associated with match length 0.

After the output codeword is generated, the sliding window is shifted forward by the match length plus one symbol and the next iteration of the algorithm is performed. These iterative steps repeat until the input is completely parsed.

**Example 2.2.2 (LZ77)**

This example shows the steps LZ77 takes while parsing the input \( x = abcabcabcabcabcabc \). A sliding window size of 15 characters was chosen with \( m_d = 10 \) and \( m_l = 5 \).

<table>
<thead>
<tr>
<th>Step</th>
<th>( W_d )</th>
<th>( W_l )</th>
<th>Output ((o,\ell,\kappa))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0123456789</td>
<td>12345</td>
<td>((-,0,a))</td>
</tr>
<tr>
<td>1</td>
<td>...........</td>
<td>abcac</td>
<td>((-,0,a))</td>
</tr>
<tr>
<td>2</td>
<td>...........a</td>
<td>bcacb</td>
<td>((-,0,b))</td>
</tr>
<tr>
<td>3</td>
<td>...........ab</td>
<td>cacbc</td>
<td>((-,0,c))</td>
</tr>
<tr>
<td>4</td>
<td>...........abc</td>
<td>acbca</td>
<td>((7,1,c))</td>
</tr>
<tr>
<td>5</td>
<td>...........abc</td>
<td>bcabc</td>
<td>((6,3,b))</td>
</tr>
<tr>
<td>6</td>
<td>.abcabcabc</td>
<td>cabca</td>
<td>((7,5,b))</td>
</tr>
<tr>
<td>7</td>
<td>abcabcabc</td>
<td>cabc.</td>
<td>((7,4,\lambda))</td>
</tr>
</tbody>
</table>

If one considers the 7 bit ASCII alphabet for this example, the input consists of 133 \textit{bits} and the output of 105 \textit{bits}, using 4 \textit{bits} each for \( o \) and \( \ell \). Note that real implementations of LZ77 would choose larger values for \( m_d \) and \( m_l \) for higher compression ratios. The innovation for the last iteration of the algorithm is the empty string \( \lambda \).

LZ77 parsings are likely to end in known patterns. This is the case for the parsing in Example 2.2.2. If an LZ77 parsing ends in a known pattern, the innovation for the last iteration of the algorithm is \( \lambda \).

LZ77 “forgets” previously seen patterns if they do not reappear at least once every \( m_d \) symbols. This may have a negative effect on the compression ratio. As an alternative, Lempel and Ziv developed another compression algorithm which builds and maintains a dictionary of all past production steps in the course of parsing. This algorithm is called LZ78 and is discussed in detail in the following section.
2.2.4 LZ78

LZ78 [69], [17] is a compression algorithm which matches known prefixes \( p_i \) from a dictionary \( D = \{ \lambda, p_1, \ldots, p_k \} \) against an input string \( x \in A^* \). \( \lambda \) is the empty string, the only initial element in \( D \).

The output of LZ78 is a sequence of 2-tuple codewords: the first component is a pointer to an earlier dictionary entry and the second component is a symbol of \( A \). Hence, each output codeword is a reference to a known prefix and, as in LZ77, an innovation added to that prefix. The reference to \( \lambda \) in \( D \) is 0.

The operation of LZ78 is outlined as follows: let \( 1 \leq j \leq n \) be the current parsing position. Starting with zero, LZ78 iterates an offset counter \( i \) as long as \( x(j, j+i) \) exists in \( D \). When the first pattern \( x(j, j+i) \) is encountered which does not exist in \( D \), this pattern is added to \( D \) and the parsing position is updated to \( j + i + 1 \). This is repeated until \( x \) is fully parsed. Similar to LZ77, an LZ78 parsing of \( x \) is likely to end in a known pattern. In this case the empty string \( \lambda \) is used as innovation.

**Example 2.2.3 (LZ78)**

The input \( x = abcabcabcabcabcabc \) is parsed with LZ78.

<table>
<thead>
<tr>
<th>Index</th>
<th>( D ) entry</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>(0,a)</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>(0,b)</td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td>(0,c)</td>
</tr>
<tr>
<td>4</td>
<td>ac</td>
<td>(1,c)</td>
</tr>
<tr>
<td>5</td>
<td>bc</td>
<td>(2,c)</td>
</tr>
<tr>
<td>6</td>
<td>ab</td>
<td>(1,b)</td>
</tr>
<tr>
<td>7</td>
<td>ca</td>
<td>(3,a)</td>
</tr>
<tr>
<td>8</td>
<td>bca</td>
<td>(5,a)</td>
</tr>
<tr>
<td>9</td>
<td>bcab</td>
<td>(8,b)</td>
</tr>
<tr>
<td>10</td>
<td>c</td>
<td>(3,\lambda)</td>
</tr>
</tbody>
</table>

LZ78 is less flexible than LZ77 in that it only permits references to certain addresses in the parsed part of \( x \). In LZ77 any address within \( W_i \) can be chosen. LZ77 and LZ78 form the basis for many popular compression algorithms, among them **zip** and **gzip**. In the context of string complexity, the number of steps in LZ77 and the size of \( D \) in LZ78 may be used as approximations for LZ production complexity.

This concludes the discussion of the Lempel-Ziv family of measures. The following section discusses T-complexity, a complexity measure which expresses complexity as the number of steps required to build a string – very much like the Lempel-Ziv family of measures. T-complexity serves as the basis for the T-information and T-entropy measures, the latter of which was used for the majority of experiments carried out in this thesis. The relationship between these measures will be discussed in Section 2.5.
2.2 Complexity measures

2.2.5 T-complexity

T-complexity is a string complexity measure proposed by Titchener [102, 103, 104]. Like the Lempel-Ziv family of measures discussed in the previous sections, it describes the complexity of a string \( x \) in terms of a number of elementary steps required to build \( x \).

In T-complexity, a basic building step comprises the choice of a known codeword in a prefix-free code\(^2\) and the generation of new codewords by prefixing the chosen codeword to other elements of the prefix-free code. In a recursive copy-and-append process called T-augmentation, which chooses suitable prefixes and combination codewords at each step, this is repeated until \( x \) itself becomes a codeword of the prefix-free code. As T-augmentation is thus key to the understanding of T-complexity, this section proceeds with a description of T-augmentation. Note that the introduction to T-complexity (and T-information theory in general) presented here largely follows the one given in Günther [42].

T-augmentation

T-augmentation is defined as follows:

\[ C^{(k)}_{(p_1, \ldots, p_n)} = \bigcup_{i=0}^{k} \{ p_i a \mid a \in C \setminus \{ p_i \} \} \cup \{ p_i^{k+1} \} \]  \hspace{1cm} (2.2)

The T-augmentation of \( C \) with T-prefix \( p \) and T-expansion parameter \( k \). More general, a series of \( n \) T-augmentations with T-prefixes \( p_1, \ldots, p_n \) and T-expansion parameters \( k_1, \ldots, k_n \) is denoted by

\[ C^{(k_1, \ldots, k_n)}_{(p_1, \ldots, p_n)} = \bigcup_{i=0}^{k_n} \{ p_i^{k_n+1} \} \]  \hspace{1cm} (2.3)

In particular, \( C^{(0)}_{()} = C \) for \( n = 0 \). \( n \) is called the T-augmentation level.

\[ A^{(k_1, \ldots, k_n)}_{(p_1, \ldots, p_n)} \]

\[ \text{A set } A^{(k_1, \ldots, k_n)}_{(p_1, \ldots, p_n)} \text{ derived from } A \text{ via a series of } n \text{ T-augmentations is called a T-code set.} \]

\[ \text{The 3-tuple } (A, (k_1, \ldots, k_n), (p_1, \ldots, p_n)) \text{ is called the T-prescription of the T-code set } A^{(k_1, \ldots, k_n)}_{(p_1, \ldots, p_n)}. \]

A prefix-free code is a code in which no codeword is a prefix of another (distinct) codeword. Note that prefix-free codes are also often referred to as prefix codes. See Berstel and Perrin [19].
T-code sets are, among other codes, known for the following properties: T-code sets are uniquely decodable and complete, i.e., it is not possible to add another codeword without violating the prefix-freeness of the code. Furthermore, it has been noted that T-code sets exhibit a strong tendency towards self-synchronisation. Proofs for these properties are beyond the scope of this thesis. However, they may be found in [41] and references therein.

Like other variable length codes, such as Huffman codes, T-code sets can be represented graphically as decoding trees. This permits an alternative description of T-augmentation in terms of an algorithm which operates on T-code decoding trees, as shown in Algorithm 2.2.2

**Algorithm 2.2.2 T-augmentation of T-code decoding trees**

Require: Let $T_{n-1}$ denote the decoding tree for the T-code set $A_{(p_1, \ldots, p_{n-1})}^{(k_1, \ldots, k_{n-1})}$. If $n = 1$, the tree is trivial, i.e., there is a leaf node for each character in the alphabet $A$ and all leaf nodes are directly attached to the root node.

1: Pick a leaf node $p_n$ of $T_{n-1}$. Choose $k_n \in \mathbb{N}^+$.
2: $T_n = T_{n-1}$. Define a counter $i = 1$.
3: while $i \leq k_n$ do
4: Replace the leaf node $p_n^i$ in $T_n$ with a copy of $T_{n-1}$
5: $i = i + 1$
6: end while
7: $T_n$ is the new decoding tree for the T-code set $A_{(p_1, \ldots, p_n)}^{(k_1, \ldots, k_n)}$.

The operation performed by this algorithm is demonstrated in the following example:

**Example 2.2.4 (T-augmentation)**

Figure 2.1 illustrates three successive T-augmentations over a binary alphabet $A = \{0, 1\}$ and the resulting T-code sets by means of decoding trees: the trivial T-code set is denoted by the leftmost decoding tree whose leaf nodes are simply 0 and 1, the alphabet’s characters. For the first T-augmentation $p_1 = 1$ and $k_1 = 1$ are chosen. In the decoding tree, this T-augmentation replaces the leaf node 1 with a copy of the entire current decoding tree. The result is depicted as $A_{(1)}^{(1)}$ in Figure 2.1.

Now $p_2 = 10$ and $k_2 = 2$ are chosen for the second T-augmentation. For the first iteration with $i = 1$, a copy of the entire current decoding tree replaces leaf node 10. During the next (and last) iteration with $i = k = 2$, a second copy of the current decoding tree replaces the intermediate leaf node $p_2^2 = 1010$, resulting in the decoding tree depicted as $A_{(1, 10)}^{(1, 2)}$ in Figure 2.1.

For the last T-augmentation, $p_3 = 0$ and $k_3 = 2$ are chosen. In the first step ($i = 1$), the decoding tree $A_{(1, 10)}^{(1, 2)}$ is attached in place of the leaf node 0 in a copy of itself. In the resulting intermediate tree $A_{(1, 10, 0)}^{(1, 2, 1)}$, the leaf node $p_3^k = 00$ is replaced with a second copy of $A_{(1, 10)}^{(1, 2)}$, resulting in the final T-code decoding tree $A_{(1, 2, 2)}^{(1, 10, 0)}$.  


Observe how the T-augmentation algorithm described above prepends \( k_i \) copies of \( p_i \) to the longest codewords at T-augmentation level \( i \) in the decoding trees: as an immediate consequence, the longest codewords in T-code decoding trees are always of the form \( p_n^{k_n} p_{n-1}^{k_{n-1}} \ldots p_1^{k_1} a \), with \( a \in A \). In other words: all the information required to reconstruct a T-code set by T-augmentation is encoded in the longest codewords of the set.

Nicolescu and Titchener [80] demonstrated that T-code sets are in fact fully specified by any one of their longest codewords. Furthermore, their work shows that a unique T-code set can be found for every finite string, thus establishing a duality between finite strings and T-code sets. Titchener [100, 101] and Nicolescu and Titchener [80] also developed an algorithm which retrieves the T-prefixes and T-expansion parameters involved in the construction of arbitrary strings. This algorithm, which is called T-decomposition, will be discussed in the next section. The T-expansion parameters the algorithm retrieves are then used in the computation of T-complexity, which is presented on page 33.

**Reversing T-augmentation: the T-decomposition of a string**

Let \( x \) be a finite string over the alphabet \( A \). The T-decomposition of \( x \) is a process which parses \( x \) into a set of T-prefixes \( p_1, \ldots, p_n \) and a set of T-expansion parameters \( k_1, \ldots, k_n \). From Definitions 2.2.1 and 2.2.2 we already know that the information about T-prefixes, T-expansion parameters and the underlying alphabet is sufficient to construct a T-code set.

Algorithm 2.2.3 shows the original T-decomposition algorithm as published by Titchener and Nicolescu [100, 101], [80]. Note that the T-decomposition of a string \( x \) is independent of the last symbol \( a \) of \( x \), as \( x \) is only one instance of the \#\( A \) longest codewords in the associated T-code decoding tree and
the T-decomposition for all these codewords is identical. For this reason, the alternative writing \( x = x'a \)
is used in Algorithm 2.2.3. \( x' \) is called the T-handle of \( x \).

**Algorithm 2.2.3 T-decomposition**

**Require:** Finite alphabet \( A = \{a_1, a_2, \ldots, a_{n-1}, a_n\} \) and \( x' a \in A^+ \).

1: Initialise a counter \( n = 0 \).
2: Parse \( x' a \) left-to-right over \( A \) into a sequence of tokens \( w_0 a_1 w_0 a_2 \ldots w_0 \xi(0) = x' a \), so that \( \xi(0) = |x' a| \). Thus each symbol is parsed into a token.
3: while \( \xi(n) > 1 \) do
4: \( n = n + 1 \).
5: Identify \( p_n \) as the penultimate token \( w_{(n-1)} \xi(n-1) - 1 \). Identify the maximum length \( k_n \) (in tokens) of the run \( p_n^k \) of \( k_n \geq 1 \) tokens that are copies of \( p_n \) which ends in \( w_{(n-1)} \xi(n-1) - 1 \).
6: if the run identified in the previous step starts with the leftmost token \( w_{(n-1)} 1 \) then break
7: Parse \( x' a \) left-to-right into a new set of tokens \( w_{n-1} \ldots w_{n+1} \xi(n) \) using the following rule: existing consecutive tokens \( w_{(n-1)} \ldots w_{(n-1)} l + 1 \) with \( 1 \leq l \leq k_n \) are merged into a single token if \( w_{(n-1)} \ldots w_{(n-1)} l + 1 = p_n^k \). Note that this merger must be greedy.
8: end while
9: return \((p_1, \ldots, p_n)\) and \((k_1, \ldots, k_n)\).

There are faster algorithms for T-decomposition than Algorithm 2.2.3. An early acceleration was implemented by Titchener and Wackrow in the form of tcalc [107]. At the time of writing, the fastest algorithm for T-decomposition is ftd. ftd [95] was developed by Speidel and Yang as an improvement over two previous algorithms and terminates in \( O(n \log n) \) time. It uses complex data structures to optimise its performance. While ftd was used for T-entropy measurements in this thesis, a detailed discussion of ftd is beyond the scope of this thesis. Details about this algorithm can be found in [118].

To illustrate the operation of the T-decomposition algorithm, an example run is shown step-by-step in Example 2.2.5. For readability, dots are used as token delimiters.

**Example 2.2.5 (T-decomposition)**

Consider a string \( x = 00101011 \). Initially, T-decomposition parses \( x \) into 0.0.1.0.1.0.1.1. During the first run through the loop, \( p_1 = 1 \) and \( k_1 = 1 \) are computed. As the run of \( p_1 \) tokens ending in the penultimate token does not start with the leftmost token, 0.0.10.10.11 is computed as new parsing of \( x \) and the loop iterates.

Now \( p_2 = 10 \) and \( k = 2 \) are computed from the current parsing of \( x \). Still the run of \( p_2 \) tokens ending in the penultimate token does not start with the leftmost token, so 0.0.10.10.11 becomes the new parsing of \( x \) before the loop iterates another time.

\( p_3 = 0 \) and \( k_3 = 2 \) are found during the third iteration of the loop. This time, the run of \( p_3 \) tokens ending in the penultimate token starts with the leftmost token, so the program terminates, returning \((p_1, p_2, p_3) = (1, 10, 0)\) and \((k_1, k_2, k_3) = (1, 2, 2)\).

Note that each token is a codeword from \( A^{(k_1, \ldots, k_n)}_{(p_1, \ldots, p_n)} \).
2.2 Complexity measures

The weighted number of T-augmentation steps as complexity measure

Titchener [104, 102, 103, 105] proposed the T-expansion parameters \( k = k_1, \ldots, k_n, |k| = n \) retrieved from the T-decomposition of a string \( x \) to be used as indicators for the complexity of \( x \). He thus defined the T-complexity of \( x \) as a function of the T-expansion parameters \( k_i \).

**Definition 2.2.3 (T-complexity)** Consider a string \( x \) whose T-expansion parameters from T-decomposition are \( k_1, \ldots, k_n \). The T-complexity of \( x \) is defined as follows:

\[
C_T(x) = \sum_{i=1}^{n} \log_2(k_i + 1)
\]

(2.4)

The unit in which a T-complexity of a string \( x \) is expressed is \( \text{taugs} \), the weighted number of T-augmentations required for the construction of \( x \).

**Example 2.2.6 (T-complexity)**

The T-expansion parameters retrieved from a T-decomposition of the string \( x = 00101011 \) in Example 2.2.5 were \( (1, 2, 2) \). For the computation of \( C_T(x) \), these are used as input for Equation 2.4:

\[
C_T(00101011) = \sum_{i=1}^{3} \log_2(k_i + 1) = \log_2(18) = 4.17 \text{ taugs}
\]

Some properties of T-complexity are:

- \( \prod_{i=1}^{n} (k_i + 1) \) is the number of internal nodes in the corresponding T-code decoding tree. So \( C_T \) is the number of bits required to address the internal nodes in the T-code decoding tree uniquely.

- Although a closed form for an upper bound for the maximum T-complexity of strings of length \( |x| \) has not been formulated yet, Titchener [102] observed that the upper bound of T-complexity appears to be asymptotically equivalent to the logarithmic integral \( li(|x|) \), a convex and invertible function on the interval \((1, \infty] \).

The next section discusses entropy measures. The entropies presented there all have linear upper bounds, a circumstance which makes it difficult to relate T-complexity directly to entropy measures. It is possible to “linearise” T-complexity by applying the inverse logarithmic integral \( li^{-1}(|x|) \). This will be discussed in more detail in Section 2.5.
2.3 Entropy measures

Entropy measures are the second large family of information measures. This section starts out with a brief discussion about the origin of the term *entropy*. The discussion then revisits the relation between entropy in a thermodynamical sense and in an information-theoretical sense. Specific information-theoretical entropies such as the Shannon entropy and $N$-gram entropy will be presented before the chapter concludes with an introduction to T-entropy, an entropy measure derived from T-complexity.

2.3.1 Entropy in Thermodynamics and Information Theory

The term *entropy* was first introduced by Rudolf Clausius in the context of thermodynamical processes. Clausius’ view of such processes was of a macroscopic nature, i.e., it did not involve information about individual particles participating in a process. Boltzmann subsequently pioneered the field of statistical mechanics in physics. With the tools provided by statistical mechanics, he interpreted thermodynamics at a microscopic level by attributing probabilities to the states of the particles involved in thermodynamical processes. For instance, the probability $p_1$ of observing a single ideal gas molecule in a specific section of an otherwise evacuated container is proportional to the ratio between the section’s volume $V_s$ and the container’s volume $V_c$. Following this thought, the probability $p_n$ of observing all molecules in the section when the container contains $n$ molecules can be expressed as:

$$p_n = \left(\frac{V_s}{V_c}\right)^n$$

Clearly, for $V_s < V_c$ and large $n$ the likelihood of observing all molecules inside $V_s$ quickly becomes very small. The observations Boltzmann made at a microscopic level thus still allowed him to make statements about macroscopic systems involving many particles like those examined by Clausius.

2.3.2 Shannon entropy

Shannon used the term entropy for his now well-known information measure after it was pointed out to him (cf. Petz [84]) that his measure happened to have the same mathematical form as Boltzmann’s entropy. In his paper [88], Shannon defined the measure $H(x)$ as the uncertainty in predicting the next event of a discrete random variable $x$ for which only the event probabilities $p_1, \ldots, p_n$ are known.
Shannon states three criteria which he requires for this measure. These are only met if $H$ has the form of Boltzmann’s entropy:

$$\begin{align*}
H &= -K \sum_{i=1}^{n} p_i \log p_i \\
&= 2.5
\end{align*}$$

$K$ is merely a constant which depends on a unit of measure and on the base of the logarithm chosen. In the information-theoretical context of entropy, typical units of measure are *bits per bit* or *bits per symbol* if the base of the logarithm is 2. If the natural logarithm is used instead, a typical unit would be *nats* per symbol. In the thermodynamical sense of entropy, $K$ is Boltzmann’s constant, the unit of measure is *Joule per Kelvin* and the base of the logarithm is $e$.

Some of the properties of $H$ are:

- $H = 0$ iff all except one the $p_i$ are zero (the non-zero $p_i$ is 1, of course).
- $H$ is maximal iff $p_1 = p_2 = \ldots = p_n = \frac{1}{n}$
- The triangular inequality holds: for two random variables $x$ and $y$, $H(x, y) \leq H(x) + H(y)$, with equality iff $x$ and $y$ are independent.

Probably *Shannon entropy rate* would describe the nature of this information measure better than the term *Shannon entropy*. As said before, Shannon entropy merely is a function of the probability distribution of an information source, not a function of the strings generated by the source. Hence, Shannon entropy cannot be used to compute the information content of strings.

### 2.3.3 $N$-gram entropy

Shannon explored situations for which the $p_i$ are not known but have to be derived from observation. Shannon entropy relies on the assumption that consecutive symbols generated by an information source are not correlated. This assumption, however, does not hold for many real information sources, one example being natural languages such as English. Shannon thus extended his concept of entropy by proposing the *$N$-gram entropy* [89]: here, a sequence of which the first $L$ symbols are known is broken up into blocks of sizes $1 \ldots L$ called $N$-grams. Shannon uses the number of occurrences of each $N$-gram to estimate the $p_i$ for each of the possible $N$-grams. For each $N$, this results in a probability distribution for

\[^3\text{nats} = \text{natural units}\]
which the $N$-gram entropies $F_N$ can easily be calculated with Equation 2.5. As $N$ approaches infinity, Shannon suggests using the limit of the resulting series of values as the overall entropy of the sequence:

$$H = \lim_{N \to \infty} F_N$$  \hspace{1cm} (2.6)

This approach introduces a number of difficulties if the sequence is finite, as is the case in all practical situations. Firstly, the limit does not exist for finite sequences. Secondly, the entropies become meaningless as soon as $N$ reaches the order of $\log L$: at this point it is no longer possible for all conceivable patterns to appear and the resulting probability estimates become increasingly biased.

### 2.3.4 Entropy estimators

Complexity measures for which an upper bound is known permit the definition of linearised measures to estimate the entropy (cf. last paragraph of the following section, page 37). Lempel and Ziv describe an upper bound for the LZ production complexity in [67] as $f(n) = n/\log n$. Szczepański et al. [97] and Amigó et al. [16] used this upper bound to define an entropy estimator for LZ production complexity by linearising LZ production complexity with $1/f(n) = \log n/n$. Szczepański et al. refer to this estimator as *normalised complexity*. This thesis uses this entropy estimator for reference purposes in Chapters 4 and 9.

### 2.4 Commonalities and differences between the two approaches

Sections 2.2.1 and 2.3.2 discussed two fundamental concepts for information measurement, Kolmogorov-Chaitin complexity and Shannon entropy. Both of these concepts can unfortunately not be used in practice to determine the exact information content of strings, Kolmogorov-Chaitin complexity because it is not computable and Shannon entropy because it merely expresses the entropy rate, not the actual entropy.

Approximations which can be used for finite strings exist for both concepts. However, both types of approximations introduce two sources of error:

- Complexity approximations adopt specific algorithms, which directly violates the conditions imposed by Kolmogorov and Chaitin. Various methods are used to improve computational time
Commonalities and differences between the two approaches

- In LZ77, the window size is limited. As discussed in Section 2.2.3, this results in situations where patterns are recognised as new because previous occurrences of the pattern are beyond the window boundary.

- In LZ78, only certain positions in the string are used as starting points for the search. Again, patterns may be recognised as new because previous occurrences of the pattern do not start at a position that LZ78 recognises.

- In T-complexity, repetitive patterns may not be recognised in their full length because of synchronisation delays (see Günther [43] and Günther and Titchener [44]).

Shannon entropy approximations suffer from the limitation of the string length $|x|$ in two ways:

1. $x$ cannot accommodate all possible $N$-grams for $N$-grams of length $N \geq \log(|x|)$. Even a source that emits all $N$-grams with equal probability cannot produce a string of length $|x|$ in which all $N$-grams occur with equal frequency. Therefore, for $N \geq \log(|x|)$, the estimation of probabilities from frequencies is problematic.

2. The number of occurrences of an $N$-gram in a string is always an integer, the probabilities estimated from this are always rational numbers. Hence, an exact determination of the probabilities is not possible for sources whose probabilities include irrational numbers. However, for both rational and irrational probabilities the precision of the estimation becomes more accurate as the length $|x|$ of the sample string is increased.

Attempts to relate approximation techniques of different approaches to each other often rely on the behaviour of the associated information measures as the string length approaches infinity. For example, a paper by Titchener, Gulliver, Nicolescu, Speidel and Staiger [106] confirms that T-complexity is asymptotically equivalent to Shannon entropy, i.e., it exhibits an $n/\log(n)$ behaviour.

The length of strings considered in this thesis is not necessarily in the convergence zone of some of the measures. For such (relatively short) strings, at least one fundamental “incompatibility” exists between the measures of both domains. This will be discussed in the following paragraph.

Consider a string $x$ of maximum Shannon entropy. In the Shannon sense, it should be possible to extend $x$ with constant and maximum entropy rate. Now consider a string $y$ of maximum complexity. At least the Lempel-Ziv family of measures and T-complexity do not permit $y$ to be extended with constant and maximum complexity rate, due to the convex upper bounds which these measures exhibit.
As already mentioned in Section 2.3.4, one possible solution to this problem is to “linearise” an affected complexity measure in order to develop an information measure which is compatible with Shannon entropy. This is possible if a function $f$ can be found that describes the behaviour of the upper bound of a particular complexity measure. Two possible strategies can be used for linearisation:

- If the corresponding inverse $f^{-1}$ exists, it can be used for the purpose of linearisation. The T-information measure presented in the following section linearises T-complexity in this way.

- The measure can be linearised by applying $1/f$. This is what Szczepański et al. used for the definition of normalised complexity.

### 2.5 T-information and T-entropy

As already mentioned on page 33, Titchener conjectured that the upper bound of T-complexity is asymptotically equivalent to $li(|x|)$. In an effort to acquire a Shannon entropy compatible information measure from T-complexity, Titchener [102] thus proposed the inverse logarithmic integral $li^{-1}(x)$ as a suitable function for linearising T-complexity. This motivated the following definition:

**Definition 2.5.1 (T-information)** The T-information of a string $x$ is defined as the inverse logarithmic integral of the T-complexity of $x$:

$$I_T(x) = li^{-1}(C_T(x))$$

The logarithmic integral $li(x)$ is defined as follows (cf. Wolfram MathWorld [114] or Wikipedia [116]):

$$li(x) = \int_0^x \frac{dt}{\ln(t)}$$

$li(x)$ has a singularity at $x = 1$ and a positive zero at $x = 1.4513692\ldots$, the Ramanujan-Soldner constant. T-information is measured in *nats* if the natural logarithm is used in the computation of the inverse logarithmic integral. If the base of the logarithm is chosen as 2 instead, the unit is *bits*.
Example 2.5.1 (T-information)

Following Example 2.2.6, the T-information for the string $x = 00101011$ is 4.92 nats, which is equivalent to 7.10 bits.

T-information as such does not describe an entropy rate. Hence, Titchener further derived a measure called $T$-entropy from T-information. T-entropy is defined as the gradient of T-information.

**Definition 2.5.2 (T-entropy)** The T-entropy of a string $x$ is defined as the gradient of the T-information of $x$ with respect to the length $|x|$ of $x$:

$$H_T(x) = \frac{dI_T(x)}{d|x|}$$

A special case of this is the average $T$-entropy:

$$H_T(x) = \frac{I_T(x)}{|x|}$$

As T-entropy is differentiated across the string length $|x|$, it is measured in nats per symbol if the natural logarithm is used for the computation of T-information. If the base of the logarithm is 2 instead, units such as bits per bit or, more generally, bits per symbol are commonly used.

From Section 2.2.5, we know that the computation of T-complexity (and thus T-entropy) involves variable-length blocks which depend on patterns in the input. This is a fundamentally different approach to measures such as Shannon’s $N$-gram entropy, where the block size $N$ is fixed.

Example 2.5.2 (T-entropy)

The average T-entropy of the string $x = 00101011$ used in Examples 2.2.6 and 2.5.1 is about 0.62 nats/symbol which corresponds to approximately 0.89 bits/bit.

T-entropy is the preferred information measure in this thesis, because it has a number of attractive properties:

- With ftd (cf. page 32), the T-entropy of a string can be computed efficiently.
• Ebeling, Steuer, and Titchener showed that T-entropy exhibits a close relationship with known entropies such as the Kolmogorov-Sinai entropy (Pesin entropy) of the logistic map [32].

In the following chapters bits/byte is used as the unit for T-entropy, because in this thesis T-entropy is generally computed from byte sequences.

2.6 Concluding remarks

This chapter presented information measures from two families, the family of complexity measures and the family of entropy measures. The upper bounds of both types of measures behave somewhat differently, which makes it problematic to relate measures from different families to each other. Eventually, a recently developed information measure called T-entropy was presented.

T-entropy was chosen as central information measure in this thesis, because it can be computed efficiently and it has been shown to estimate known entropies accurately. In the following chapters, other information measures such as LZ78 and LZ76 are mainly used for reference purposes.
The information measures discussed in the previous chapter operate on strings. The output of any discrete information source can thus be coded in a suitable way to examine its entropy/complexity with information measures. This thesis uses information measures to estimate the information density of network stream data. Section 3.1 discusses suitable places for tapping network stream data. Section 3.2 then looks at the type of incidents that one may wish to regard as network events. Section 3.3 describes the network environments considered in this thesis and Section 3.4 discusses a variety of network events that may be detectable with information measures.

3.1 Possible information measurement points within a network

Computer networks provide many possible points of access, so the question arises as to which points are most suitable for information measurement. Depending on the measurement focus, one quickly finds that
some nodes within a network are more suitable than others: For example, traffic produced by file servers is usually not meant to leave a local area network. For this reason, internal nodes can be interesting candidates to measure file server activity in this scenario. In contrast to this, a large part of this thesis deals with events beyond network operator control, i.e., traffic passing through border gateways is of interest. Naturally, this makes border gateways better candidates for monitoring than internal nodes, for several reasons:

- All traffic entering and leaving the network has to pass through a border gateway.
- Border gateways are the first points of contact for any unwanted/malicious traffic directed against a target network.
- During times of normal operation, the cumulative view of the network traffic as seen at border gateways permits statements concerning the normal traffic profile for an organisation or business.

This resolves the question as to where to capture network stream data for information measurement. The following three sections address the question as to what to look out for in the data.

### 3.2 Events and non-events

Whether one would want to regard an incident in a network as a network event largely depends on the nature of the network under observation. This may be motivated by way of two simple examples:

**Example 3.2.1 (Small home network)**

Consider a small residential home network of an average family comprising two computers connected to the Internet through a DSL router. Assume that all family members are out at work or school during the day. Further assume that the computers are generally used in the evening for typical Internet activities such as e-mail exchange, web browsing, chatting, or online-games.

- The normal state of the network described here is silence, i.e., absence of any packets, and does not indicate a failure.
- Publications on Internet background radiation such as Pang et al. [81] indicate that it is common for most if not all computers connected to the Internet to receive unrequested packets. As no Internet services are provided in the example network, such unsolicited packets could be considered events in this context, because they could signify an attack of some sort. On the other hand one might argue that Internet background radiation is normal and may therefore not be considered an anomaly.
The failure of a computer constitutes an event. This may be observed in the distribution of internal IP addresses. Of course, this may be misleading if one of the computers does not suffer from a failure but is simply switched off.

This example shows that the distinction between events and non-events is sometimes neither simple nor clear-cut.

**Example 3.2.2 (Medium-sized business network)**

Now consider a typical business network with 1000 client computers and 20 servers. During the day, users generate traffic by exchanging e-mails, VOIP calls, web browsing and data transfers on VPN. During the night, the client computers are refreshed with automatic software updates; administrators carry out work such as database migrations that would impair operation during the day. If the business considered has a web presence of general interest (consider a newspaper website, for instance) there will be constant web and e-mail traffic, independent of the time of day.

- The absence of any packets for an extended period of time would be considered a network event, because it might indicate a failure.
- In such a company network, the mere appearance of unrequested packets at the border gateway(s) would hardly be considered a network event. In this case, it is normal to see service requests appearing at the border gateway all the time. On the other hand, the arrival of more traffic than the border gateways can handle may be an event, e.g., some kind of attack.
- The failure of a single user terminal device in the company may not be considered a network event, because its impact on the overall state of the network is negligible.

In these two scenarios, the definition of a network event clearly depends on the network context. For this reason, the next section will briefly describe the network contexts of the experiments in this thesis. This will be the foundation for Section 3.4, which classifies a selection of incidents in the network of the University of Auckland into events and non-events.

### 3.3 Networks considered in this thesis

The data for the experiments in this thesis was collected at the University of Auckland (UoA). At the time of writing, the University of Auckland has approximately 38000 students and 5600 full time staff members, see UoA website [9] and UoA 2008 calendar [56]. This size is comparable to many universities.
worldwide. The UoA has a 16 bit address prefix, i.e., an address pool of 65534 IPv4 addresses. About 14000 of the addresses from this pool were assigned to network devices at the time data was captured for experiments.

UoA has a network policy which greatly influences both the IP address distribution and the traffic composition observed at the border gateway. Some of the points this policy stipulates are:

- Network users do not usually have direct access to the Internet. All web traffic is directed through a web proxy server. Most e-mail traffic is managed by a central mail server. Other traffic originating from user applications must be directed through a SOCKS proxy server.
- The use of peer-to-peer file sharing applications is not permitted on the university network.
- DNS responses for external hosts are cached locally, so most DNS requests are not forwarded through the border gateway.
- During the time of data collection, the policy also provided for network accounting on external connections. Students had relatively tight traffic allowances for their course work and had to pay to keep Internet access once an allowance had been used up. In the author’s own experience talking to fellow students, students were generally quite careful with their Internet usage. It is highly probable that this had an influence on the traffic composition.

For instance, the external traffic volume generated by the university network is likely to be lower than the volume a typical ISP would have to set aside for 14000 end users, as an effect of the network policy. Furthermore, the IP address distribution observed at the border gateway is strongly biased in favour of a few proxy server addresses. For this reason, a second set of trace files was collected at an internal campus gateway. The next paragraphs describe the limitations imposed by the biased address distribution and motivate the collection at the internal campus gateway.

### 3.3.1 Trace files collected at the UoA border gateway

Data captured on a network may be stored in trace files for later evaluation. The trace file format used in this thesis will be discussed in more detail in Section 4.1.

Between July and September 2005, the first set of trace files for this thesis was captured at the UoA border gateway. At this time, the university had a single 100 Mbps Ethernet connection to the Internet. In the afternoons of the capture period, about 8000 packets/s passed through the border gateway. Considering
the link speed, the maximum data volume one may expect is 12,500,000 bytes/s. A typical PC with a PCI bus and a fast disk drive is able to process this volume, in particular if only truncated packets are captured, as is the case in this thesis.

One class of network events discussed in this thesis is hardware faults which affect a large fraction of the traffic seen at a border gateway. In a university context, this might be the disconnection of a department or an entire campus due to a router failure or a link cable being cut. As no such event took place during the capture period, simulation was considered as an alternative. A simulation can in principle be carried out (at least for TCP and UDP) by filtering the following packets from the trace files:

1. All outbound packets from the IP address range affected.
2. All inbound TCP packets to the IP address range affected, if the TCP ACK flag is set.
3. All DNS traffic for the affected IP address range except incoming DNS requests (outgoing DNS requests are covered in Point 1).

Because of the network policy at the University of Auckland, this simulation strategy turned out to be problematic: for trace files captured at the border gateway, various proxy servers made it impossible to decide whether a particular web, e-mail or DNS packet originated from a subnet that should have been filtered for simulation purposes. Hence, for a large part of the traffic, such a simulation was impossible. Experiments were carried out with the small amount of residual traffic, but no judgement was possible as to whether the results of these experiments are representative for the overall university traffic.

### 3.3.2 Additional trace files collected at an internal campus gateway

To address the proxy server problem, a second set of trace files was captured at the border gateway of the Tamaki campus, one of the three large campuses at UoA. The location of this border gateway guarantees the preservation of the actual IP addresses in web and e-mail traffic, as both the web proxy and the e-mail servers are located at the UoA city campus.

Unfortunately (in the context of this thesis), there are dedicated DNS servers for the Tamaki campus, so even at the Tamaki campus gateway it is not possible to decide where a particular DNS request actually originated from. However, DNS traffic generally only contributes a small fraction to the overall traffic. Considering this, the tainting effect of (wrongly) unfiltered DNS responses in a network hardware fault simulation might be regarded as being negligible.
As the Internet traffic generated at Tamaki is less than that of the entire university, one might expect the trace files captured at this campus gateway to be smaller than those described in the previous subsection. However, this is not so: client computers at Tamaki use (among others) central AFS file servers on the city campus, which contribute to significant additional traffic that one would not observe at the university border gateway: here up to 19700 packets per second were observed.

In order to model a hardware fault as discussed in Section 8.8, traffic of some subnets at the Tamaki campus was filtered from trace files captured at the Tamaki campus gateway. A second experiment (cf. Section 8.8, page 133) filtered packets according to the transport protocol. In a real network, such an event could arise from misconfiguration, for example.

### 3.3.3 Third-party trace files: NLANR PMA Special Traces Archive

A worm event is discussed by example of the SQL Slammer worm outbreak [31], [86], [33] in Section 8.4. Trace files of this outbreak are available from NLANR’s *Special Traces Archive* website [12, 11]. These trace files are anonymised, i.e., all IP addresses have been replaced by private IP addresses incrementing in order of occurrence: 10.0.0.1, 10.0.0.2, and so on. SQL Slammer uses random addresses for its spread, therefore the entropy of the address fields should usually be high.

The anonymisation just described lowers the entropy of the address fields considerably, because similar address prefixes appear frequently in the stream (10.0.0, for instance). A distinction of internal and external IP addresses was also not possible with this anonymisation scheme.

Apart from this, anonymised trace files such as NLANR’s SQL Slammer trace files often contain no payload. For SQL Slammer, the payload is constant and would thus be an excellent property to exploit in entropy measurement. These kinds of modifications often impair entropy measurement with anonymised trace files.

This concludes the description of the network contexts considered for the experiments in this thesis. The following section discusses a selection of events considered in these contexts.

### 3.4 Events considered in this thesis

In the literature, many taxonomies for network events can be found (e.g., [111], [73], [72], [62], [59], [47], [20], [48], [65], [15], [57]). The central question for this thesis is whether particular types of network
events are generally detectable with information measures or not.

The question as to which events should be considered in this thesis partly found its answer in the network context examined: an administrator’s point of view was adopted to select events with sufficient potential to disturb the network operation at the University of Auckland. A second consideration for the selection of suitable events may be derived from the knowledge we have about information measures from Chapter 2: Information measures react to the presence (or absence) of patterns in a string. This narrows down the selection of suitable events: it only makes sense to consider events that influence the information rate of the packet stream. If the information measure used behaves like Shannon entropy, i.e., the dominance of a few symbols leads to a low information rate and a uniform symbol frequency leads to a high information rate, two scenarios are possible:

1. High entropy monitoring: In this scenario, one chooses observables from the stream that generate a relatively high information rate. Drops in the information rate may be interpreted as network events. An example for a suitable observable is the length field from the IPv4 header: the distribution of the values in this field is relatively uniform, so that the associated information rate is relatively high. An event such as the SQL Slammer worm outbreak [31], [86], [33] introduces many packets of identical length into the stream. During the event it is thus easier to predict the next observed packet length, hence the information rate of this observable drops.

2. Low entropy monitoring: Here, one chooses observables from the stream that generate a relatively low information rate. Sudden rises in the information rate may indicate the presence of a network event. As an example one might consider the protocol field from the IPv4 header: the majority of packets observed in the UoA network uses TCP as the transport protocol. Hence, the predictability associated with this field is relatively high, implying a low information rate. Again, consider an event such as the SQL Slammer worm outbreak: as SQL Slammer uses UDP to spread, the sudden appearance of many UDP packets in the stream raises the information rate of the protocol field.

There are events whose footprint in the data stream is too small to be recognised, even if such an event contributes new patterns to the stream. An example is the class of Denial-of-Service (DoS) attacks which exploit security holes to compromise a target machine. Such attacks, hereafter referred to as low volume DoS attacks often consist of only a few packets. One example is an exploit for the Mailslot Ring0 Memory Corruption [14], [13] security hole which existed in most Windows operating systems: vulnerable hosts could easily be crashed by sending four forged packets to the target host.
If a low volume DoS attack is initiated manually for a single target machine, it is unlikely to have any measurable effect on the entropy of a relatively busy network link such as the one at UoA. For this reason, no efforts were made to detect this type of attack. However, it is easily possible to initiate an automated low volume DoS attack against many target machines in a network. In such a scenario, the footprint of the attack may be detectable because the contributed pattern occurs frequently in the packet stream. The more pronounced footprint of such attacks arises from the address scanning layer which is wrapped around the actual attack. Hence, one might categorise this type of attack into the class of scanning attacks which will be discussed in Section 3.4.4.

In summary, this thesis only considers large-scale events that cause significant changes in the traffic patterns. The following sections examine properties of certain common large-scale network events and discuss their detectability with information measures. Note that information-theoretical analysis makes no assumptions about the specific fields affected by events, although the small selection of events that were chosen for discussion below might suggest otherwise: As described in the outset of this thesis (page 13), it is the particular strength of this approach that it considers all available information. In order to improve the detectability of certain event types, users have the choice to limit this information by applying filters. However, this may impair the detectability of other events if the affected fields are partly or completely filtered out. Filter design strategies are presented in Chapter 6 and corresponding examples are presented in Chapter 8.

Specifically, the event types selected for consideration in the following sections are: DDoS attacks, Windows Messenger spam, Internet worm outbreaks, scans, SSH dictionary attacks, SYN floods, ICMP floods and network equipment faults.

### 3.4.1 Distributed Denial of Service (DDoS) attacks

DDoS attacks aim to consume nearly all of one or more resources a service provider needs in order to provide its service. In a network context, a popular attackable resource is the limited bandwidth Internet link of a target computer. If an attacker manages to congest a link with fake service requests, legitimate service requests will be dropped by the congested routers. As a result, the target machine responds to plenty of fake service requests. Legitimate requests are likely not to reach the target host. Should a legitimate request happen to reach the target host, it is still quite possible that the response is dropped on the congested link.
For a DDoS attack to be successful, an attacker needs to generate large amounts of fake requests. In many cases, this is achieved in a two-step scheme:

1. The attacker scans many hosts on the Internet for an exploitable security hole. An attack agent is remotely installed and started on vulnerable machines.
2. To start an attack, the attacker triggers as many attack agents as possible to send fake service requests to the target machine.

As the time between the setup of the attack agents and an actual attack may be relatively long, it is particularly difficult to trace DDoS attacks back to their real origin. Locating attack agents is usually not a simple task, because many attack agents fill the IPv4 source address field with random values. IPv4 does not make any provision for packet traceback, and proposals to utilise rarely used fields from the IPv4 header partly break the protocol (cf. Song and Perrig [92]).

In order to conceal a DDoS attack from an information measure, the DDoS attack would have to produce an entropy very similar to that observed under normal conditions. This is generally difficult to achieve for a remote attacker without feedback from the measurement point. If one considers an attack against a web server, for example, an attacker basically has the choice between two extremes:

1. Minimising the entropy by sending many identical requests, or
2. Maximising the entropy by sending many random requests.

In both cases, it is likely to have observables in the stream showing a different entropy than under normal conditions. For this reason, DDoS attacks are interesting event candidates to be considered in this thesis.

### 3.4.2 Spam

At the time of writing, the majority of e-mail traffic on the Internet consists of unsolicited bulk e-mails, also simply referred to as spam. Legal efforts made towards confining this problem are restricted to national boundaries. Spammers therefore moved away from local spamming agents towards remote spamming agents for anonymisation. Like the attack agents used for DDoS attacks, spam agents may be installed through security holes on vulnerable hosts.

At least the spam messages intended for human readers (as opposed to those intended to merely compromise spam filters) contain sections of repetitive content. One might claim that spam should be considered
as a network event in this thesis. However, there are two reasons why spam detection is not possible with the trace files collected for this thesis:

1. As with normal e-mail, any repetitive content in spam is preceded by a number of mail headers and potentially some garbage text. The trace files in this thesis use a record size of 80 bytes (see Section 4.1). This is too small to read far enough into the payload to find repetitive patterns contributed by spam e-mail.

2. Spam represents the majority of the overall e-mail traffic, therefore spam events are the norm, not the exception. In other words: any alarm controlled by a spam detector would constantly be triggered, which is useless.

For these reasons, e-mail spam was not considered as a network event in this thesis. However, there is another form of spam, Windows Messenger spam, which is considered here and described in the subsequent section.

3.4.3 Windows Messenger spam

Spamming is more or less common in many communication infrastructures, one example being the Windows Messenger service [77]. In Windows operating systems, administrators may use this service to inform users about maintenance work, shutdowns, and so on in the form of a pop-up window. The service uses UDP for message delivery and port 1026 by default; sometimes 1027 is used instead.

Before Windows XP SP2, spammers were able to abuse this service to deliver messages by sending forged packets to UDP port 1026. Windows runs on the majority of PCs worldwide, and many users do not care about security updates. It is therefore likely for spammers to deliver at least some messages successfully by brute-force scanning entire IP address ranges, i.e., the same forged payload is sent to each IP address on a network.

Such brute-force messenger service spamming attempts may influence several observables in a packet stream:

- UDP is less commonly used than TCP. The sudden appearance of many UDP packets is an indicator for a possible network event.

- Messages sent through the Windows Messenger service appear to have a fixed preamble before the actual text. Thanks to the small UDP header, up to 38 bytes of this preamble are visible in the trace
files collected for this thesis (cf. page 81), so the UDP payload may be an interesting observable.

- Packets belonging to Messenger spam generally use UDP destination ports 1026 or 1027, so the destination port field becomes an interesting observable for this type of network event.

As discussed before, address scanning is usually part of Messenger spam events. More general scans are discussed in the next section.

### 3.4.4 Scans

Scans are a very common network event. The previous section briefly introduced IP address scanning as part of Windows Messenger spam events, but there are more scan types. Usually, the purpose of scanning is to locate hosts with some sort of vulnerability. Some typical parameters classifying a scan are:

- **Frequency:** There are high rate scans and low rate scans. Low rate scans are generally difficult to detect.
- **Scanned field:** Two examples are TCP/UDP ports (“port scan”) or IP addresses (“address scan”).
- **Systematic vs. chaotic scanning:** Some scans probe ports/addresses in some systematic order, other scans use random values for addresses and ports.

Attackers not having a clear idea which security leak to use for breaking into remote hosts may run a port scan on a machine. Depending on the ports found to be open, the attacker may then decide which exploits to try.

Attackers who aim to compromise many machines with a particular exploit will rather run address scans and keep the destination port fixed.

During high rate scans, many packets passing through the border gateway of a target network may exhibit similar or even identical values in certain TCP, UDP, and IP fields. Furthermore, the payload of the forged packets may be constant. As usual, one may expect such fields to lower the information rate of the packet stream. Scans using randomised values for ports and/or addresses may either be detectable with low entropy monitoring, or one may observe fields that cannot easily be randomised, such as the IPv4 protocol field.

The next section discusses worms, an often very aggressive network event type.
3.4.5 Worms

Worms are self-replicating computer programs which obtain access to computers through security holes. Once a computer is infected, the worm uses that computer to spread to other vulnerable hosts. This way, an exponential growth in the number of infected systems is possible. This was the case for the SQL Slammer worm outbreak (cf. Moore, Paxson, Savage, Shannon, Staniford, and Weaver [31] and Ray [86]), for example: in the early phase of its outbreak, the number of infected computers doubled about every 8.5 seconds.

Due to the self-replication, one may expect the worm payload to appear as repetitive pattern on affected links. Furthermore, IP fields such as length or protocol and TCP/UDP fields such as the destination port may be constant for all worm packets. Hence, these fields may be suitable observables for information measurement.

Studies about entropy-based detection of computer worms have, among others, been carried out by Wagner [109], Wagner and Plattner [110], Lall, Sekar, Ogihara, Xu, and Zhang [64], Tang and Dong [99], and Eimann, Speidel, and Brownlee [33].

3.4.6 SSH dictionary attack

SSH (SecureSHell) is a popular software on Unix and Linux machines. By default, a host providing an SSH service permits encrypted terminal and file transfer facilities. Apart from that, SSH tunnels may be set up to secure otherwise insecure TCP communications.

Logging into an SSH server typically requires a username and password. If insecure username/password combinations exist on a target machine, an SSH dictionary attack may permit an attacker to break into that machine.

To run an SSH dictionary attack against a host, attackers use a database consisting of common usernames and insecure passwords which may match user credentials on a target system. An example is the use of the combination test/test as a dummy test user for some system. If a host with such a user account runs a public SSH service, the system is vulnerable to this sort of attack.

It lies in the nature of this attack type that many attempts with different username/password combinations are necessary for a successful break-in. A section from a log file which recorded a number of break-in attempts is shown below:
As a consequence, many SSH connection request packets are entering the network of the target host. As SSH has its own dedicated TCP port, usually 22, the TCP destination port becomes an interesting observable for information measurement during break-in attempts. Furthermore, the overall effect on the information rate of the network stream is amplified by the outgoing TCP acknowledgment (TCP ACK) packets. Note that IP source address spoofing would not make sense for this type of attack as the attacker’s client program needs the responses to establish the communication with the server.

These properties make SSH dictionary attack interesting candidates for network event detection via the information rate.

### 3.4.7 SYN flood attacks

TCP connections are established with a three-way handshake. The initial part of this is an empty TCP packet with a raised SYN flag. This asks the receiver machine to reserve resources for the new connection. If no further communication occurs on the connection, the connection times out. This connection timeout is usually much longer than the time required to establish a connection.

SYN flood attacks exploit this by quickly opening new connections to a remote host, until the maximum number of open connections to that host is reached. In this situation, the victim host is no longer able to open any new connection.

For a successful SYN flood attack, SYN packets need to be sent to the victim host at a high rate, otherwise it may not be possible to occupy all of the target system’s resources. An Information measure may respond to SYN flood attacks, because many largely identical packets pass through the border gateway of the target host’s network. The SYN flag is an obvious observable one may wish to monitor during this type of attack.
3.4.8 ICMP floods

The purpose of ICMP is to transport control and error messages which may arise from dropped packets or packets which cannot be processed. One program that uses ICMP is ping. Its purpose is to check whether a remote computer is switched on and what the round-trip-time to this computer is. To do this, ping typically uses 40 byte packets which are sent out once per second by default.

For an ICMP flood, an attacker may increase the payload size from 40 bytes to the maximum of 65507 bytes1 and send these large packets at the maximum packet rate to a target host. This way, an attacker may be able to congest a victim’s communication channel with ICMP traffic, causing regular traffic to be dropped. If the victim machine responds to the incoming ICMP packets, the congestion effect is further amplified by the victim machine’s echo replies.

ICMP floods may influence observables such as the IPv4 length and protocol fields of the payload which should permit information rate-based detection of this attack type.

A Smurf attack (cf. Shay [90], page 339) is a special kind of ICMP flood attack: during a smurf attack, ICMP echo request packets with forged source addresses are sent to the broadcast address of a network. If such an attack succeeds, a copy of the echo request is sent to each host on the network, multiplying the attack traffic by the number of hosts. The hosts, in turn, may respond to the requests, which may further amplify the impact of the attack.

3.4.9 Failure of network equipment

Faulty or misconfigured network equipment might cause certain patterns in a network stream to appear less frequently than they would normally. Typical examples for observables affected by this type of event are the IPv4 address fields, the TCP and UDP port fields or the IPv4 protocol field.

To an extent, this type of event constitutes a special case among the network events discussed here: while all the events mentioned above add packets to a network stream, network equipment faults usually remove certain packets (and thus patterns) from a network stream.

---

1Together with the 8 bytes for the ICMP header and the 20 bytes for the IPv4 header, the overall length of the packet becomes 65535 bytes, the maximum possible length under IPv4.
3.5 Conclusions

This chapter suggested network border gateways as suitable points for network event observation. The definition of what one may wish to consider a network event largely depends on the network context. As the data for this thesis originates from the network of the University of Auckland, the network context of the university was examined in detail to obtain a clearer notion of suitable network events to consider: only large scale events are considered in this thesis. The last sections of the chapter investigated some event types that may be detectable with information measures in more detail.

Now that we have the necessary tools and a more focussed view of what to look out for, it is time to throw a stone into the water and examine the information rate of raw IPv4 traffic. This will be discussed in the next chapter.
Viewed at a basic level, network streams between a sender and a receiver are simply strings of bytes and can therefore immediately serve as input for information measures. In the literature numerous approaches for examining network traffic using complexity measures have been discussed, e.g., in Wehner [112], Cilibrasi and Vitányi [29], and Wagner and Plattner [110], but there is also a large number entropy-based approaches, such as Kulkarni, Bush, and Evans [60], Lakhina, Crovella, and Diot [63], and Ziviani, Gomes, Monsores and Rodrigues [121]. Apart from the work undertaken for this thesis, T-entropy has not been used for studies on network traffic before. For this reason, T-entropy of network traffic is the primary object of study in the present chapter. More specifically, the T-entropy of plain network traffic, filtered network traffic, and plain network traffic with artificial network events added to it will be discussed. These entropy examinations are preceded by a brief discussion concerning details of network traffic capture.
4.1 Capturing network traffic

The meaning of the terms sender or receiver varies with the observed layer in the protocol stack. In the context of the present chapter, they refer primarily to the network layer which today is typically implemented with IPv4. In this thesis, mainly data from the network layer is used for entropy measurement. In principle, data from the data link layer could be used as well. However, at least for Ethernet, the information contributed by this layer is largely replicated at the network layer.

There are numerous techniques for network traffic capture. One common method is the use of an Endace DAG card [2, 8], which is specifically designed not to drop packets. Each packet record generated by a DAG card also features a very precise GPS (hardware) timestamp.

Due to the unavailability of a working DAG card at the time of network traffic capture, tcpdump [7] was used for data capture in this thesis. tcpdump is a program usually found in Linux or BSD distributions. It is based on libpcap [7], a packet capture API and works with typical consumer network cards. As libpcap captures traffic at the data link layer below the network layer, the data returned contains extra information beyond that used by the network layer. More specifically, it carries the entire header of the data link layer protocol. libpcap itself also adds a PCAP packet header. In our case, the data link layer protocol is Ethernet, so the data in these headers may be summarised as follows:

- The PCAP packet header provides packet length information and a software timestamp. The packet length information is redundant as the IPv4 header contains its own length field. The software timestamp provides useful information for event detection, as it allows packet interarrival times to be computed. This will be discussed at a later stage in Section 6.4.1.

- The Ethernet header only contains a frame type field and two MAC addresses. The latter could be used to determine the direction of packets. In our case, this information is also redundant, because the direction of the packets can also be derived from the addresses in the IPv4 header. However, it may be useful for anonymised trace files, because in this case the direction of packets can usually no longer be determined from the IPv4 addresses (cf. Section 3.3.3). This thesis only considers IPv4 packets, so the frame type field provides no useful information either: it is constant if only IPv4 packets are considered. Like the IPv4 protocol field, this field may be useful in scenarios where other data link layer protocols, such as ARP, are involved.

Figure 4.1 shows the structure of records generated by tcpdump when capturing traffic from an Ethernet interface with the maximum capture length chosen to be 80 bytes per packet. Together with the PCAP
4.2 T-entropy of raw IPv4 traffic

As mentioned earlier, neither the extra data in the PCAP packet header nor that of the Ethernet header is considered for entropy measurement in this thesis. The only exception is a test carried out in the discussion of the PCAP packet header timestamp field in Section 6.4.1.

Removing the 30 bytes occupied by the PCAP packet header and the Ethernet header from each record leaves a maximum of 66 bytes of raw and possibly truncated IPv4 packets. These string snippets usually cover the entire IPv4 header, transport protocol header, and at least a portion of the payload, if any. This preparation permits entropy measurements of pure network layer traffic. The following section examines this in more detail.

4.2 T-entropy of raw IPv4 traffic

It is natural to expect network events to influence the complexity of network streams rather than of individual packets. For instance, the signature of individual SQL Slammer worm (cf. Ray [86]) packets might not exhibit particular changes in the entropy compared to signatures of packets belonging to regular traffic. What might rather influence the entropy is the sheer flood of SQL Slammer packets in the overall
network stream, because SQL Slammer’s signature appears over and over again.

For this reason, no entropy tests are carried out with individual packet signatures. Instead, a number of \( N_S \) consecutive packet signatures are concatenated into a single string before the T-entropy of this string is computed. These input strings for information measures are also referred to as *traffic samples* in this thesis. Output values of the information measure will be referred to as *entropy samples* or *complexity samples*, depending on the measure considered. *Sample* may be used for simplicity if either no clear distinction between entropy samples/complexity samples and traffic samples is required or where this is obvious from the context.

**Trace file notation in this thesis (part 2)** Given a traffic sample size \( N_S \) and a trace file \( T_{date} \), \( N_T(T_{date}) = N_R(T_{date}) / N_S \) specifies the number of traffic samples \( T_{date} \) yields.

The next question is now: what would be a sensible size for \( N_S \)? Chapter 7 discusses this question in detail. For the purposes of the present chapter, a rough estimation suffices.

With typical HTTP or SMTP transactions involving dozens of packets, \( N_S \) should be sufficiently large to get a reasonable balanced sample of such and other packets. A tentative upper bound for \( N_S \) can be derived from the packet rate of the observed link, which is approximately 8000 packets/s for the trace file used in this chapter: some events, such as port scans, may be shorter than one second and their impact may be averaged out if \( N_S > 8000 \) is chosen. Thus, a value between several hundred and 8000 appears sensible for \( N_S \). At this point \( N_S = 2000 \) is simply chosen as a heuristic guess between the lower and upper bounds for \( N_S \). The author likes to stress at this point that \( N_S = 2000 \) is simply an initial value. No claims are made that this value is optimal in any sense. Section 7.5 will discuss the choice of \( N_S \) in further detail. The paper by Feinstein et al. [36] motivated the use of a sliding window scheme to select trace file records for a traffic sample.

Traffic volume variation over time can cause time jitter in a simple sliding window scheme breaking a trace file into sequences of \( N_S \) packet records each. In order to avoid this, the main sliding window scheme used in this thesis considers windows starting at regular time intervals. In other words: for the present chapter, each traffic sample \( s_i \) reflects the 2000 packet records following a timestamp \( t_i \). In order to avoid excessive overlap or gaps in the data considered for traffic samples, the interval time \( \tau = t_{i+1} - t_i \) may be chosen as follows:

\[
\tau = \frac{\Delta(T_{date})}{N_T(T_{date})} = \frac{\Delta(T_{date})}{N_R(T_{date}) / N_S}
\]  

(4.1)
4.2 T-entropy of raw IPv4 traffic

Trace file $T_{20050722}$, which is used for the following experiments, spans three hours and consists of $N_R(T_{20050722}) = 82.5\ million$ packet records, which yields $N_T(T_{20050722}) = 41250$ traffic samples of $N_S = 2000$ packet signatures each. Hence, $\tau = 10800\ s/41250 \approx 0.26\ s$.

The following experiments were thus undertaken with this initial choice of $N_S$. Details of the data sampling schemes considered during this project will be discussed in Chapter 7.

**Experiment 4.2.1 (Information measurement of raw IPv4 traffic)**

**Description:** This experiment examines the T-entropy, normalised complexity, and LZ production complexity of IPv4 stream data from trace file $T_{20050722}$. As mentioned above, the data in this trace file yields $N_T(T_{20050722}) = 41250$ traffic samples with 2000 packet signatures each, covering an observation period of three hours. We compute the T-entropy, normalised complexity, and LZ production complexity for each traffic sample (i.e., obtain the corresponding entropy samples/complexity samples) and plot the results.

**Expectation:** Depending on the activity on the network during the intervals covered by each traffic sample, the traffic samples may contain patterns. The information measures may react to these patterns.

**Observation:** Figure 4.2, 4.3, and 4.4 show the results of the experiment.

![Figure 4.2: T-entropy of raw IPv4 traffic in $T_{20050722}$. Most of the time, the T-entropy is in the range of 3 bits/byte. Two deviations from this background entropy are visible: there is a long T-entropy drop between $2144\ s$ and $2631\ s$ down to approximately 0.8 bits/byte and regular short drops down to about 1.25 bits/byte occurring once in approximately $430\ s$. The drops are possibly the result of events.](image)

All three figures exhibit three noticeable features:

- For most of the samples, the entropy/complexity of the traffic in $T_{20050722}$ falls into narrow bands: around 3 bits/byte for T-entropy, 2.5 bits/byte for normalised complexity, and
Entropy of Network Traffic

Figure 4.3: Szczepański’s normalised complexity (cf. [97] and Section 2.3.4), an entropy estimator for LZ production complexity, using the same raw IPv4 traffic samples as Figure 4.2.

![Figure 4.3: Szczepański’s normalised complexity](image)

Figure 4.4: LZ production complexity of raw IPv4 traffic in T20050722. The complexity drops are very similar to the entropy drops in Figure 4.2 and 4.3, except for minor details: for example, a small drop at \( \approx 10700 \) s visible in the entropy plots is not visible here.

![Figure 4.4: LZ production complexity](image)

25000 production steps for LZ76. These may be seen as the entropy/complexity levels of “regular traffic” at the UoA border gateway at the time when the trace file was recorded. They will be referred to as background entropy/background complexity hereafter. Chapter 6 will discuss background entropy levels for subsets of the IPv4 data.

- For all measures, the samples between 2144 s and 2631 s exhibit a long steep entropy/complexity drop down to about 0.8 bits/byte for T-entropy and normalised complexity and 7000 production steps for LZ production complexity.

- At regular intervals of approximately 430 s, the traffic samples cause all measures to produce steep entropy/complexity drops. These drops reach levels of approximately 1.25 bits/byte (T-entropy), 1.1 bits/byte (normalised complexity), and 12500 production steps (LZ production complexity).
4.3 T-entropy of filtered IPv4 traffic

In conclusion, a number of steep drops in the T-entropy, normalised complexity, and LZ production complexity of typical IPv4 traffic were observed. Figure 4.3 and 4.4 demonstrate that the overall behaviour of T-entropy is highly similar to established measures such as Szczepański’s normalised complexity and LZ production complexity. The normalised complexity measure yields a background entropy 0.5 bits/byte lower than T-entropy. This difference decreases for lower entropies. Apart from this, any differences in the outcomes of this experiment are very much in the detail.

The following section addresses the question as to which features in the traffic are responsible for the entropy/complexity drops observed in the previous experiment.

4.3 T-entropy of filtered IPv4 traffic

After carrying out Experiment 4.2.1, a detailed examination of the intervals showing entropy drops in the plot was carried out with trace file $T_{20050722}$. During the drop for the entropy samples between $2144$ s and $2631$ s, many outbound TCP SYN packets with constant IP addresses and destination port 5406 were observed. Likewise, during the regular entropy drops, many inbound UDP packets with destination ports 1026 and 1027 are present. To verify that the drops are indeed caused by the observed packets, the following experiment repeats Experiment 4.2.1 with the suspect packets filtered out.

Experiment 4.3.1 (Entropy of filtered network layer traffic)

**Description:** This experiment consists of two tests, each of which is essentially a repetition of Experiment 4.2.1, with a modification of the data passed to the entropy measure: in the first test (Test 1), all packet records using 5406 as source or destination TCP ports are removed. In the second test (Test 2), all packet records accessing UDP destination ports 1026 or 1027 are removed. For each test, a T-entropy plot is generated.

**Expectation:** If the filtered packets in the first test are related to the entropy drop for the samples between $2144$ s and $2631$ s in Experiment 4.2.1, this entropy drop should no longer be present in the current experiment. Likewise, one may associate the filtered UDP packets with the regular entropy drops present in Experiment 4.2.1 if Test 2 does not reproduce these drops.

**Observation:** Figure 4.5 shows the result of Test 1: the entropy drop observed in Experiment 4.2.1 for the samples between $2144$ s and $2631$ s disappears, apart from a comparatively small residual effect. One may therefore attribute the entropy drop to the filtered packets. The properties of the filtered
Entropy of Network Traffic

packets suggest that the packets belong to a SYN flood attack. The residual effect may be the result of congestion during the SYN flood event: legitimate packets may have been dropped during the attack, triggering TCP’s congestion control mechanism for some flows, which, in turn, may have resulted in smaller IP packets and smaller advertised window (cf. page 99) values in TCP. With an increased number of smaller packets, all these effects taken together may have introduced unusual patterns into the packet stream, hence slightly raising the T-entropy during the event. Note that this is interpretation and speculative, however.

The result of Test 2, presented in Figure 4.6, shows a largely identical picture to the one presented in Experiment 4.2.1, except that the regular entropy drops disappear. Here, the payload signature of the filtered packets was found to be consistent with Windows Messenger spam events.

Experiment 4.3.1 thus indicates a cause-effect relationship between the filtered packets, i.e., the events, and the T-entropy drops. Both event effects can individually be removed by filtering the associated events.
prior to entropy computation, leaving the T-entropy of the remaining samples largely unchanged.

Another way of confirming this relationship is to inject artificial traffic into existing trace file traffic, as discussed in the following section.

### 4.4 T-entropy of IPv4 traffic with added artificial events

The previous experiment showed how entropy drops can be removed by filtering certain packet types, and established a cause-effect relationship between the filtered packets and the entropy drops. This section presents an experiment taking the opposite approach: an artificial network event will be injected into the trace file used in the previous experiments and the event’s effect on the T-entropy will be observed. Before turning to the actual experiment, a brief description of event generation in trace files is given.

Compared to other event simulation approaches, such as *pktd* proposed by Gonzales and Paxson [39], the approach presented below is rather simple: it multiplexes synthetic event traffic with real traffic stored in a trace file, disregarding any effects the simulated event might have had on the real traffic. Ziviani et al. [121] appear to use a similar approach, but no detailed description is given in their publication.

Simulation of an entire network to enable realistic network event simulation would not have been an alternative to the approach chosen here, for the following reasons:

- The traffic recorded at the border gateway of the UoA is quite complex with thousands of participating hosts, both inside and outside the university network and a large number of different protocols. This would have been too complex to be simulated with a testbed network or tools such as NS-2 [4].
- Network event simulation in the real environment would not have been possible due to the sensitive nature of the monitoring point.

Network simulation is generally not unproblematic. Floyd and Paxson [37] describe various difficulties and pitfalls in this area.

#### 4.4.1 Generating artificial events for injection into a trace file

One possible approach for network event simulation is the injection of event packet records at certain times into a stream of trace file packet records. The following paragraphs specify certain times more
precisely. A possible injection mechanism is discussed subsequently.

Real network event packet arrival times are, among others, influenced by routing queue delays, quality of service constraints, and different path lengths. For this reason, network event packets do generally not arrive at fixed intervals, even if a fixed interdeparture time applies to the packets at the source.

As a real-life example, the interarrival time of the Windows Messenger spam packet records in $T_{20050722}$ was analysed. Figure 4.7 presents a logarithmic histogram of this analysis.

Between $0 \mu s$ and approximately $80 \mu s$ the distribution rises quickly to its maximum. For interarrival times larger than $80 \mu s$, the overall shape of the distribution falls exponentially, here roughly linearised by the logarithmic vertical axis.

A complex yet regular peak pattern along its slope between $200 \mu s$ and $1200 \mu s$ is also observed. The detailed interpretation of this pattern is not entirely clear. Among others, it might be the result of a time-division multiplexing (TDM) policy implemented in one or more nodes along the packet path, it could arise from the sampling time slices of the network card used, or it could be a granularity effect caused by interspersed packets of minimal/maximal size. The last of these presumptions is motivated by the IPv4 length distribution shown in Figure 6.5.

For the purpose of event simulation in this thesis, we may observe that the overall shape of the distribution can roughly be modelled as a Poisson distribution: a Poisson-distributed jitter is thus applied to the arrival times of artificially generated event packets. The detail peak pattern in Figure 4.7 was not considered in event simulations undertaken here. The considerations of Paxson and Floyd [82] on the subject of Poisson modelling are not applicable here, because Windows Messenger spam uses UDP and might originate from
many different sources.

The actual packet injection may be carried out with a multiplexing algorithm operating on two queues, as shown in Figure 4.8, one queue carrying artificial network event packets (blue), the other carrying packets originating from the trace file (yellow). The algorithm can decide which packet to output next by comparing the timestamps of the packet records waiting at the queue outputs.

![Image of a multiplexing algorithm and two packet queues](image.png)

**Figure 4.8:** A multiplexing algorithm and two packet queues may be used for simple event simulation. One queue carries event packets (lower queue, blue), the other carries packets originating from a trace file (upper queue, yellow). The packet records of both queues are sorted according to their timestamps (here shown as comparatively small integers). At each step, the algorithm selects the packet record with the smaller timestamp value from the two inputs and forwards it to the output. I.e., the packet record with timestamp 27 will be picked next.

This type of event simulation is not perfect, because certain effects that occur in reality cannot be simulated: for instance, modelling congestion effects arising from artificial events is difficult, because a simulation of TCP’s congestion control mechanism would require bookkeeping across many packets.

The event generator used in this thesis produces artificial packet records at an adjustable fixed rate, i.e., a preliminary arrival timestamp is assigned to each generated packet record. Before the packet records enter the event queue, network delays are modelled as a Poisson-distributed jitter which is added to the arrival timestamps. The generator then multiplexes the artificial packets into a stream of real trace file packets as shown in Figure 4.8. Congestion and multiplexing effects are not considered. Certain fields of the generated packet signatures may contain dynamic data to permit simulation of port scans, for example. For the experiment in the following subsection, the generator was set up to imitate the SYN flood packets that were observed in the previous experiments.
4.4.2 Example: injection of an artificial SYN flood

Consider the following experiment:

**Experiment 4.4.1 (Entropy of plain network layer traffic with an artificial network event)**

**Description:** This experiment examines the effect of an artificial SYN flood event injected into the data provided by the same trace file that was used for the previous experiments. Artificial event packets are generated at intervals of 400 $\mu$s, which is lower than the rate of the events observed in the previous experiments (roughly 60 $\mu$s for the Windows Messenger spam events and about 10 $\mu$s for the SYN flood event). The interval between the 13th and the 14th Windows Messenger spam related T-entropy drop in Figure 4.6 (samples between 5293 s and 5722 s) was arbitrarily chosen as the location for the artificial event. As before, a T-entropy plot of the result was generated.

**Expectation:** If the assumption that the introduction of patterns into the packet stream lowers the entropy holds true, the T-entropy for the affected samples should drop. More specifically, the drop should be smaller than those observed in the previous experiments as the packet rate is lower.

**Observation:** Figure 4.9 confirms the expectation. The largely constant patterns inserted by the artificial packets dilute the overall complexity of the packet stream, leading to a lower T-entropy for the affected traffic samples.

![T-entropy of IPv4 traffic in T20050722 with an artificial SYN flood event in the samples between 5293 s and 5722 s. The steep entropy drops during the artificial network event are caused by short periods of silence in the affected traffic samples, which the event generator now fills exclusively with repetitive packet signatures.](image)

This experiment verifies the observation made in the previous experiment in the opposite direction: before, intervals of low T-entropy were seen as indicators for certain patterns in the trace file. In this
experiment, an event was injected into the data provided by the trace file to confirm that this lowers the T-entropy.

Note the steep drops inside the time interval of the artificial event, which did not appear in this form in the plot presented in Figure 4.2. The reason for this is that the trace files collected at the UoA regularly exhibit intervals of silence sometimes spanning several $\tau$; in some trace files up to 14 s of silence were observed. The sliding window scheme discussed at the beginning of this chapter causes sequences of identical traffic samples. In a magnification of Figure 4.2 these appear as a short entropy “plateau”, i.e., a short horizontal line.

The origin of these periods of silence is unknown. Among others, possible explanations would be packet drops by the network card, bus access issues with the network card or lack of CPU time for tcpdump.

The event packet generator used here does not recognise or respect intervals of silence in a trace file. Hence, the generator may fill the intervals with artificial event packets. This, in turn, leads to long sequences of regular patterns in the resulting packet stream and pronounced entropy drops in the affected areas.

### 4.5 Conclusions

The experiments in this chapter show that at least some network events can be detected by measuring the T-entropy of IPv4 stream data. Without particular refinements, two different events were detected in Experiment 4.2.1. The low entropy sections of the trace file clearly exhibit accumulations of certain event-related packet types.

In order to confirm that the entropy drops are really caused by these packet types, two approaches can be taken: either filter them out and show that the effect disappears or generate them artificially and show that an entropy drop is the resulting effect. This chapter pursued both of these avenues. The results of the respective experiments both indicated a cause-effect relationship between the accumulations of event-related packet types and phases of low entropy. These results suggest that entropy measurement of IPv4 stream data is a viable way for the detection of at least some network events.

This confirms observations with different entropy/complexity measures obtained by Wagner and Plattner [110], Wehner [112], Kulkarni [60] and Feinstein [36].

The experiments carried out in this chapter also showed that in the absence of outstanding events, the background entropy was distributed in a neighbourhood of 3 bits/byte. The following chapter examines
this phenomenon in more detail.
5.1 Types of noise

Estimation of string information is generally subject to “noise”-like phenomena. At least three possible sources of noise can be isolated for consideration:

- Synchronisation noise: Most computable information measures produce a range of different values for a string shifted bytewise in a cyclic fashion, although no information is added or removed from the string. As an example, Figure 5.1 shows this type of noise for T-entropy with a 100000 byte string that is cyclically shifted bytewise until the initial state is reached again. This way, no information is added or removed from the string. The string is taken from the start of $T_{20050722}$.

In the case of Figure 5.1, most of the noise extends over a T-entropy band from roughly 2.88 bits/byte to 3.01 bits/byte. As the law of large numbers applies here, the difference between the upper and
Figure 5.1: T-entropy of bytewise cyclic shifts of a 100000-byte input string. Although no information is added or removed from the string, there is little correlation between shifts and T-entropy. Hence, the shifting process resembles the generation of noise.

lower bounds of this band increases as the length of the input decreases.

- Entropy computation noise: While there is strong correlation between different entropy measures, this correlation is not perfect, i.e., there is no linear relationship, due to inherent misestimation for each measure.

- Information source noise: Adjacent finite string samples captured from an information source vary naturally in complexity and thus in entropy.

From hereon, this thesis will refer to all these effects as *entropy estimation errors* or simply as noise.

Figure 5.2 shows that the histogram of the entropies in Figure 5.1 closely resembles a $(\mu, \sigma)$ normal distribution with $\mu = 2.939$ and $\sigma = 0.017$. A slight bias towards lower entropies is visible, which may originate from the presence of weak events in the data.

Figure 5.2: The histogram of T-entropy values plotted in Figure 5.1 roughly follows a normal distribution.
5.2 Signal-to-noise ratio

For the LZ78 complexity measure, the complexity histogram also resembles a normal distribution. In a normal distribution, outliers beyond four standard deviations are very uncommon. Based on this, one may define a measure to discriminate signals from noise in the next section.

5.2 Signal-to-noise ratio

Processing data samples from some source with an entropy measure results in a sequence of scalar values. One may interpret this sequence as a description of the status of the data source over time. In the case of networks, a sudden entropy drop may indicate the presence of an event. Thus, event detection is essentially a classification of scalar values into either events or non-events. Any such classification attempt carries the risk of incorrect classification, i.e., producing false positives and/or false negatives.

If we focus our view on computer networks as a data source, we may expect the absence of events to be the norm and the presence of events to be the exception. In this kind of situation, one can regard an event as a signal one wishes to detect. The distribution of non-event values then forms a background signal, which may be regarded as a kind of noise.

Communication engineering provides the concept of the signal-to-noise ratio (SNR) as an indicator for correct signal/noise classification, see Sklar [91] or Shay [90]. In engineering, the quantities signal and noise are usually either described by their voltage amplitudes or their power levels. Similar to the analogy of event/non-event and signal/noise, one may regard entropy values as voltages incident on a receiver input, which motivates Definition 5.2.1. The required quantities for the definition are the mean and the standard deviation of non-event entropy samples as well as the median of event entropy samples. All of these can easily be computed from a sequence of entropy samples. Note that most SNR values in this thesis are within the range of 1 and 30. Therefore this thesis uses straight ratios rather than the otherwise common logarithmic (dB) representation.

Definition 5.2.1 (Signal-to-noise ratio) Let $S$ be the set of entropy samples of a trace file. We may partition $S$ into two subsets, $E$ and $E'$, by choosing a discrimination threshold $b$ with $\max(S) > b > \min(S)$ so that $E = \{s \in S | s \geq b\}$ (for low entropy monitoring) or $E = \{s \in S | s \leq b\}$ (for high entropy monitoring). $E' = S \setminus E$. We call $E$ the set of event samples and $E'$ the set of non-event samples. We thus define the signal-to-noise ratio $snr(S, b)$ as

$$snr(S, b) = \frac{|\mu(E') - median(E)|}{\sigma(E')}$$

(5.1)

1 See page 23 for notation.
where $\mu$ and $\sigma$ are mean and standard deviation, respectively. $\sigma(E')$ may be taken as a measure for the residual noise itself.

For normal distributed quantities, one commonly chooses four standard deviations $\sigma$ as a cutoff to discriminate between normal samples and outliers. We may thus formulate the following sufficient condition to discriminate events from non-events in terms of the SNR we just defined:

$$ snr(S, b) \geq 4 \quad (5.2) $$

To demonstrate the practical impact of this condition, Figure 5.3 shows a one hour trace file with an artificial SYN flood event (SNR=4) located between 600 s and 720 s. This event is clearly visible.

The following example shows that the SNR of real events may well be significantly higher than four.

Example 5.2.1 (SNR of the Windows Messenger spam events in Experiment 4.2.1)

This example determines the SNR of the Windows Messenger spam events in Chapter 4. The three hour trace file used there, $T_{20050722}$, also contains a SYN flood event. This SYN flood event would distort the result if the entire trace file were considered here, because the SYN flood entropy samples would be part of the set $E$. As the SYN flood event takes place during the first hour of $T_{20050722}$ only, the last two hours of this trace file, $T_{20050722_{15-17}}$, are considered here.

To discriminate event samples from non-event samples, $b=2.38$ bits/byte was chosen. Application of Equation 5.1 to the entropy samples yields $snr(S_{T_{20050722_{15-17}}}, 2.38) = 21.66$. With $\sigma(E')=0.073$ bits/byte,

\footnote{Note that filtering the SYN flood event would not be appropriate here, as the SYN flood had an impact on the background entropy (as discussed in Experiment 4.3.1). This impact would increase $\mu(E')$ and $\sigma(E')$.}
the residual noise is relatively low in comparison to other experiments in this thesis.

This finishes the discussion about noise effects in entropy measures. The SNR tool specified in this chapter will be used for evaluation purposes in the next chapters: the following chapter introduces the concept of packet mapping and effects on the entropy of network events can be evaluated via SNR computation. In Chapter 7 the SNR will be used to optimise the amount of stream data reflected by a single entropy sample.
6.1 Motivation

One more or less obvious disadvantage of raw packet signatures as input for entropy measures is the large data volume involved. This naturally slows down the entropy computation and may be difficult to sample or handle. In Chapter 4, we already used a sliding window algorithm to observe changes in the entropy over the observation period. This sliding window algorithm also keeps the amount of data in each traffic sample at a manageable size. However, there are two main problems to be considered here:

- Is it necessary to process all of the data in each packet signature? If not, which field (or combinations of fields) in the data are worth looking at?

- What is a good window size?
The first of these questions is addressed in the present chapter. Questions concerning the sliding window will be dealt with in Chapter 7.

In order to avoid effects on the entropy arising from different window sizes, the sliding window delivers a constant number of bytes per traffic sample and should cover a constant number of packets. This precaution should be taken to keep the expected entropy estimation error in the same range for all traffic samples. For raw network stream data, this condition cannot be met, because the packets vary in size.

Figure 4.1 presented a typical packet record produced by `tcpdump`. In most cases, the IPv4 component of such a record contains two headers, the IPv4 header and the header of the transport protocol used. Certain fields in these headers have very stable content (IPv4 version field, IPv4 header length, for instance) while others fluctuate substantially (checksum fields and payload, for instance). Hence, the individual types of fields contribute different amounts of information to the stream. By extracting only certain fields from the packet stream, the mapping procedure described in the present chapter aims to condense this information in order to meet four goals:

1. **Data volume reduction.** A reduction in the amount of data passed to the information measure. Depending on the information measure chosen, this results in at least linear processing speed improvements.

2. **Prevention of cancellation effects.** Chapter 3 already introduced the concepts of low and high entropy monitoring. Cancellation effects occur when fields from both domains are considered in an entropy computation: while some fields lower the entropy, other fields raise it at the same time.

   Figure 6.1 demonstrates that the IPv4 protocol and IPv4 source address fields are candidates for this kind of behaviour during Windows Messenger spam events: the entropy of the protocol field (red samples) rises as more UDP packets appear while the entropy of the last two octets of the source address field (blue samples) drops as many repetitions appear. The SNR of the last two octets of the source address field is 14.06 with a discrimination threshold of \( b = 2.25 \). The SNR for a combined measurement (green samples) considering the protocol field and the last two octets of the source address is lower: 13.16 with a discrimination threshold of \( b = 1.75 \). In the worst case, such cancellation effects may conceal events.

3. **Constant expected entropy estimation error.** Chapter 5 discussed estimation errors inherent in computable entropy and complexity measures. While it is not possible to avoid these errors, it is at least possible to keep the expected entropy estimation error in the same range for all traffic samples extracted from a network stream while measuring. To do this, each traffic sample should reflect a fixed number of packet records and be constant in length. The simple concatenation of
6.1 Motivation

Figure 6.1: The entropies of the IPv4 protocol field (red samples) and the last two octets of the IPv4 source address (blue samples) for $T_{20050722}\rightarrow17$ during the Windows Messenger spam events, the entropy of the protocol field rises, due to a more balanced mix of TCP and UDP packets, while it drops for the last two octets of the external IPv4 address at the same time. If both fields are jointly considered as observables (green samples) for the detection of Windows Messenger spam events, the impact of the event on the entropy is less pronounced than it is with the last two octets of the IPv4 source address only: the SNR of the blue samples is 14.06 (discrimination threshold $b = 2.25$), while it is only 13.16 (discrimination threshold $b = 1.75$) for the green set of samples.

raw IPv4 records used in Chapter 4 fails to meet this condition because raw IPv4 records vary in size. Mapping makes it particularly easy to condense information from raw IPv4 records into fixed size data blocks, which can then be passed on to an information measure.

4. Reduction in redundant information. There are correlations between various header fields. For instance, the central web proxy server of UoA has the IPv4 address 130.216.191.182. Packets with this IPv4 address passing through the university border gateway are thus not only likely to use TCP, but also to use ports 80 or 443. Furthermore, the IPv4 TTL field is constant for all packets originating from the proxy.

The following sections discuss how these goals can be met by considering only certain fields (or parts of fields, i.e., bits or bytes extracted from fields) from an IPv4 stream, dropping the remaining data. Thus, each packet is mapped into a symbol which comprises part of the packet’s original data. In this thesis, this procedure is generally referred to as packet mapping or simply as mapping. $\omega$ is the width in bits of the symbols resulting from a mapping.

Mappings meeting Goal 3 (constant expected entropy estimation error) are called fixed-volume mappings in this thesis. It is also possible to define mappings which do not meet this Goal. This thesis refers to such mappings as variable-volume mappings. Experiment 4.2.1 is an example for a variable-volume mapping: it removes all data except the (variable length) IPv4 component from the packet records and uses this IPv4 remainder then as symbol. For simplicity, this thesis will use the term mapping by itself
to refer to fixed-volume mappings unless explicitly stated otherwise.

For fixed-volume mappings, \( \omega \) is arbitrary but fixed. In practice, one would prefer to have \( \omega \) a multiple of eight to improve processing speed.

Instead of a raw IPv4 data stream (as it was used for the generation of Figure 4.2, for example), the resulting symbol stream is then passed to an information measure.

Avoiding cancellation effects may also increase the SNR. An example of this effect will be presented in Section 6.6.

Using a mere subset of the available fields immediately raises the question as to which fields are candidates to be considered in a mapping. The following section investigates ways to vet such fields.

### 6.2 Field selection

The problem of field selection was already briefly mentioned in Chapter 3, when the concepts of high and low entropy monitoring were introduced: general knowledge about the value distribution of the fields in a network stream may be used to select fields with either:

- a low background entropy and entropy peaks during events (low entropy monitoring), or
- a high background entropy and entropy drops during events (high entropy monitoring).

The following discussion assumes high entropy monitoring. Therefore, mapping field candidates have to meet two criteria:

- During the absence of network events, the values of the field should be uniformly distributed to maximise the entropy. While a uniform value distribution is not a guarantee for high entropy\(^1\), it is very likely to produce higher entropies in scenarios where the data originates from a computer network: as many individual machines contribute to the network traffic, regular arrangements of the values are very improbable.

- During the presence of network events, values should focus on a small subset of the available range to yield low entropies.

\(^1\)Consider uniformly distributed symbol sequences with low entropy, such as \(abecedabecedabecedabeced\ldots\).
The same considerations apply in reverse for low entropy monitoring.

Histograms may be used to compare the value distributions of fields during the presence and absence of events in order to decide whether a field meets the criteria and is thus suitable for inclusion into a mapping. Figure 6.2 shows such a comparative histogram for the payload of TCP and UDP packets in $T_{20050722\,15-17}$. A truncated eight bit checksum (cf. Section 6.3, page 82) of the payload bytes was used for the generation of this graph.

![Figure 6.2: Distribution of the truncated 8 bit sum (cf. Section 6.3, page 82) of TCP and UDP payload during the absence (red, mark: +) and the presence (green, mark: x) of Windows Messenger spam events in trace file $T_{20050722\,15-17}$](image)

The plot exhibits a largely uniform value distribution during the absence of events (red point series, mark: +). The most pronounced outlier is bin 32, which occurs approximately six times as often as other values. During the presence of the Windows Messenger spam events (green point series, mark: x), the situation is clearly more polarised: here, the count for bin 154 is roughly 500 times higher than the count for the other bins. This effect is a result of the constant UDP payload prefix in the Windows Messenger spam packets of the $T_{20050722}$ trace file. The captured part of this payload is the following hexadecimal byte sequence, whose truncated 8 bit sum is 154:

```
04 00 28 00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 f8 91 7b 5a 00 ff d0 11 a9 b2 00 c0 4f b6
```

The histogram suggests that we may expect a low entropy during the Windows Messenger spam events and a relatively high entropy during the absence of these events. According to the criteria for suitable fields mentioned before, this field can be considered a possible candidate for use in a mapping intended for high entropy monitoring. The following section uses this field for a simple mapping. Subsequently, Section 6.4 examines properties of fields from some common protocols that are available for the use in mappings.
6.3 Simple mapping

In the simplest case, a mapping may only consider a single observable from a packet stream. Such a mapping will be referred to as simple mapping from hereon. Both fixed-volume mappings and variable-volume mappings are conceivable as simple mappings. However, to meet mapping Goal 3 (constant expected entropy estimation error), one might prefer simple mappings to produce fixed size symbols.

If the size of a field used in a simple mapping is already fixed, one may simply use the field’s values as raw symbols. Examples are the IPv4 protocol or the IPv4 length fields. In some cases, it may be sensible to consider only parts of the data in a field. One possible motivation for this is the value range of the field exceeding the range of the actual values by one or more orders of magnitude. This is often the case for the IPv4 length field, for instance, because its value range is usually limited to the frame size of the underlying data link layer protocol.

One may compute a fixed size checksum from variable volume fields, such as payload or the entire TCP headers. For instance, consider a function \( f(i, n) \) which only returns the lowest \( n \) bits of an integer \( i \) and a vector of \( k \) bytes \( V \). \( x = f(\sum_{j=1}^{k} V_j, n) \) is an \( n \)-bit checksum of \( V \). This type of checksum will be referred to as truncated \( n \) bit sum for the remainder of this thesis. Alternatively, one could successively XOR fixed size bit sequences from \( V \) to generate a checksum. These checksums may then be used as symbols. The following bullet list roughly describes some properties of the entropy of truncated \( n \) bit sums:

- If the entropy of all participating fields is low, the entropy of the truncated \( n \) bit sum is at most as high as that of the sum of entropies of the participating fields (triangular inequality).
- Let \( F \) be the field with the highest entropy among the participating fields in a truncated \( n \) bit sum. The entropy of the truncated \( n \) bit sum at most as high as the entropy of \( F \).
- Correlation between the participating fields lowers the entropy of the truncated \( n \) bit sum, depending on the degree of correlation. For instance, consider two participating fields \( F_1 \) and \( F_2 \), which are highly correlated: \( F_1 = c_1 + x \) and \( F_2 = c_2 - x \) with constants \( c_1 \) and \( c_2 \). In this scenario, entropy of the truncated \( n \) bit sum of \( F_1 \) and \( F_2 \) is minimal, because \( F_1 + F_2 = c_1 + c_2 \) for all \( x \).

The following experiment uses a simple mapping of the payload field. As the payload is a variable volume field, the mapping uses the checksum procedure above. It utilises only the lowest eight bits of the payload byte sum. The resulting symbol sequence is then used as input for the entropy measure.
Mapping-notation in this thesis
During the project work for this thesis, mappings were numbered sequentially in the order of their implementation. This way, each mapping used in this thesis can simply be referred to by its identification number or Mapping Id for short: for instance, $M_5$ refers to the mapping with Mapping Id 5, the mapping used in Chapters 4 and 5. The mappings used in this thesis are listed by their mapping Id in Appendix (Chapter A.3).

Experiment 6.3.1 (Simple mapping)

Description: This experiment applies a high entropy monitoring strategy to $T_{20050722,15--17}$ by mapping TCP and UDP packet records into the truncated 8 bit sum of the TCP or UDP payloads. The sample size is $N_S = 2000$ as in Chapter 4. The T-entropy of the resulting traffic samples is computed and presented in a graph.

As already discussed in previous experiments, $T_{20050722}$ contains a number of Windows Messenger spam events. The experiment uses the last two hours of $T_{20050722}, T_{20050722,15--17}$, because it does not cover the SYN flood event discussed in Chapter 4; $T_{20050722,15--17}$ therefore only contains Windows Messenger spam as obvious events, permitting statements about the (Windows Messenger spam) SNR to be made.

Expectation: The payload field meets the criteria for high entropy monitoring (cf. page 100) and the Windows Messenger spam uses a fixed payload prefix as described on page 81. Hence, the mapping should preserve the effect of the Windows Messenger spam events on the T-entropy, known from the experiments in Chapter 4.

Observation: Figure 6.3 still exhibits the steep drop pattern in the T-entropy that is already known from the experiments undertaken in Chapter 4.

![Figure 6.3: $T_{20050722,15--17}$: a mapping based on the truncated 8 bit sum of TCP and UDP payload. The effect of the Windows Messenger spam events discussed in Chapter 4 is still clearly visible as a sequence of steep entropy drops.](image-url)
The expectation is thus confirmed. However, the plot permits a number of additional observations:

1. The background entropy is approximately one bit higher than it was in the experiments with raw IPv4 data.

2. Although the volume of each packet record is considerably reduced as it is mapped into a single byte symbol, the SNR still reaches a value of 16.67 for $b = 2.8$.

3. The residual noise is higher than it was with raw IPv4 data ($\sigma(E') = 0.1338$ bits/byte). A reduction of this noise would result in a higher SNR.

Fixed volume simple mappings meet the goals defined in the motivation to this chapter:

1. The amount of data passed to the entropy measure is at the very least reduced by a factor of $20/\omega$. This factor is based on the smallest possible IPv4 packet which is 20 bytes. As most IPv4 packets carry more data than just the IPv4 header, this factor is rather conservative.

2. Cancellation effects with other fields cannot occur as this mapping only considers data from a single field.

3. The symbol size $\omega$ is fixed to a single byte, therefore the traffic samples are also of constant size when considering a fixed number of packets.

4. Correlation effects with other fields are not possible for the same reason as already mentioned under Point 2.

The following section first examines the characteristics of other fields that may be considered in mappings. Subsequently, Section 6.6 will discuss a more complex mapping, which considers a combination of several observables.

The mapping presented above causes a degradation of the SNR compared to Example 5.2.1. Section 6.6 demonstrates that the SNR of the Windows Messenger spam events can in fact be increased beyond that of Example 5.2.1 if one chooses the observables carefully.

### 6.4 Fields available for mapping

At the time of writing, 137 different transport protocols [5] exist. Each of them uses its own set of protocol header fields. This section is restricted to the protocols that were most commonly observed at
the border gateway of the University of Auckland during this project. These are Ethernet, IPv4, TCP, and UDP. This section provides brief field descriptions for the header fields of these protocols and some additional fields contributed by the PCAP packet header (in the same order as they appear in a libpcap packet record). For certain fields, source code snippets from Linux 2.6 [3] are presented to explain how this (and probably other) operating systems respond to certain values. Also, the discussions of some fields include experiments with simple mappings for that field for demonstration purposes.

Note that statements about the distribution of the individual fields only apply to the border gateway of the UoA. Other networks may exhibit different distributions.

The entropy properties of common protocol fields have also been discussed in Speidel, Eimann, and Brownlee [93, 94].

6.4.1 PCAP packet header fields

Apart from a timestamp field, the PCAP packet header only contains two fields, concerning the packet’s actual length and the length of the captured portion. The latter does not provide useful information for network event detection and the actual length is redundant information in the context of this thesis, as only IPv4 packets are considered here and IPv4 itself provides this length information. For this reason, only the timestamp field is considered here.

**Timestamp (32+32 bits)**

As mentioned before, the timestamp in the PCAP packet header may be used to compute the interarrivial time $t_{int}$ of packets. $t_{int}$ may be used as an indicator for the presence of network events. For example, a sudden SYN flood or ICMP flood causes the interarrival time of packets in the target network to decrease. In order to narrow down the domain of possible values, one may choose intervals that are mapped into symbols. A simple example for an interval mapping is provided below:

This mapping was chosen with $T_{20050722}$ in mind: the longest interarrival time in this trace file is approximately 850 ms, so it makes no sense to consider separate symbols for additional intervals beyond 1 s. For version 2.4 of its file format, libpcap uses the `timeval` C-structure with a resolution of 1 µs, so there is no point in defining intervals below this value. More sophisticated mappings could provide higher interval/symbol resolutions. The average packet rate of the link under observation may guide the design of more refined interval mappings: for example, one might choose to map interarrival times within
a certain neighbourhood of the average interarrival time into a single symbol and have a higher symbol resolution beyond this range.

During high rate events, one would expect to observe more symbols for short intervals. To validate this conjecture, the coarse interval mapping just described was applied with $T_{20050722}$. Figure 6.4 shows the result.

![Figure 6.4: Entropy of $T_{20050722}$ with a coarse interarrival time interval mapping. Both the SYN flood attack and the Windows Messenger spam events lead to sequences of identical symbols which lower the entropy, due to their relatively high packet rates. An example for such a sequence is presented in Table 6.2. Another feature in this plot is the relatively sharply bounded background entropy. Note that libpcap uses software timestamps that may be inaccurate, due to interrupt queues, for instance. If this plot would have been generated from data recorded with a DAG card (cf. Section 4.1 and [2, 8]), the bounds of the background entropy might have been less clear due to the more precise (hardware) timestamps these cards provide.](image)

During the SYN flood attack and the Windows Messenger spam events, the packet rate rises, i.e., the average interarrival time drops. This may result in sequences of identical symbols, leading to entropy drops. An example of such a sequence is presented in Table 6.2.

Due to the rather coarse symbol resolution of the interval mapping presented in Table 6.1, the background entropy of the resulting traffic samples is rather low at just over 1 bits/byte. The background entropy is generally limited to a (relatively) smaller range than was observed with other mappings (cf. Figure 4.2, 4.4, 6.3, and 6.13, for example). For $T_{20050722}$, the mapping clearly identifies the two malicious event

<table>
<thead>
<tr>
<th>Interarrival time $t_{int}$</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{int} \leq 10 \mu s$</td>
<td>'a'</td>
</tr>
<tr>
<td>$10 \mu s &lt; t_{int} \leq 100 \mu s$</td>
<td>'b'</td>
</tr>
<tr>
<td>$100 \mu s &lt; t_{int} \leq 1 ms$</td>
<td>'c'</td>
</tr>
<tr>
<td>$1 ms &lt; t_{int} \leq 10 ms$</td>
<td>'d'</td>
</tr>
<tr>
<td>$10 ms &lt; t_{int} \leq 100 ms$</td>
<td>'e'</td>
</tr>
<tr>
<td>$100 ms &lt; t_{int}$</td>
<td>'f'</td>
</tr>
</tbody>
</table>

Table 6.1: An interval mapping for packet interarrival times. As the time resolution of the libpcap v2.4 trace files is $1 \mu s$, intervals below this bound would not yield additional information. Likewise, intervals beyond $1 s$ would not yield additional information, because no interarrival times beyond $1 s$ were observed in $T_{20050722}$. 
6.4 Fields available for mapping

<table>
<thead>
<tr>
<th>Timestamp [s:µs]</th>
<th>$t_{int}$ [µs]</th>
<th>Symbol</th>
<th>Src socket</th>
<th>Dest socket</th>
<th>SYN flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>1121999956:817533</td>
<td>40</td>
<td>'b'</td>
<td>130.216.221.82:58155</td>
<td>81.5.176.160:5406</td>
<td>True</td>
</tr>
<tr>
<td>1121999956:817607</td>
<td>64</td>
<td>'b'</td>
<td>130.216.221.82:58155</td>
<td>81.5.176.160:5406</td>
<td>True</td>
</tr>
<tr>
<td>1121999956:817649</td>
<td>42</td>
<td>'b'</td>
<td>130.216.221.82:58155</td>
<td>81.5.176.160:5406</td>
<td>True</td>
</tr>
<tr>
<td>1121999956:817713</td>
<td>64</td>
<td>'b'</td>
<td>130.216.221.82:58155</td>
<td>81.5.176.160:5406</td>
<td>True</td>
</tr>
<tr>
<td>1121999956:817757</td>
<td>44</td>
<td>'b'</td>
<td>130.216.221.82:58155</td>
<td>81.5.176.160:5406</td>
<td>True</td>
</tr>
<tr>
<td>1121999956:817799</td>
<td>42</td>
<td>'b'</td>
<td>130.216.221.82:58155</td>
<td>81.5.176.160:5406</td>
<td>True</td>
</tr>
<tr>
<td>1121999956:817870</td>
<td>71</td>
<td>'b'</td>
<td>130.216.221.82:58155</td>
<td>81.5.176.160:5406</td>
<td>True</td>
</tr>
<tr>
<td>1121999956:817915</td>
<td>45</td>
<td>'b'</td>
<td>130.216.221.82:58155</td>
<td>81.5.176.160:5406</td>
<td>True</td>
</tr>
<tr>
<td>1121999956:817956</td>
<td>41</td>
<td>'b'</td>
<td>130.216.221.82:58155</td>
<td>81.5.176.160:5406</td>
<td>True</td>
</tr>
<tr>
<td>1121999956:818028</td>
<td>72</td>
<td>'b'</td>
<td>130.216.221.82:58155</td>
<td>81.5.176.160:5406</td>
<td>True</td>
</tr>
<tr>
<td>1121999956:818071</td>
<td>43</td>
<td>'b'</td>
<td>130.216.221.82:58155</td>
<td>81.5.176.160:5406</td>
<td>True</td>
</tr>
<tr>
<td>1121999956:818112</td>
<td>41</td>
<td>'b'</td>
<td>130.216.221.82:58155</td>
<td>81.5.176.160:5406</td>
<td>True</td>
</tr>
</tbody>
</table>

Table 6.2: Properties of 12 consecutive packets, taken from the SYN flood event in $T_{20050722}$: the interarrival time for all packets is greater than 10 µs and less than 100 µs. Consequently, application of the interval mapping presented in Table 6.1 results in a sequence of 12 consecutive 'b' symbols.

types in the trace file, i.e., Windows Messenger spam and the SYN flood.

6.4.2 Ethernet header fields

Source and destination MAC address (48 bits each)

As already mentioned in Section 4.1, these fields are usually not of great value, because the information they contribute is largely redundant if IPv4 frames (or any other frames of a network layer protocol that provides its own addressing scheme) are considered. However, if these fields are made available in otherwise anonymised trace files, they may at least be useful to determine the inward or outward direction of packets. The maximum size of an Ethernet is 1024 nodes (cf. Peterson and Davie [83] and Shay [90]). To avoid congestion, Ethernets usually do not grow to this size. For this reason, the entropy of these fields should generally be quite low. For border gateway Ethernet connections, one would expect to observe only a handful of different Ethernet MAC addresses, therefore the entropy of these fields should be particularly low.

Ether type (16 bits)

Similar to the protocol field of IPv4, the ether type field in the Ethernet header describes the higher level protocol type transported in an Ethernet frame. At the time of writing, its value is most often 0x0800, indicating that IPv4 data is being transported. Version 6 of the Internet Protocol (0x86dd) was not widely used at the time of writing, but is likely to be adopted in the future as the availability of IPv4 addresses decreases. Apart from this, ARP frames occur occasionally with a value of 0x0806 in the frame
Packet Mapping

88 Packet Mapping

type field. Many other frame types have been reserved for legacy protocols such as DECNet (0x6003) or AppleTalk (0x809b). These protocols are no longer widely used and were not observed in the trace files captured for this thesis.

For this thesis, the ether type field is not of interest, because only IPv4 frames are considered. However, it may be of value in some cases: in order to collect large amounts of insecure WEP initialisation vectors, wireless encryption key cracking tools such as aireplay-ng send out several ten thousand ARP requests in a relatively short time. Such behaviour is quite unusual during normal network operation, for two reasons:

1. Most operating system kernels cache ARP responses. Hence, ARP requests are are only necessary if the cache needs to be refreshed.

2. In the case of Ethernet, the number of ARP cache entries is typically in the range of dozens. For this reason a host only needs this number of ARP requests to initialise its ARP cache.

Under normal circumstances one would expect the entropy of the ether type field to be low, due to the many IPv4 frames and occasional ARP frames. A flood of ARP requests is thus likely to raise the entropy of this field as the mix of values becomes less polarised. At the time of writing, this field would therefore most likely be considered for low entropy monitoring.

For the same reason one would expect the entropy of this field to rise with the adoption of IPv6. If the occurrence of IPv4 and IPv6 frames should reach similar levels in the future, this field could also be useful for high entropy monitoring, because an increased amount of either frame type would then cause the entropy to drop.

6.4.3 IPv4 header fields

Version (4 bits)

As the name suggests, this fields value is 4 for all IPv4 packets. It is thus not suitable for entropy measurement, because no information is contributed by this field. Operating systems are likely to simply drop all packets carrying forged values in this field, so it is difficult to see how it could be abused for malicious network events. For Linux 2.6 [3], the corresponding code can be found in the file ip_input.c:
6.4 Fields available for mapping

Source 6.4.1 (Excerpt from Linux 2.6 source file ip_input.c)

```c
if (iph->ihl < 5 || iph->version != 4)
    goto inhdr_error;
[...]
inhdr_error:
    IP_INC_STATS_BH(IPSTATS_MIB_INHDRERRORS);
drop:
    kfree_skb(skb);
```

Note that this field may be of interest if both IPv4 and IPv6 packets appear on the monitored link.

**Header length (4 bits)**

This field’s value is 5 in most cases, indicating IPv4’s default header length of five 32 bit words. Only if additional option fields in the header are required for a packet, the value may be larger than 5. This appears to be rarely the case: some trace files examined in the course of this project comprise 75 million packet records and at least for some of these files not a single record has a header length different from 5. The information contributed by this field is thus negligible, unless one wants to monitor for exploits of this field.

However, manipulation of this field appears not sensible, even for malicious events such as attacks, because the target host’s operating system would miss the correct entry point for processing the transport protocol header. Furthermore, the code snippet in the discussion of the version field suggests that at least for Linux 2.6 packets with header length values smaller than five are dropped.

**Differentiated services and congestion notification (8 bits)**

The purpose of this field has changed over the time. RFC3168 [85] describes its current purposes for packet prioritisation and congestion notification. For most packets this field has a value of 0, only about 10% of the packet records of the trace files examined in this thesis have bits set in this field. For this reason one may consider it for low entropy monitoring and the detection of events which use this field – most likely DoS attacks.

**Length (16 bits)**

This field contains the overall length of an IPv4 packet in bytes. Figure 6.5 shows a log histogram of IPv4 lengths from \( T_{20050902} \). This trace file was chosen because it does not contain large scale events,
apart from Windows Messenger spam, which may distort the histogram.

![Figure 6.5: Distribution of the IPv4 lengths in trace file $T_{20050902}$. The bins for 40 bytes and 1500 bytes have the highest counts. Other packet lengths are distributed within three orders of magnitude; here, smaller packets generally appear more often than larger packets, possibly an example of the Newcomb-Benford [79, 18] law. Outliers in this band may be the result of certain events types: for example, Windows Messenger spam causes elevated counts for bins 508 (322707 additional packets), 465 (176237 additional packets), 349 (101531 additional packets), and 346 (39826 additional packets).](image)

Although the field spans 16 bits, the maximum transfer unit (MTU) value on the hardware used often restricts its value. The University of Auckland uses Ethernet as network hardware which limits the body size to 1500 bytes for its frames (see Peterson and Davie [83] and Shay [90]).

The most noticeable peaks are found for lengths of 40 bytes (used for SYNs and ACKs) and 1500 bytes (maximum packet length, appearing for messages that are either more than or exactly 1500 bytes long). These two peaks could be one plausible explanation for the regular peak pattern in Figure 4.7. The remaining packet lengths are distributed within three orders of magnitude. Generally, shorter packets occur more often than longer packets. A number of individual outliers, which may arise from frequently used services is also present.

High rate events with constant packet length swamp this field’s distribution with their characteristic packet lengths, lowering the entropy. Hence, the IPv4 length field is a candidate for high entropy monitoring. However, with the distinctive peaks at lengths of 40 bytes and 1500 bytes, this field’s background entropy is already reasonably low.

Figure 6.6 demonstrates that even a simple mapping into four different length categories is sufficient to visualise the impact of the Windows Messenger spam and SYN flood events on this field’s entropy.
6.4 Fields available for mapping

![Graph showing average T-entropy over time](image)

Figure 6.6: A coarse interval mapping of the IPv4 length field for trace file T20050722: the mapping assigns different symbols for length < 40 bytes, length = 40 bytes, 40 bytes < length < 1500 bytes, and length ≥ 1500 bytes. Although mappings providing higher symbol resolutions are conceivable, the mapping chosen here suffices to visualise the SYN flood and Windows Messenger spam events known from this trace file.

**Ident (16 bits)**

Generally, the ident field is uniformly distributed with a large peak at value 0 being an exception. This peak is the result of a router/operating system policy to set the ident field to 0 when the *Don't Fragment* flag (DF, cf. next section) is set. As a direct consequence of DF=1, the *More Fragments* flag (MF, see next section) and the offset field (see below) also contain zeros. Not all routers appear to implement this policy, however, as the DF flag appears to be set independently from the ident value. Nevertheless, the following (one-way) correlation can be recorded: if the ident field is 0, it is likely that one will find the 16 bit value 0x4000 in the flags/offset fields. Apart from this, some operating systems set both ident and DF to zero in responses to initial TCP SYN packets. Among Linux 2.4 and 2.6, Windows XP, MacOS X 10.4 and Solaris 8, this behaviour was only observed with Solaris 8. In the same situation, Windows and MacOSX set DF=1 and fill ident with increasing values. Both Linux 2.4 and 2.6 respond with DF=1 and ident=0.

Once a connection is established between two hosts, each host increments its packet’s ident fields by a fixed value, typically 1, with each packet sent. Due to the distinctive peak at value 0 this field would most likely be considered for low entropy monitoring. However, predictions as to whether an event will lower or raise the entropy are not easy for this field: if an event contributes packets with ident value zero, one might expect the entropy to drop even further. Other events may contribute variable ident values which would reduce the relative frequency of zeros and thus increase the field’s entropy. As a third possibility, an event may contribute fixed, nonzero ident values. This has two directly opposed effects: on the one hand, the entropy may drop, because the contributed value appears more often. On the other hand, the entropy may rise for the same reason as in the variable value contribution: relatively speaking, there
would be fewer zeros in the pool of ident values.

**Flags (3 bits)**

Three flags are used in the IPv4 header: the Reserved Flag (RF, always zero), the Don’t Fragment flag (DF), and the More Fragments flag (MF). As RF is constant and fragmentation is a relatively rare phenomenon, the DF flag carries most information in this field. Trace file analysis suggests it is set in approximately 85% of the packets, resulting in a relatively low background entropy. Low-volume events not setting this flag could thus be detected with low entropy monitoring, and high-volume events setting the flag with high entropy monitoring.

**Offset (13 bits)**

For fragmented packets, this field contains the fragment’s offset in the original packet. Due to the low frequency of fragmented packets the field’s value is zero for most packets. If events associated with increased fragmentation are considered this field may used in low entropy monitoring, otherwise it is of little interest.

In the past, this field has been used in DoS attacks which attempted to compromise vulnerable TCP/IP stacks with incomplete fragment sequences. These vulnerabilities have been known for a long time and have been fixed in most modern TCP/IP protocol stacks. One may therefore regard this as a problem of the past.

**Time to live (8 bits)**

The TTL field is used to limit the lifetime of packets in the network. Its initial value depends on the sender host’s operating system and may also vary for transport protocols; typical initial values are 30 (AIX - UDP), 32 (Windows 3.11/9x - TCP and UDP), 60 (AIX - TCP), 64 (recent Linux/BSD versions - TCP and UDP), 128 (recent Windows versions - TCP and UDP), and 255 (ICMP packets from most operating systems; Sun OS 5.7 - TCP).

Figure 6.7 shows a typical histogram of TTL values observed at the UoA border gateway. The green graph suggests most outbound connections to originate from hosts which are 2 or 3 hops away from the border gateway. Corresponding maxima can be found at values 62 and 125. A large proportion of the traffic may originate from students. According to university network policy, students and most staff use
Inbound packets are typically (hopwise) travelling longer distances: here, maxima can be found at values 50 and 117, i.e., 14 respectively 11 hops away from the sender hosts. Also, inbound counts are generally distributed across a larger range of values, as – from a border gateway perspective – the outside network is much larger than the inside network.

Given the higher inbound counts for values from 64 downwards compared to those from 128 downwards, the graph also suggests that, in total, more incoming packets originate from Linux/BSD systems than from variants of Windows. However, given the inbound count maxima, Windows hosts generally appear to be closer to the university’s border gateway. There are two possible explanations for this:

- Linux and Unix derivatives in general are not widely used in New Zealand. Many students using university web services from outside the university network use Windows (i.e., client outside, server inside the university network, TTLs originate from request packets).

- In New Zealand, Windows is widely used for servers outside the university network (client inside, server outside the university network, TTLs originate from response-packets).

The value distribution, especially for inbound packets, makes this field an interesting candidate for mapping. As the counts beyond 128 are small, one may consider dropping the highest bit ($2^7$). The second-highest bit ($2^6$) may be regarded as a flag indicating whether a packet originates from Windows or Linux/BSD. Thus, it may act as an indicator as to which of the two “humps” in Figure 6.7 it may be attributed. An event originating from a single host or platform may disturb the count ratios between...
both humps. Events such as attacks often rely on security holes present on a specific platform, making the 2^6-bit particularly interesting for event detection.

**Protocol (8 bits)**

As already mentioned, 137 transport protocols are defined in conjunction with IPv4 at the time of writing. The operating system on the receiver host can decide how to process the payload of an IPv4 packet by examining IPv4’s protocol field. In the trace files used for this thesis, the IPv4 protocol field is clearly dominated by the value 6, TCP’s protocol number. Table 6.3 shows counts of protocol numbers found in $T_{20050902}$, which contains 67742682 IPv4 packet records:

<table>
<thead>
<tr>
<th>Protocol Number</th>
<th>Protocol</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>HOPOPT</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>ICMP</td>
<td>295049</td>
</tr>
<tr>
<td>6</td>
<td>TCP</td>
<td>61855124</td>
</tr>
<tr>
<td>17</td>
<td>UDP</td>
<td>5564724</td>
</tr>
<tr>
<td>41</td>
<td>IPv6</td>
<td>38</td>
</tr>
<tr>
<td>46</td>
<td>RSVP</td>
<td>9</td>
</tr>
<tr>
<td>50</td>
<td>ESP</td>
<td>26998</td>
</tr>
<tr>
<td>103</td>
<td>PIM</td>
<td>738</td>
</tr>
<tr>
<td>255</td>
<td>Reserved</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6.3: Protocol number counts in $T_{20050902}$. Protocol numbers not mentioned have a count of zero. Note that the count of UDP packets is higher than usual, due to a number of Windows Messenger spam events present in this trace file.

As long as no other transport protocols gain significant popularity, one may drop four of the field’s bits: $2^7$, $2^6$, $2^5$, and $2^3$, with the remaining bits still being sufficient to distinguish TCP, UDP, ICMP, and ESP.

Due to TCP’s dominance among transport protocols, this field generally has a low entropy and is mostly of interest for low entropy monitoring. This field’s entropy would rise as soon as other protocol numbers appear more often. For example, this happened during the SQL Slammer worm outbreak, which will be discussed in Section 8.4. Exploits for security leaks in badly maintained legacy transport protocols may have the same effect. Relatively new transport protocols such as SCTP may also suffer from yet undiscovered security leaks.

The following abbreviated source code snippet is taken from the Linux 2.6 kernel. It suggests that filling the protocol field with numbers of unsupported protocols is not a viable strategy for a network attack, because the respective packets would simply be dropped. It is likely that other operating systems adopt a similar policy. Hence, one may generally assume that field will probably carry valid data.
6.4 Fields available for mapping

Source 6.4.2 (Excerpt from Linux 2.6 source file ip_input.c [3])

```c
hash = protocol & (MAX_INET_PROTOS - 1);
if ((ipprot = rcu_dereference(inet_protos[hash])) != NULL) {
    ret = ipprot->handler(skb);
} else {
    kfree_skb(skb);
}
```

Header checksum (16 bits)

Generally, the IPv4 header checksum field has a fairly uniform distribution, i.e., its background entropy is relatively high. There are, however, situations in which the value of this field stabilises: as discussed on page 91, some operating systems’ responses to initial TCP SYN packets contain fixed ident values of 0. Thus, SYN flood attacks run against such hosts from a single source (or single forged IPv4 source address used by many sources) may result in responses with fixed checksum values. During this thesis project, such behaviour was observed in responses from a site running Solaris 8 while receiving a low rate SYN flood attack from inside UoA’s network. Unfortunately, the attack stretches across the entire trace file, which makes statements about its influence on the traffic entropy difficult.

Source & destination address (32 bits each)

At the time of writing, TCP is the predominant transport protocol on the Internet. TCP is connection-oriented, causing destination addresses in request packets to appear as source addresses in the responses. Hence, the content of either field is a mixture of internal and external addresses. Under normal conditions, one may therefore expect to see a very similar background entropy for both address fields.

If a disturbance in this mixture is observed, it may indicate the presence of an event. Examples are:

- SYN flood attacks: Here, the address of the victim host will appear more frequently than other addresses. Depending on the victim’s ability to respond to the request, both address fields may be affected and their entropy should drop. If the victim host is unable to respond to the flood of connection requests, only the destination address field should be affected by a lower entropy, independent of the in- or outbound attack direction.
• Port scans/address scans: For port scans on a single IPv4 address, the entropy of the address fields should drop, due to the frequent appearance of the victim host’s address in both fields. The entropy drop of the source address field should be smaller or equal to the entropy drop of the destination address field, depending on ICMP error messages being sent for probed ports that are unused. For address scans, the entropy of the address fields should rise, because more addresses are probed than are actually in use. The conclusions reached with regard to the source address field in port scans apply here as well.

Apart from the address mixture, the ratio of in- and outbound traffic may provide clues about the presence of a network event. Most organisational networks occupy an address block with a fixed address prefix. For instance, the UoA prefix is 130.216. Such prefixes should appear frequently on typical organisation or business border gateway links. The informational value of this prefix is merely one bit, indicating whether a packet is inbound or outbound. This bit may be derived from the position of the prefix in the source or destination IPv4 address field.

The ratio of in- and outbound traffic can be found by counting the appearances of the prefix in the source and destination address fields over a period of time. Examples for events that may affect this ratio are:

• DDoS attacks: These attacks usually designed to overwhelm target hosts with traffic, i.e., the targets are supposed to be unable to respond to all requests. For inbound attacks, one may thus find the prefix more often in the destination address than in the source address. For outbound attacks, the reverse would be the case. Due to the the frequent appearance of the address prefix, the associated address field entropies should drop.

• Windows Messenger spam: The Windows Messenger spam events present in the trace files collected during this project appear not to provoke additional ICMP traffic about unused IPv4 addresses or hosts not offering the Windows Messaging service. The UoA address prefix thus occurs more often in the destination address field only, particularly because the direction of the events appears to be inbound only. This should lower the entropy for the first two octets and increase the entropy of the last two octets of the destination address, the latter because of the systematic address scanning often seen with Windows Messenger spam (cf. Table 8.2, for instance).

This finishes the discussion of the IPv4 header fields. The following section examines the entropy characteristics of TCP header fields.
6.4 Fields available for mapping

6.4.4 TCP header fields

Source & destination port (16 bits each)

On the client side of a TCP connection, the operating system usually chooses a random source port number for the initial connection request. This port number is then used for the duration of the connection. An exception to this rule can be attack events such as the SYN flood event in Figure 4.2: here, the same source port is “recycled” for each “connection attempt”.

On the server side, responses are sent using fixed source port numbers assigned to specific applications. Examples for default port assignments are port 80 for HTTP and port 22 for SSH.

Events which open unusual ports increase the entropy of the port fields whereas flood events with recycled port numbers or heavily loaded server ports are likely to lower the entropy of these fields. For this reason the port fields may be considered for both high and low entropy monitoring.

Sequence & acknowledgement number (32 bits each)

TCP’s sequence and acknowledgement numbers are used for flow control and to assemble data segments in the correct order at the receiver side. As TCP is a byte-oriented protocol, sequence numbers between two consecutive packets belonging to the same flow usually differ by the payload size of the packets. Initial sequence numbers for each flow are generally randomised and, in the absence of events, one would thus expect a relatively uniform value distribution for this field, i.e., a high entropy.

In an event causing congestion with packets being dropped, retries may lower the entropy of this field. This also applies to flooding attacks such as SYN floods, whose packets may use identical sequence and acknowledgement numbers. Of course such attacks may easily be concealed (with respect to these fields) by randomising the values in the attack packets.

Header length (4 bits)

In the trace files collected for this thesis, the values observed in this field are mostly 5 (minimum) or 8. The value 6 was also frequently observed and appears to be correlated with SYN ACK, whereas the values 7 and 10 occurred alongside SYN and SYN ACK. Values of 11 and 13 were also encountered a lot, while values of 9 and 12 were seen less frequently. The value 14 was also seen in a small number of packets and then only in conjunction with outgoing SMTP. Since it is not entirely clear what causes the
differences, this field was not considered useful for entropy-based network event detection in this thesis. However, it may yet contain information where there are connections between specific events and the value of this field.

**Flags (6 * 1 bit)**

The TCP protocol uses six flags (URG, ACK, PUSH, RESET, SYN and FIN). These flags are an interesting field for entropy measurement because of their widespread use in attacks, especially in SYN floods. The following list examines the TCP flag properties one-by-one:

- **URG**: This flag indicates the use of the *Urgent Pointer* field described below. Typical applications using this flag are *telnet* and *rlogin* which employ it to give a higher precedence to the Unix CRTL-c command interruption, for example. For security reasons, these applications have largely been replaced by *ssh*, so that the use of both applications is rather uncommon nowadays. Hence, this flag is practically unused and therefore probably not a good choice for entropy measurement of any kind.

- **ACK**: The ACK flag is set in most packets as one would expect: if possible, TCP acknowledges the reception of a packet with the transmission of a subsequent packet. In some situations the ACK flag may be set less often than in regular TCP traffic, e.g., due to the impact of a SYN flood attack. For this reason, the ACK flag is a good candidate for low entropy monitoring.

- **PUSH**: In the trace files used for this thesis, the PUSH flag was also observed on a regular basis, e.g., along with web and IMAP traffic. About 20% of the packets in the trace files have this flag set, making it interesting for both, high and low entropy monitoring.

- **RESET**: TCP uses the RESET flag to recover from protocol errors and is occasionally seen, making it a reasonable candidate for low entropy monitoring as events such as address spoofing or misconfigurations may raise the observed number of RESETs.

- **SYN**: The SYN flag is only set at the beginning of a TCP connection, i.e., not very often, making it a typical candidate for low entropy monitoring. Events such as SYN flood attacks are likely to increase the entropy of this field. However, depending on the amount of attack traffic in the regular traffic, such an attack may also lower the entropy: this may happen if the attack reaches a state where almost all observed packets belong to the SYN flood attack. Section 8.1 uses this flag for SYN flood attack detection in $T_{20050722}$. 
6.4 Fields available for mapping

- TCP’s FIN flag may be regarded as the complementary flag to the SYN flag: it is used to close connections. Under normal circumstances, the FIN flag should thus be as frequent as the SYN flag, making it a low entropy monitoring candidate also. Mappings using this flag may be useful for the detection of events where this flag is set more often than usual: for example, Section 8.6 uses such a mapping for the detection of SSH dictionary attacks.

**Advertised window (16 bits)**

The value of this field largely depends on the operating system and the transmitting host, so it is not an obvious candidate for entropy measurement. However, like the sequence number and acknowledgement number fields, flooding attacks with spoofed packets may lower the entropy of this field. This behaviour makes this field a candidate for high entropy monitoring.

**Checksum (16 bits)**

Usually the checksum field is uniformly distributed. It is computed from the TCP pseudo header fields and the payload. In practice, only completely identical TCP packets (in particular including the sequence number and acknowledgement number fields) have identical checksum values. For this reason, the checksum may be a useful observable in the detection of flood attacks and worms, as such attacks may bypass the TCP stack of host operating systems.

**Urgent pointer (16 bits)**

This field is directly correlated with the URG flag described above. See the discussion of this flag for details. At the time of writing, only legacy applications appear to utilise this field. For this reason, it was not considered for entropy measurement in this thesis.

**Options**

TCP options are used for protocol extensions which are not covered by the standard TCP fields. Hence, the purpose of this field depends on the nature of the protocol extension. In the trace files used for this thesis, the vast majority of packets, including most common event-related packets did not use TCP options. As it is not obvious what the actual value of this field for entropy measurement might be, it was not considered in this thesis.
6.4.5 UDP header fields

The discussion of TCP ports applies to UDP as well. For simplicity, one may choose to consider TCP and UDP ports jointly. Such a mapping will be used in the discussion of the SQL Slammer worm outbreak in Section 8.4.

The length field of UDP correlates with the IPv4 length field, so the information it contributes is redundant. Like the TCP checksum, the UDP checksum is usually uniformly distributed. However, in UDP one may observe sequences of identical packets in normal traffic, which then have identical checksums, of course: consider retransmissions of a dropped UDP packet.

6.4.6 Payload

Due to its variable size, the use of the payload of various transport protocols in mappings appears difficult at first glance. However, it is easily possible to compute a fixed-size checksum from this field as discussed on page 82. Although this constitutes a loss of information, it is still possible to detect constant payload, e.g., as it appeared during the SQL Slammer worm outbreak (see Moore, Paxson, Savage, Shannon, Staniford, and Weaver [31] and Ray [86]). This field was already used for a simple mapping in Section 6.3.

The payload field is a high entropy monitoring candidate, because its content is usually unstable under normal conditions and may stabilise during an event.

6.5 Conditional field selection for mappings

The mappings discussed so far all use observables related to fields statically located within a packet structure. The actual position of the data for observables such as minimum TCP port or last two octets of the internal IPv4 address depends on the context, however. For the latter example, a mapping would check which of the IPv4 addresses matches the prefix of the internal network before deciding whether the value of the source or destination IPv4 address field is mapped. Here, there is no longer a fixed one-to-one relation between an observable and specific field in the network stream.

This kind of observable permits the definition of more general mappings: rather than mapping a specific network stream field, arbitrary rules may be defined as to which packet record fields (or parts thereof) are used for mapping, on a per packet basis. The following paragraphs discuss properties of the internal
IPv4 address observable as an example.

For an internal vs. external view of IPv4 addresses, it is relatively easy to eliminate the constant network prefix of internal IPv4 addresses. Furthermore, such a classification may yield a more focused view of events: if, for instance, a SYN flood attack against an external machine is run on an internal machine, both IPv4 addresses involved appear in both address fields, as each address will appear as source and as destination. An internal vs. external view of the event forces each address into an individual class, hence maximising the event’s effect on the entropy.

For internal UoA IPv4 addresses, the network part occupies 16 bits. Without the 130.216 address prefix, only 16+32 bits are left for mapping purposes. To permit histogram representation, these can be split into three parts: the last two octets of the internal IPv4 address, the first two octets of the external IPv4 address, and the last two octets of the external IPv4 address. As the UoA network comprises approximately 14000 machines, only about a quarter of the available $2^{16}$ host identifiers is in use. Figure 6.8 shows the distribution of the last two internal IPv4 address octets in $T_{20050902}$.

![Figure 6.8: Last two address octets of internal IPv4 addresses in trace file $T_{20050902}$. At the time of writing, only about 14000 out of $2^{16} - 2$ possible addresses are in use. The red band spanning the entire address range with counts between 4 and about 70 is an indicator for port scanning activity. In the case of $T_{20050902}$, this port scanning is related to frequent Windows Messenger spam events.](image)

The histogram in Figure 6.8 shows a base count between 4 and 70 for the last two octets of internal UoA IPv4 addresses. This results from the Windows Messenger spam activity in $T_{20050902}$. Counts higher than 100 are present for 2400 internal IPv4 addresses only. During the Windows Messenger spam events, one would expect the entropy of this field to rise, as the unusual addresses contributed by these events constitute new patterns. Figure 6.9 presents a simple mapping based on the last two octets of the internal IPv4 addresses, which confirms this expectation.

Analysis of external IPv4 address counts suggests that most of the usual Internet activity observed
Figure 6.9: Simple mapping using the internal host identifiers in trace file $T_{{20050722}}$. This trace file does not contain large scale events such as the SYN flood in $T_{{20050722}}$. However, several regular Windows Messenger spam events appear to run concurrently. The address scanning associated with the Windows Messenger spam events causes the entropy of the affected traffic samples to rise. These samples appear as entropy peaks in this plot.

at UoA is limited to approximately 6000 different two byte prefixes of external IPv4 addresses. From this perspective, the first two octets of external IPv4 addresses exhibit a comparable distribution to the one shown in Figure 6.8, with the main difference being the lack of scanning-related base counts. During attacks like SQL Slammer (see Moore, Paxson, Savage, Shannon, Staniford, and Weaver [31] and Ray [86]), which generate random target IPv4 addresses, this field’s entropy is likely to increase. It may therefore be used for low entropy monitoring. However, due to the many different patterns contributed by the approximately 6000 prefixes, the background entropy of this field is already relatively high.

For the last two octets of the external IPv4 address, both values and counts are widely distributed. This distribution can be disturbed by high rate events originating from external machines. Figure 6.11 documents this with two distributions of this field, one recorded during the absence of Windows Messenger spam packets (red point series), the other during the presence of Windows Messenger spam events (green point series) in $T_{{20050722}}$: during the presence of Windows Messenger spam events, this field may exhibit a more polarised distribution, making it a possible candidate for high entropy monitoring.

### 6.6 More complex mappings

For the traffic observed at the University of Auckland, several fields in the raw IPv4 data meet the constraints described in Section 6.2. The present section will combine data from suitable fields in order to extend the simple mapping discussed in Section 6.3. The result is a multi-byte mapping which not only meets the mapping goals listed on page 78 but also increases the SNR.
Ideally, for high entropy monitoring, the fields in a multi-byte mapping should be uncorrelated for normal traffic and show a higher degree of correlation during events. For instance, such fields are the IPv4 length field and the last two octets of the external IPv4 address\(^2\) for Windows Messenger spam events.

Figure 6.10 and 6.11 show histograms of the IPv4 length field and the last two octets of the external IPv4 address with respect to the UoA network. As in Figure 6.2, separate distributions are shown for Windows Messenger spam event regions and non-event regions.

The histograms suggest that, during Windows Messenger spam events, the value distribution in both fields behaves similarly to that of the payload field discussed in Sections 6.3 and 6.4.6. The value distribution

\(^2\)A limitation to the last two octets of the external IPv4 address was chosen to keep the number of bins in the histogram of Figure 6.11 manageable: 65536 instead of 4.2 billion.
polarises during the events, i.e., a smaller set of values becomes more pronounced. With respect to high entropy monitoring, one may therefore consider the associated fields as compatible: cancellation effects should be small if these fields are jointly considered in a mapping.

In order to satisfy Goals 1 (data volume reduction) and 3 (constant expected entropy estimation error) defined in Section 6.1, it would be sufficient simply to concatenate the length field, the last two octets of the external IPv4 address and the truncated 8 bit sum of the payload into 5 byte symbols.

However, it is possible to do better than this in terms of data volume: as described on page 89, the bits $2^{11}$ to $2^{15}$ of the IPv4 length field are zero, because the university uses Ethernet for the data link layer. Hence one may choose not to consider these bits at all. This constitutes a saving of 5 bits.

It is even possible to save an entire byte in each symbol if one also drops the bits $2^0$, $2^1$ and $2^2$ from the length field. As only the bits $2^3$ to $2^{10}$ remain under consideration, bins spanning eight values each are established for the IPv4 length field. The IPv4 length of the Windows Messenger spam packets in $T_{20050722.18-17}$ is 508 bytes. Given that the counts around bin 508 in Figure 6.10 are relatively low during events, the counting error associated with bins spanning eight values may be considered negligible.

Another byte can be saved by considering the truncated 8 bit sum of the last two IPv4 address octets rather than using both octets. These considerations leave us with the three byte symbol structure shown in Figure 6.12.

![Figure 6.12: Symbol structure of a more complex mapping for Windows Messenger spam events; the symbols in this mapping span three bytes.](image)

It must be pointed out that this combination of fields was derived by experimentation and may thus not necessarily be universally applicable. In other words: the particular field combination of the payload, the IPv4 length and the last two octets of the external IPv4 address may not be suitable at sites with other traffic profiles than those observed at UoA. Generally, complex mappings need to be adjusted to the traffic profile at a monitoring site. Also, the applicability of a particular mapping may fade over time as the composition of the traffic changes on a permanent basis.

The following experiment was undertaken with this three byte symbol structure.
Experiment 6.6.1 (Complex mapping)

Description: This experiment is essentially a repetition of Experiment 6.3.1, with the exception that the multi-byte mapping just described was applied instead of the simple payload mapping used before.

Expectation: As described before, the fields considered in this mapping are “compatible” with respect to their entropy in high entropy monitoring and show low/high correlation during absence/presence of Windows Messenger spam events. These properties justify two expectations:

1. The Windows Messenger spam events should still be visible.
2. The noise affecting the background entropy may be reduced, due to the low correlation of the contributing fields during the absence of events. The reason for this is an increased diversity of patterns (compared to Experiment 6.3.1) arising from the two extra fields considered. This may increase the SNR.

Observation:

Figure 6.13 shows a T-entropy plot of the mapped trace file. Compared to Experiment 6.3.1, the entropy level has generally dropped by approximately 0.5 bits/byte. Furthermore, outliers in the background entropy appear to be less frequent and less pronounced than in Experiment 6.3.1: for example, a clearly visible entropy drop in Figure 6.3 in the samples around second 6733 disappears completely with the current mapping.

![Figure 6.13: Multi-byte mapping based on the IPv4 length and the truncated 8 bit sums of the payload and the last two octets of the external IPv4 address. This mapping yields a higher SNR for Windows Messenger spam events (23.96) than the other mappings discussed so far.](image)

With $\sigma(E') = 0.085$ bits/byte, the residual noise is lower than in Experiment 6.3.1, confirming the expectation. This circumstance contributes to a relatively high SNR for this experiment. For $b = 2.7$,
it reaches 23.96, which is even higher than the SNR obtained in Example 5.2.1.

6.7 Conclusions

The present chapter discussed packet mapping as a way to enhance the results retrieved from entropy measurements of network stream data:

- The application of a mapping generally reduces the data volume. This increases the processing speed, especially for slow measures.
- With mapping, entropy cancellation effects may be avoided by choosing appropriate observables from the traffic.
- Fixed-volume mapping leads to entropy estimation errors within a constant range, thus making entropy samples from different traffic samples comparable.
- Applying a mapping may reduce redundancies in the data stream.
- The entropy of mapped traffic samples may also exhibit a higher SNR than that of raw traffic samples.

Due to these characteristics, mappings will generally be preferred for the entropy measurements in the remainder of this thesis.

As already mentioned in the motivation of the present chapter, the following chapter addresses questions concerning the window size.
7

Traffic sampling

7.1 Motivation

The concept of packet mapping discussed in the previous chapter may be seen as a method to “streamline” the traffic under observation. However, mapping does not answer the question as to how entropy changes along a mapped symbol string may be observed. The present chapter discusses three traffic sampling schemes to address this question.

Mapping generates a long symbol string, each symbol of which represents a single packet. One way to observe entropy changes along this string is the use of a sliding window. The advancement $\alpha$ of the sliding window with respect to its size $N_S$ determines the correlation of the data in the resulting traffic samples:

- $\alpha < N_S$: The data of adjacent samples overlaps. For this reason, there is a high degree of correlation. Because of the overlap, the entropy of adjacent samples changes more slowly. Each symbol has an
impact on at least one entropy sample.

• $\alpha \geq N_S$: The data of adjacent samples does not overlap. Correlation between the samples is thus caused exclusively by the traffic and not by the sampling method. If one traffic sample does not contain any event data and the following traffic sample largely consists of event data, a steep edge may result. Each symbol contributes exactly to one entropy sample in the case of equality and at most one entropy sample otherwise.

If the window size $N_S$ is small with respect to the string size, or it advances slowly across the string (i.e., $\alpha < N_S$), many traffic samples may be generated by sweeping the window over the string. The entropy samples derived from the traffic samples may then be used to observe changes. In this thesis this procedure is referred to as traffic sampling.

Two parameters may be used to determine the size of the sliding window: it may either be expressed in terms of data volume (in bytes or in symbols, for instance), or in terms of the time interval covered by the window. One may thus call the associated sampling methods volume-based sampling and interval-based sampling.

As each symbol of a traffic sample represents a single packet on the wire, it makes sense to say that a packet belongs to a traffic sample, although this is not quite correct: what actually belongs to the traffic sample is the symbol into which the packet was mapped. However, for reasons of simplicity, this terminology may be used hereafter. Likewise, it also makes sense to say that a packet belongs to an entropy sample, which is not quite correct for the same reason.

The following sections discuss the volume-based sampling and interval-based sampling schemes and present the respective advantages and disadvantages. Neither method is perfect for the trace file analysis undertaken in this thesis. A third sampling scheme that combines volume- and interval-based sampling will be discussed subsequently. This sampling scheme combines some of the advantages of the other two methods. The last section of this chapter examines sensible choices for the window size $N_S$.

### 7.2 Volume-based sampling

From Chapter 6 we already know that it is desirable to consider traffic samples with a fixed data volume during entropy monitoring. In the simplest case, this can be achieved by breaking the mapped packet string after a fixed number of symbols, if the symbols are also fixed in size. Figure 7.1 shows an example for volume-based sampling with a sliding window of $N_S = 5$ symbols: each bar on the time axis depicts
7.2 Volume-based sampling

a packet arrival. The traffic samples to which these packets belong are indicated by the two series of sequential traffic sample numbers above and below the time axis: the top row of numbers indicates the starting points of traffic samples, the bottom row of numbers indicates where the respective traffic samples end.

![Figure 7.1: Volume-based sampling: Packet arrivals (blue) on a time axis with a volume-based sampling scheme of five packets per traffic sample. Numbers above the axis indicate the beginning of the traffic sample with that number; numbers below the axis state the end of the respective traffic sample.](image)

At each sampling step, the sliding window advances by its own size, i.e. $N_S = \alpha$, thus not producing overlaps or gaps.

Volume-based sampling is not ideal for trace file analysis, this can be observed in Figure 7.1: Traffic sample start and end times (and thus traffic sample durations) are irregularly distributed across the time axis. This does not come as a surprise, because

- the utilisation of the monitored link may change with time, and
- the simple packet counting technique applied in volume-based sampling does not preserve information about packet arrival times.

Simply plotting the entropy samples against their sample numbers permits only limited statements about the start and end times of the associated traffic samples (timing jitter). This is caused by the fluctuating packet rate on the link. Effects arising from this are discussed on page 112.

The packet records in trace files vary in size. This complicates locating packet records contributing to a particular entropy sample in a trace file. For more detailed analysis this requires the trace file to be indexed in some way.

In live interface monitoring, however, only the current state of the monitored link usually matters, i.e., no timing information is needed. Volume-based sampling produces traffic samples of constant volume, keeping the entropy estimation error constant for all traffic samples, therefore this sampling method is fully sufficient for live interface monitoring.

The sampling technique presented in the following section uses the timestamps in the packet records to
produce traffic samples covering fixed time intervals. This avoids the jitter on the time axis, possibly making it interesting for entropy plots derived from trace files.

### 7.3 Interval-based sampling

If trace files are considered, one possible solution to the time jitter inherent in volume-based sampling is to perform traffic sampling with respect to packet timestamps. For this type of traffic sampling, a constant window interval time $\tau$ is chosen. All packets whose timestamps belong to a particular interval belong to the same traffic sample. Figure 7.2 shows an example of interval-based sampling with the same input stream that was used in Figure 7.1:

![Interval-based sampling diagram](image)

Figure 7.2: Interval-based sampling: Packet arrivals (blue) on a time axis are associated with traffic samples depending on time intervals of duration $\tau$. Numbers above the packets indicate the beginning of the traffic sample with that number, numbers below the axis state the end of the respective traffic sample.

While this sampling method solves the time jitter issue, Figure 7.2 shows that the price for this benefit is high: Interval-based sampling can no longer guarantee a constant sliding window data volume $N_S$. E.g., traffic sample 8 does not contain any packets, while traffic sample 11 covers a busy time on the link and contains eight packets.

The variance in the sliding window data volume $N_S$ ($N_S = 0$ worst case) is a disadvantage of this sampling scheme: the expected entropy estimation error is not constant for the resulting samples, making comparisons between samples more difficult. For example, consider a comparison between samples 3 and 11 in Figure 7.2: an entropy value derived from sample 11 is based on a data volume four times larger than an entropy value derived from sample 3.

For live interface monitoring, interval-based sampling is not preferable, especially in comparison to volume-based sampling, because of the variation in the expected entropy estimation error. Apart from that, timing information is usually not required in live interface monitoring, because only the current state of the network matters. Interval-based sampling is thus mainly of interest for trace file analysis. However, it introduces fluctuations in the expected entropy estimation error. It thus merely shifts the problem from one axis to another.
The following section discusses another sampling method, which addresses the jitter and fluctuation issues inherent in the sampling methods discussed so far. It represents a combination of volume-based sampling and interval-based sampling, using regular start times for traffic samples but adjusting the end times so that the sliding window data volume $N_S$ is constant.

### 7.4 Combo sampling

For trace file processing, a combination of both sampling methods discussed above would be most desirable: Traffic samples should reflect a constant number of trace file records and be arranged linearly along the time axis. This can be achieved by letting traffic samples start at known times, but adjust the traffic sample’s end time so that a fixed number of $N_S$ packet records belongs to each traffic sample.

This sampling method is casually referred to as *combo sampling* in this thesis. An example for combo sampling is presented in Figure 7.3:

![Figure 7.3: Combo sampling: Traffic samples start at known times but end whenever $N_S = 5$ packets have been read. This leads to a situation where some packets are reused (green) for consecutive traffic samples while others are skipped (yellow).](image)

The code required for this sampling method is considerably more complicated than the code required for any of the previous two sampling methods: Packets may now need to be queued if a traffic sample’s end time lies after the subsequent traffic sample’s start time. All packets in between these two points in time need to be queued for the subsequent traffic sample. During periods of silence, a queue spanning several traffic samples may build up. In such a situation, only a part of the queued packet records may be removed (“popped”) from the queue while the subsequent traffic sample is processed. Other packet records must be read only, but kept in the queue for later traffic samples.

In Figure 7.3, queueing occurs for traffic sample 3: here, only two packets are collected before the start time of traffic sample 4. Therefore it is necessary to read three packets into traffic sample 4 to get the required $N_S = 5$ packets for traffic sample 3. These three packets are queued, so that they are available again once traffic sample 4 is being processed.

During periods of silence that are considerably longer than the sampling period, sequences of identical
Traffic sampling may arise: in Figure 7.3 this occurs for traffic samples 8 and 9. This will, of course, result in identical entropy values for these samples.

During busy times the opposite may happen: a traffic sample's end time may lie before the subsequent traffic sample's start time. In this case, all packets between these two points in time need to be skipped, i.e., they don't belong to any traffic sample. In Figure 7.3, this situation occurs for traffic sample 11: three excess packets that were included in this sample with the interval-based sampling scheme are skipped with combo sampling, thus keeping $N_S$ constant.

A disadvantage of combo sampling is that certain packets may belong to no traffic sample at all (if they are skipped) or to several traffic samples (if they are queued). On the one hand, events occurring during phases of skipping may therefore be unnoticed. However, one may assume the window size not to be adjusted very well if an entire event one wishes to detect completely fits into a skipping period. The next section examines sensible choices for the window size in more detail. On the other hand, short events that occur just before a longer period of silence may have an over-proportional footprint in a sequence of entropy samples, because adjacent traffic samples directly correlate due to overlaps.

Figure 7.4 shows the time jitter avoided by combo sampling. For volume-based sampling (red samples), the graph shows the estimated sample starting time, based on the average packet rate of $T_{20050722}$. For combo sampling (green samples), the actual starting time of the samples is shown. With volume-based sampling, high packet rate events appear longer than they really are, while low volume periods appear too short. Without additional indexing, volume-based sampling therefore suffers from a timing jitter which can displace samples considerably.

![Figure 7.4](image_url)

Figure 7.4: This plot compares combo sampling (green plot) with volume-based sampling (red plot) for the same trace file: volume-based sampling attributes identical amounts of time to all samples (τ was chosen here to normalise the plots). For volume-based sampling, the footprint of high packet rate events is longer than it is with combo sampling while periods of lower packet rate are shortened. The resulting time jitter from this can be considerable: the Windows Messenger spam event following the SYN flood event is displaced by more than 400 s.
7.5 Window size considerations

Combo sampling does not suffer from this problem. It is thus the preferred sampling method in this thesis.

7.5 Window size considerations

All sampling schemes discussed in the previous sections require the window size to be set prior to sampling, in terms of volume (i.e., $N_S$ fixed), interval time (i.e., $\tau$ fixed), or both ($N_S$ fixed, $\tau$ half-variable, that is: fixed start time, but variable end time). Note that for combo sampling, the preferred sampling method in this thesis, $\tau$ follows $N_S$, as motivated in Equation 4.1, and not vice-versa. Naturally, some questions arise from the choice of a window size:

- What is the effect of the window size $N_S$ on experimental results?
- Is there an optimum window size $N_S$ for which experiments yield the clearest results?

Section 7.5.1 examines the effects of window size variation in detail. Subsequently, Section 7.5.2 looks at ways to optimise the window size $N_S$.

7.5.1 Effects of window size variation

Chapter 5 described entropy estimation errors inherent in computable entropy measures. As mentioned before, the expected entropy estimation error is a function window size $N_S$: the law of large numbers leads to a larger expected entropy estimation error of small $N_S$, as outliers are more likely for small samples.

This observation appears to call for the use of large sliding windows in entropy-based network event detection. Yet events which are short in comparison to $\tau$ are progressively absorbed into the background as $N_S$ (and thus $\tau$) is increased. The following experiment examines this aspect. Before the experiment we quickly define the duration of a network event:

**Definition 7.5.1 (Duration of a network event)** Let $\varepsilon$ be a network event. The duration $\delta_\varepsilon$ of $\varepsilon$ is the time between the first and the last packet arrival of packets belonging to $\varepsilon$ in seconds, plus the time that the last event packet occupies the communication channel. Determining the exact $\delta_\varepsilon$ on a packet basis requires careful trace file analysis. For this thesis it suffices to estimate $\delta_\varepsilon$ from the samples that $\varepsilon$ affects in an entropy measurement.
Experiment 7.5.1 (Effects of window size variation)

Description: This experiment examines the effects of window size variation on the evaluation of \( T_{20050722} \). We know from previous chapters that this trace file contains several short Windows Messenger spam events. These events span approximately 1.5 seconds each. Furthermore, a comparatively long SYN flood event which extends over about 8 minutes (487 s to be precise) is present. The results of Experiment 4.2.1 and Example 5.2.1 are used as a reference for the present experiment.

Experiment 4.2.1 examined the entropy of \( T_{20050722} \) with a window size of \( N_S = 2000 \) packet records. This window size translates into a typical window interval time of \( \tau = 0.26 \) s if Equation 4.1 is considered. With these window dimensions, the Windows Messenger spam events led to T-entropy drops of about 1.75 bits/byte and the SYN flood event to a drop of approximately 2 bits/byte in Figure 4.2. Example 5.2.1 determined the SNR of the Windows Messenger spam events to be 21.66.

The present experiment is essentially a repetition of Experiment 4.2.1, except that \( N_S \) is increased by a factor of 100, i.e., to \( N_S = 200000 \) packet records. According to Equation 4.1, the traffic samples now extend across approximately \( \tau = 26 \) seconds each.

The effect of \( N_S \) on the noise and on the entropy of the events is observed.

Expectation: In Experiment 4.2.1, the duration of both events is long compared to the average value of \( \tau \), more precisely, it is always larger than \( 2 \tau \). In other words, patterns contributed by both events can dominate entire traffic samples. The T-entropy drops in Figure 4.2 can thus be explained.

In the present experiment the duration of the Windows Messenger spam events is less than \( \tau \). Now, the patterns contributed by a Windows Messenger spam event only play a minor role in the overall ensemble of patterns in a traffic sample. On the other hand, the duration of the SYN flood event is still larger than \( 2 \tau \). Hence, one may expect the Windows Messenger spam events to be less noticeable, while the effect of the SYN flood event on the entropy should largely be unchanged.

Also, the noise should be less predominant in the present experiment.

Observation: Figure 7.5 presents the result for \( N_S = 200000 \) packet records: the Windows Messenger spam events are not as clearly visible as they were in Experiment 4.2.1: their lowering effect on the T-entropy is now reduced to approximately 0.20 bits/byte. The lowering effect of the SYN flood event is unchanged, still amounting to almost 2.0 bits/byte, i.e., even at this window size, there are still traffic samples which are dominated by the SYN flood’s pattern contribution. Also, the noise is attenuated, as expected: it now only amounts to 0.025 bits/byte.
Another interesting, but less obvious, feature in the plots is a shift in the mean of the non-event samples. It decreases from 2.95 bits/byte in Figure 4.2 to 2.62 bits/byte in Figure 7.5. One explanation for this behaviour is the nature of network traffic and the patterns arising from it in the traffic samples. Consider a typical web page download, consisting of several flows, one each for the base document and some pictures in it. Usually, such a web page cannot be completely downloaded within a quarter of a second. If similar patterns arise from the individual flows, they are therefore distributed across different traffic samples and thus cannot lower a single traffic sample’s entropy. This is different with traffic samples of approximately 26 seconds: most web pages are completely downloadable within this time frame. Similar patterns arising from different flows can therefore lower the entropy of a traffic sample.

In summary, Experiment 7.5.1 shows that both noise and events that are shorter than $2\tau$ are affected by the choice of $N_S$. We may examine the relationship of these values in more detail with the signal-to-noise ratio measure defined on page 73.

Like in Example 5.2.1, the SNR of the Windows Messenger spam events may be computed considering only the last two hours of the trace file, i.e., $T_{20050722,15-17}$. Judging from Figure 7.5, $b = 2.5$ appears as a sensible value to separate event samples from non-event samples. With this value of $b$, an SNR of just below 8 was found for the Windows Messenger spam events.

The following section attempts to optimise the window size for the Windows Messenger spam events.
Window size optimisation

For short traffic samples with respect to \( \delta \), one may expect the SNR to rise as \( N_S \) (and thus \( \tau \)) is gradually increased. Here, entire traffic samples can still be dominated by the symbol contribution of a single event. This is definitely the case as long as the following condition holds:

\[
\tau \leq \frac{\delta}{2}
\]  

(7.1)

As long as \( \tau \) satisfies this condition, increasing the traffic sample size only leads to a decrease of \( \sigma(E') \) in Definition 5.2.1. One may thus expect a smoothing effect on the noise, while the event signatures are largely unchanged. Once \( \tau \) is increased beyond the limit given in Equation 7.1, increases continue to smooth the noise, but the median of the event samples also increases, thus attenuating the event signatures. The SNR drop between Experiment 4.2.1/Example 5.2.1 and Experiment 7.5.1 justifies the assumption that the rate of event signature attenuation exceeds the rate at which the noise decreases.

One may use an experiment to substantiate these expectations. This experiment investigates the behaviour of the SNR for sample sizes \( N_S \) that are in the wider vicinity of \( \delta/2 \):

**Experiment 7.5.2 (Traffic sample size sweep)**

**Description:** This experiment aims to find a traffic sample size which roughly maximises the SNR for the Windows Messenger spam events examined before in Experiment 7.5.1. To achieve this, \( N_S \) is increased in steps of 250 symbols in the range 250 . . . 20000. For each \( N_S \), a suitable discrimination threshold \( b \) is manually chosen according to Table 7.1\(^1\) and the SNR of the Windows Messenger spam events is computed for two different mappings: one mapping is the complex mapping defined in Section 6.6 (\( M_{27} \)), the other one is the variable-volume mapping used in Chapter 4 (\( M_5 \)). The result is presented in a plot.

**Expectation:** One may expect to find a local maximum of the SNR in the vicinity of the equality point of Equation 7.1. Again, trace file \( T_{20050722} \) is used, the same as in Experiment 7.5.1. We already know from previous experiments that the duration of the Windows Messenger spam events is about 1.5 s and that 2000 symbols roughly cover a quarter second, therefore equality should be reached at approximately \( N_S = 6000 \) symbols.

\(^1\)It is conceivable that the discrimination thresholds could be adjusted automatically through some adaptive algorithm. However, this is beyond the scope of this thesis.
7.5 Window size considerations

Table 7.1: Discrimination thresholds $b$ used for the mappings $M_{27}$ and $M_5$ during the traffic sample size sweep.

<table>
<thead>
<tr>
<th>$N_S$</th>
<th>$b$ for $M_{27}$</th>
<th>$b$ for $M_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>250</td>
<td>1.10 bits/byte</td>
<td>1.30 bits/byte</td>
</tr>
<tr>
<td>500</td>
<td>1.23 bits/byte</td>
<td>1.35 bits/byte</td>
</tr>
<tr>
<td>750</td>
<td>1.50 bits/byte</td>
<td>1.32 bits/byte</td>
</tr>
<tr>
<td>1000</td>
<td>1.50 bits/byte</td>
<td>1.50 bits/byte</td>
</tr>
<tr>
<td>1250</td>
<td>1.70 bits/byte</td>
<td>1.50 bits/byte</td>
</tr>
<tr>
<td>1500</td>
<td>1.80 bits/byte</td>
<td>1.50 bits/byte</td>
</tr>
<tr>
<td>1750</td>
<td>1.80 bits/byte</td>
<td>1.50 bits/byte</td>
</tr>
<tr>
<td>2000</td>
<td>1.80 bits/byte</td>
<td>1.50 bits/byte</td>
</tr>
<tr>
<td>2250 − 6000</td>
<td>2.00 bits/byte</td>
<td>1.50 bits/byte</td>
</tr>
<tr>
<td>6250 − 10000</td>
<td>2.00 bits/byte</td>
<td>2.00 bits/byte</td>
</tr>
<tr>
<td>10250 − 12000</td>
<td>2.50 bits/byte</td>
<td>2.00 bits/byte</td>
</tr>
<tr>
<td>12250 − 16750</td>
<td>2.50 bits/byte</td>
<td>2.26 bits/byte</td>
</tr>
<tr>
<td>17000 − 20000</td>
<td>2.60 bits/byte</td>
<td>2.26 bits/byte</td>
</tr>
</tbody>
</table>

Figure 7.6: SNR of Windows Messenger spam events in $T_{20070722}$ for $N_S = 250 \ldots 20000$ symbols in steps of 250 symbols. The complex mapping ($M_{27}$) described in Section 6.6 was used to acquire the samples connected by the green curve. The red curve describes the situation for raw IPv4 data (mapping $M_5$ from Chapter 4). For both curves, relatively small fluctuations exist in the left part of the plot (roughly for $N_S < 10000$). For $N_S > 10000$, there are comparatively strong fluctuations in the SNR. The SNR increases quickly between $250 < N_S < 2500$ and falls gradually for $N_S > 6750$. For relatively small $N_S$ between 1000 symbols and 3000 symbols, $M_{27}$ yields higher SNR values than $M_5$. However, the value for $N_S$ is not critical as long as it is – roughly speaking – in the range of $[3000 \ldots 10000]$.

The graph in Figure 7.5.2 can be separated into two parts: for both mappings, its left side ($N_S \leq 10000$) does not exhibit much fluctuation, whereas the right half shows strong fluctuations in the SNR. This can be explained with the alignment of events inside or across traffic samples:

- For short traffic samples, the events always straddle across several traffic samples, i.e., at least one traffic sample is exposed to the full entropy impact of the event. This keeps the median for event samples relatively stable.

- The median of the event samples becomes increasingly unstable as soon as $N_S$ no longer satisfies Equation 7.1. For example, assume $N_S = \delta_e$ for some event $e$. Two extremes are possible: either the
event exactly fits into a single traffic sample and thus can exert maximum impact on its entropy, or the event stretches from the middle of one traffic sample to the middle of the following traffic sample, minimising event impact on the entropy of both traffic samples.

Generally, the graph in Figure 7.5.2 meets the expectation in that it shows a local maximum at $N_S = 6750$ symbols, close to the expected length of 6000 symbols. For $N_S = 10500$, $N_S = 12500$, $N_S = 14250$, and $N_S = 18000$ the events happen to be well-aligned inside traffic samples. The noise is smoother for the associated window sizes $N_S$. As a result, the SNR peaks for these traffic sample sizes exceed the local maximum at $N_S = 6750$. However, maxima for $N_S$ which do not satisfy Equation 7.1 are too unpredictable to be useful for SNR optimisation.

In summary, one may record that the traffic sample size $N_S$ can be optimised for the specific duration $\delta_\varepsilon$ of some event $\varepsilon$. While this will impair the entropy footprint of shorter events, it will leave longer events unaffected. A viable setting would therefore be an optimisation of $N_S$ and $\tau$ according to the shortest events one wishes to observe according to Equation 7.1.

The following chapter uses the mapping and sampling techniques described in the last two chapters for the detection of real events.
Chapter 3 described various characteristics of network events that may be detectable with entropy measures. The present chapter analyses a number of events that were actually found in the set of trace files collected for this project. Some other event types discussed in Chapter 3 did not occur in this trace file set, these being major network equipment failures and worm outbreaks.

To investigate the impact of major network equipment failures, such failures were simulated by filtering certain packets from trace files. For worm outbreaks, third party trace files captured during the SQL Slammer worm outbreak were used.

The following sections discuss the events individually in detail.
8.1 SYN flood attacks

The SYN flood attack detected in $T_{20050722}$ has already been discussed extensively in various sections of Chapters 4, 6, and 7. For this reason, there will only be a brief discussion of this event type here.

Among the 52 trace files collected for this project at the UoA border gateway, $T_{20050722}$ contains the most pronounced SYN flood attack. Its average packet rate was found to be 12500 packets/s by trace file analysis: approximately 6.08 million packets were sent within the 487 s during which the attack was active. At the time when $T_{20050722}$ was recorded, the university had a 100 Mbps Internet connection. Judging from the residual effect on the entropy when filtering the SYN flood event in Section 4.3, it is likely that the attack congested this connection.

For entropy-based detection of high rate SYN flood attacks, the TCP SYN flag should suffice as an observable for mapping. Table 8.1 shows the details of such a simple mapping. This mapping Id assigned to this mapping is $M_{50}$ in this thesis.

<table>
<thead>
<tr>
<th>Packet properties</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP protocol=TCP and raised SYN flag</td>
<td>'a'</td>
</tr>
<tr>
<td>All other packets</td>
<td>'b'</td>
</tr>
</tbody>
</table>

Table 8.1: A simple mapping of the TCP SYN flag.

As discussed in Chapter 6 on page 98, one would expect a low background entropy for the SYN flag under normal conditions, and a SYN flood attack should thus increase the entropy. This holds as long as attack packets do not dominate the traffic. Figure 8.1 presents the effect of the SYN flood event in $T_{20050722,14−15}$ on this field’s entropy, confirming these expectations for this particular attack.

![Figure 8.1: A mapping of the TCP SYN flag for $T_{20050722,14−15}$ ($M_{50}$).](image)

The SYN flood attack is clearly visible, as it raises the entropy to about 0.5 bits/byte due to a more uniform distribution of 'a' and 'b' symbols. Windows Messenger spam events behave differently: they further increase the already high frequency of 'b' symbols, thus causing entropy drops to approximately 0.1 bits/byte.
A notable effect in Figure 8.1 is the regular pattern of entropy drops caused by Windows Messenger spam that were already observed in Chapter 4. With respect to the simple mapping used here, the additional UDP traffic from the Windows Messenger spam events introduces more 'b'-symbols than would be observed under normal conditions. Although the background entropy is already quite low at 0.22 bits/byte, these events cause entropy drops to roughly half of this value.

For \( b = 0.4 \) and sliding window dimensions of \( N_S = 5000/\tau \approx 0.65 \text{s} \), the SNR of the SYN flood in Figure 8.1 is roughly 8.20. With filtered Windows Messenger spam events, the SNR is about 8.28.

### 8.2 DDoS attacks

The set of trace files collected at the UoA border gateway does not appear to contain a DDoS attack directed against the University of Auckland. However, it is probable that the SYN flood discussed in the previous section is part of a botnet running a DDoS attack against an external host.

Note that this event was caused by a single machine. If this attack had been run from several machines, the university link to the Internet would probably have suffered from severe congestion during the attack.

### 8.3 Windows Messenger spam

Windows Messenger spam has served as an example of relatively short events in Chapter 4 and Section 6.2. Its characteristics have already been discussed in detail in these places. For this reason, only a brief summary of its properties is provided here.

The 52 trace files collected for the thesis project at the UoA border gateway all contain Windows Messenger spam events, mostly on a very regular basis as we know it from \( T_{20050722} \). One may therefore assume that Windows Messenger spam was a very common network event during the time of data capture.

Table 8.2 shows a sequence of packet records taken from the fourth Windows Messenger spam event in \( T_{20050722} \). The packet rate of the Windows Messenger spam events in this trace file is in the order of 13000 packets/s.

Table 8.2 suggests that source socket, destination port, transport protocol, IPv4 length, and IPv4 TTL could all be viable fields for the detection of this event: these fields all exhibit a rather polarised value distribution during Windows Messenger spam events. This is also the case for the payload which is not
<table>
<thead>
<tr>
<th>Timestamp [s usec]</th>
<th>Src socket</th>
<th>Dest socket</th>
<th>Protocol</th>
<th>IPv4 length</th>
<th>IPv4 TTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1121998965:98619</td>
<td>218.92.13.149:32853</td>
<td>130.216.199.27:1026</td>
<td>UDP</td>
<td>508</td>
<td>44</td>
</tr>
<tr>
<td>1121998965:98666</td>
<td>218.92.13.149:32853</td>
<td>130.216.197.70:1026</td>
<td>UDP</td>
<td>508</td>
<td>44</td>
</tr>
<tr>
<td>1121998965:98712</td>
<td>218.92.13.149:32853</td>
<td>130.216.199.28:1026</td>
<td>UDP</td>
<td>508</td>
<td>44</td>
</tr>
<tr>
<td>1121998965:98758</td>
<td>218.92.13.149:32853</td>
<td>130.216.199.29:1026</td>
<td>UDP</td>
<td>508</td>
<td>44</td>
</tr>
<tr>
<td>1121998965:98851</td>
<td>218.92.13.149:32853</td>
<td>130.216.199.28:1027</td>
<td>UDP</td>
<td>508</td>
<td>44</td>
</tr>
<tr>
<td>1121998965:98897</td>
<td>218.92.13.149:32853</td>
<td>130.216.197.70:1027</td>
<td>UDP</td>
<td>508</td>
<td>44</td>
</tr>
<tr>
<td>1121998965:98943</td>
<td>218.92.13.149:32853</td>
<td>130.216.199.29:1027</td>
<td>UDP</td>
<td>508</td>
<td>44</td>
</tr>
<tr>
<td>1121998965:98990</td>
<td>218.92.13.149:32853</td>
<td>130.216.197.71:1026</td>
<td>UDP</td>
<td>508</td>
<td>44</td>
</tr>
<tr>
<td>1121998965:99036</td>
<td>218.92.13.149:32853</td>
<td>130.216.197.71:1027</td>
<td>UDP</td>
<td>508</td>
<td>44</td>
</tr>
<tr>
<td>1121998965:99088</td>
<td>218.92.13.149:32853</td>
<td>130.216.197.72:1026</td>
<td>UDP</td>
<td>508</td>
<td>44</td>
</tr>
<tr>
<td>1121998965:99232</td>
<td>218.92.13.149:32853</td>
<td>130.216.199.44:1027</td>
<td>UDP</td>
<td>508</td>
<td>44</td>
</tr>
<tr>
<td>1121998965:99278</td>
<td>218.92.13.149:32853</td>
<td>130.216.197.85:1026</td>
<td>UDP</td>
<td>508</td>
<td>44</td>
</tr>
<tr>
<td>1121998965:99325</td>
<td>218.92.13.149:32853</td>
<td>130.216.197.85:1027</td>
<td>UDP</td>
<td>508</td>
<td>44</td>
</tr>
<tr>
<td>1121998965:99372</td>
<td>218.92.13.149:32853</td>
<td>130.216.199.46:1026</td>
<td>UDP</td>
<td>508</td>
<td>44</td>
</tr>
</tbody>
</table>

Table 8.2: Properties of 15 consecutive packets taken from the fourth Windows Messenger spam event in T20050722. This event systematically sends identical messages to UoA internal addresses: addresses from the subnets 197 and 199 appear to be probed one-by-one for Windows Messaging services on ports 1026 and 1027. Additional ICMP error message traffic brought about by unused IPv4 addresses or closed ports was not observed.

considered in Table 8.2 due to its size, but was already an object of study in Section 6.2. Furthermore, the third octet of the internal IPv4 address may also serve as an indicator, as the Windows Messenger spam events access internal addresses sequentially (as is the case in Table 8.2).

### 8.4 Worms

The trace files collected in the course of this project do not contain any major worm outbreaks. So in order to evaluate the impact of such an event on the entropy of a network stream, trace files from the special traces archive [12] of the U.S. National Laboratory for Applied Network Research (NLANR) were used instead. Amongst others, NLANR provides trace files recorded at various places in the USA during the SQL Slammer worm outbreak. The files used here originate from the Colorado State University. Unlike the files collected in the course of this project at UoA, NLANR’s SQL Slammer trace files consist of traffic snapshots of 90 s each, which were recorded at irregular times, roughly 2 to 3.5 hours apart.

As mentioned in Section 3.3.3, these trace files are anonymised and therefore not ideal for entropy-based event detection. The IPv4 addresses have been altered in a way that does not reflect the original entropy of these fields. Other fields suitable for entropy evaluation, such as the payload, were not recorded at all.

SQL Slammer used a buffer overflow bug in Microsoft SQL Server which was exploitable through UDP port 1434 (0x059a). The worm fits into a single UDP packet: its code is only 376 bytes long. The code does not change as the worm propagates, so the payload would have been a suitable observable for high entropy monitoring.
In the trace files provided by NLANR, the source port appears to be random. The UDP checksum is computed from the UDP pseudo header (which – apart from the actual UDP header – includes the addresses, the protocol field and the length field of the IP header) and the payload.

For this reason, at least three fields which contribute to the checksum change frequently during the spread of SQL Slammer: the IPv4 addresses and the source port field. One may thus expect the checksum field to exhibit a relatively high entropy, for both SQL Slammer packets and non-SQL Slammer packets. Even if the checksum field would not have been anonymised by NLANR, it would probably have been of little value for entropy-based detection of SQL Slammer.

Two fields from the SQL Slammer packets were chosen to evaluate the impact of SQL Slammer: the IPv4 protocol field and the source port field of TCP and UDP. The entropy of the protocol field is usually low (see Section 6.4.3, page 94), because of TCP’s dominance among transport protocols. Figure 8.2 shows the T-entropy of the protocol field for eight traffic snapshots recorded on 25th January 2003, side-by-side.

The packet rate at Colorado State University was approximately 29000 packets per second during capture. As the SQL Slammer event could be observed at this site for several hours, the snapshot size of approximately 90 seconds is the upper limit for the window size. In order to observe shorter events such as the one at the beginning of the fifth traffic snapshot (starting at 12:42:25), the window dimensions were arbitrarily set to much smaller values: $N_S = 8000$ packets and $\tau = 260\text{ ms}$.

The SQL Slammer worm broke out on 25th January at about 5:30 UTC. By 6:30, when the third traffic snapshot was recorded, the impact of SQL Slammer on the T-entropy of the protocol field had almost reached its maximum at the monitoring site: for the following three snapshots, recorded at 9:08, 12:42,
and 15:17, the T-entropy is only slightly higher. This is consistent with observations of Moore, Paxson, Savage, Shannon, Staniford, and Weaver [31]: the SQL Slammer worm reached its maximum scanning rate after just three minutes, and within 10 minutes, 90% of the vulnerable hosts were infected.

The increased entropy is the result of a larger than usual number of UDP packets among TCP packets. This can simply be visualised by comparing the transport protocol composition of a traffic snapshot recorded before SQL Slammer’s outbreak with another snapshot recorded during the outbreak. Below, two sequences of 120 I, T, or U characters (indicating the transport protocols ICMP, TCP, and UDP, respectively) are presented. Each character represents the transport protocol of one packet in order of arrival. The first sequence was generated from the first 120 packets of traffic snapshot one (starting at 00:54) and the second sequence from the first 120 packets of traffic snapshot five (starting at 12:42).

```
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
UUUUTUUUUUUTUTTTUTITUTTTTTUUTUUTUTTUTTTTUTTUUTTTTTUUTTTTTTTU
TUUTTTUTTTUTTTTTUTUTTUUTUTTUUTTTTUTTUUTTTTUTTTTUTTTTUTTU
```

In the first sequence, only three non-TCP packets are present, of which merely a single one is UDP. In contrast, the second sequence is completely interspersed with (mostly SQL Slammer) UDP packets.

Due to its characteristics, SQL Slammer is relatively easy to discriminate from other traffic. Many site administrators simply configured the network hardware to drop all UDP traffic with destination port 1434 once they learned about the worm. At Colorado State University, this measure was apparently taken between 15:18 and 18:51. For the last two traffic snapshots in Figure 8.2, the entropy is almost back to the level it had before the outbreak.

Figure 8.3 presents SQL Slammer’s impact on the entropy of the destination port field at Colorado State University. For simplicity, the mapping considers TCP and UDP destination ports jointly. As client host operating systems usually choose source ports randomly, the typical entropy of this field is relatively high (because of the TCP responses) with just over 4 bits/byte. Due to its fixed destination port and high scanning rate, SQL Slammer caused the entropy to drop for the central four traffic snapshots in Figure 8.3. The cause of the entropy step between the first two and the last two of these traffic snapshots is an open question.
Figure 8.3: The impact of SQL Slammer on the T-entropy of the TCP and UDP destination port fields at Colorado State University. SQL Slammer uses a fixed destination port. During the presence of SQL Slammer traffic, i.e. during the middle four traffic snapshots, this port appeared frequently, thus lowering the entropy of this field. The cause for the entropy step between the first two and the second two of these is unclear. For the last two traffic snapshots, the entropy is almost back to the level it had before the worm outbreak.

8.5 Scans

The trace files collected for this project at the UoA border gateway contain dozens of address scans. Many of them appear to scan the university network for open SSH ports (default port: 22). Table 8.3 presents a typical packet sequence from such an event.

<table>
<thead>
<tr>
<th>Timestamp [µs]</th>
<th>Src socket</th>
<th>Dest socket</th>
<th>IPv4 proto</th>
<th>IPv4 length</th>
<th>SYN flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>1120796057:566394</td>
<td>63.110.140.12:22024</td>
<td>130.216.6.27:22</td>
<td>TCP</td>
<td>48</td>
<td>True</td>
</tr>
<tr>
<td>1120796057:566409</td>
<td>63.110.140.12:22024</td>
<td>130.216.6.29:22</td>
<td>TCP</td>
<td>48</td>
<td>True</td>
</tr>
<tr>
<td>1120796057:566411</td>
<td>63.110.140.12:22024</td>
<td>130.216.6.31:22</td>
<td>TCP</td>
<td>48</td>
<td>True</td>
</tr>
<tr>
<td>1120796057:566413</td>
<td>63.110.140.12:22024</td>
<td>130.216.6.30:22</td>
<td>TCP</td>
<td>48</td>
<td>True</td>
</tr>
<tr>
<td>1120796057:566420</td>
<td>63.110.140.12:22024</td>
<td>130.216.5.147:22</td>
<td>TCP</td>
<td>48</td>
<td>True</td>
</tr>
<tr>
<td>1120796057:566426</td>
<td>63.110.140.12:22024</td>
<td>130.216.5.168:22</td>
<td>TCP</td>
<td>48</td>
<td>True</td>
</tr>
<tr>
<td>1120796057:566444</td>
<td>63.110.140.12:22024</td>
<td>130.216.5.166:22</td>
<td>TCP</td>
<td>48</td>
<td>True</td>
</tr>
<tr>
<td>1120796057:566446</td>
<td>63.110.140.12:22024</td>
<td>130.216.6.33:22</td>
<td>TCP</td>
<td>48</td>
<td>True</td>
</tr>
<tr>
<td>1120796057:566448</td>
<td>63.110.140.12:22024</td>
<td>130.216.6.37:22</td>
<td>TCP</td>
<td>48</td>
<td>True</td>
</tr>
<tr>
<td>1120796057:566450</td>
<td>63.110.140.12:22024</td>
<td>130.216.6.32:22</td>
<td>TCP</td>
<td>48</td>
<td>True</td>
</tr>
<tr>
<td>1120796057:566452</td>
<td>63.110.140.12:22024</td>
<td>130.216.6.35:22</td>
<td>TCP</td>
<td>48</td>
<td>True</td>
</tr>
<tr>
<td>1120796057:566454</td>
<td>63.110.140.12:22024</td>
<td>130.216.6.34:22</td>
<td>TCP</td>
<td>48</td>
<td>True</td>
</tr>
<tr>
<td>1120796057:566456</td>
<td>63.110.140.12:22024</td>
<td>130.216.6.36:22</td>
<td>TCP</td>
<td>48</td>
<td>True</td>
</tr>
<tr>
<td>1120796057:566458</td>
<td>63.110.140.12:22024</td>
<td>130.216.5.159:22</td>
<td>TCP</td>
<td>48</td>
<td>True</td>
</tr>
<tr>
<td>1120796057:566461</td>
<td>63.110.140.12:22024</td>
<td>130.216.5.161:22</td>
<td>TCP</td>
<td>48</td>
<td>True</td>
</tr>
</tbody>
</table>

Table 8.3: Properties of 15 consecutive packets taken from an address scan in T_{20050708}. UoA internal addresses are scanned for open SSH ports in quick succession.

Attackers may use the data retrieved from such address scans to run SSH dictionary attacks against any host in the target network providing an SSH service. Apart from this, attackers may scan SSH servers for particular software versions. This is particularly easy, because SSH servers report their version to clients by default, as the following example demonstrates:
The version information may be useful for attackers that search for known vulnerabilities in SSH server software. Hosts running vulnerable SSH servers may subsequently be compromised with forged SSH requests.

Figure 8.4 demonstrates the impact of the address scan partially listed in Table 8.3 on the T-entropy. The mapping used here is $M_{27}$, which has already been described in Section 6.6. The sliding window parameters are $N_S = 5000$ and $\tau \approx 900 \text{ms}$ (chosen according to Equation 4.1). Due to the brevity of the event, only a short section of $T_{20050708}$ is presented in the figure. This trace file also contains a number of Windows Messenger spam events. They were filtered out to obtain a clearer view of the SSH address scan.

![Graph showing T-entropy over time](image)

**Figure 8.4**: An SSH address scan in $T_{20050708}$. The multi-byte mapping presented in Figure 6.12 ($M_{27}$) was used to generate this plot. The address scan ran for about four seconds. In this time, more than 65000 hosts were scanned.

### 8.6 SSH dictionary attacks

As already mentioned in Section 3.4.6, SSH dictionary attacks introduce regular patterns into network streams. For this reason, they were considered here, although they have a relatively low packet rate compared to other attacks covered in this chapter. The low packet rate is caused by the need to establish a TCP connection between the attacker and the victim host. Also, SSH dictionary attacks usually affect individual hosts rather than entire networks, unlike other attack types discussed in this chapter.

In a first attempt to detect this type of attack, one might use a mapping that monitors for new con-
connections on TCP port 22, i.e., for SYN flags. However, this would lead to ambiguity with SSH address scans as discussed in the previous section. Fortunately, there is a fundamental difference that permits discrimination between these two event types.

SSH is a service that is commonly found on Unix and Linux operating systems. On Windows, which is used for the vast majority of end-user desktop hosts at UoA, this service is relatively uncommon. Most hosts involved in an address scan will thus not respond to connection attempts.

On the other hand, it only makes sense to run SSH dictionary attacks against hosts that actually provide an SSH service, i.e., connections are established. However, new connections are usually not required for each authentication attempt, as most SSH servers permit three failed authentications before they close the connection. When a connection is closed, the SSH server sets the TCP FIN flag. A mapping for SSH dictionary attack detection may thus consider the TCP port fields in combination with the TCP FIN flag:

<table>
<thead>
<tr>
<th>Packet properties</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP protocol=TCP and (source or destination) TCP port=22 and TCP FIN flag raised</td>
<td>'a'</td>
</tr>
<tr>
<td>All other packets</td>
<td>'b'</td>
</tr>
</tbody>
</table>

Table 8.4: One possible mapping for the detection of SSH dictionary attacks: this mapping identifies packets which close SSH connections.

The typical duration of legitimate SSH connections ranges from seconds to several days. For this reason, very few packets should meet the condition for ‘a’ symbols in Table 8.4 under normal conditions.

However, during SSH dictionary attacks with potentially several hundred thousand failed authentication attempts, SSH connections are relatively short and packets meeting the conditions for ‘a’ symbols in Table 8.4 appear frequently. One would thus expect a less uneven mix of both symbols to increase the entropy.

$T_{20050713}$ contains an SSH dictionary attack with 314697 SSH connections that are explicitly closed with a FIN flag. The attack spans the entire trace file. For this reason, this trace file by itself cannot be used to understand the effect of the mapping defined in Table 8.4 on the entropy. However, a trace file with legitimate SSH connections only may serve this purpose: in $T_{20050902}$ only 109 SSH connections are explicitly closed with a FIN flag, which is a fairly typical value for a three-hour period at the UoA border gateway. Figure 8.5 presents entropy plots of both trace files considering the above mapping.

$N_S = 20000$ was used for both plots. For both trace files, traffic samples of this length cover approximately three seconds. As SSH dictionary attacks are usually events that last for minutes or even hours, this period appeared reasonable.
Figure 8.5: An SSH dictionary attack in $T_{20050713}$ (red) in comparison to normal SSH activity in $T_{20050902}$ (green). The attack spans the whole of $T_{20050713}$, however its intensity fades towards the end. For normal SSH activity, connections are rarely closed. Hence, the entropy of the samples taken from $T_{20050902}$ hardly deviates from the lowest possible level. During the attack in $T_{20050713}$, connections are closed frequently after failed login attempts. This raises the entropy of the traffic samples with attack traffic far beyond the minimum level.

For $T_{20050902}$, the entropy rarely rises above the lowest possible T-entropy for $N_S = 20000$ which is 0.00248 bits/byte. Given the low residual noise of $\sigma(E') = 0.0000057$ bits/byte during non-event times such as the period covered by $T_{20050902}$, this event type should easily be detectable at the UoA border gateway. Due to the fading intensity of the attack in $T_{20050713}$, the SNR was computed separately for the first 2000 and the last 2000 seconds of $T_{20050713}$: it is 579.15 and 291.05, respectively. Therefore, the detection of high rate SSH dictionary attacks should not be difficult with the mapping presented above.

### 8.7 ICMP floods

Several ICMP floods were detected in the trace files collected at the UoA border gateway. As an example, an ICMP flood located in $T_{20050926}$ is discussed here. Table 8.5 presents a typical traffic snapshot from this trace file during the attack.

According to the payload, the packets in Table 8.5 report a TTL expiry of ping packets. However, there are no incoming ICMP packets to which those in Table 8.5 would contribute a valid reply. As the source IPv4 address in Table 8.5 does not change often, this is not likely to be the result of a Smurf attack (cf. Section 3.4.8 and Shay [90], page 339).

Figure 8.6 gives an impression of the this event’s impact on the entropy. This event was detected with $M_{27}$, the complex mapping defined in Section 6.6.

The duration of this event is approximately 1.43 s. During this time, more than 31000 ICMP packets...
8.8 Simulation of network equipment failures

The set of trace files recorded for this thesis does not contain outages affecting larger parts of the university network. In order to be able to examine the effect of such an event on the entropy, a network hardware
fault was simulated. The simulation removed packets according to certain properties out of the trace files. The properties chosen here are:

1. IPv4 addresses: Packets matching a selected address range were removed.

2. Transport protocol. A misconfiguration or fault in network equipment may lead to situations where certain transport protocols no longer appear on a network link. For the simulation a transport protocol number was chosen and packets using the associated transport protocol were filtered.

The following experiments discuss the these two fault simulation scenarios.

Experiment 8.8.1 (Fault simulation and the entropy of internal IPv4 addresses)

Description: Damaged network cables or broken router ports are real world events that change the frequency of certain IPv4 addresses in a network stream. If a router port or a cable is defective, certain addresses appear with a different frequency than they would otherwise do.

To an extent, it is possible to simulate such events by selecting a subset of IPv4 addresses and filter packets in which one of these addresses appears. Generally, one should not filter all packets matching the selected address range to keep this type of simulation realistic, as some traffic may be initiated from outside the selected address range. One of the filters used in this section considers this, at least for TCP. Furthermore, a sudden cable break would lead to TCP packet retransmissions for unacknowledged packets. A simulation of this effect would not be easy to implement and the accuracy gain would only apply to the first few seconds of the simulation until the interrupted TCP connections time out. For these reasons, the simulation here does not reflect the TCP retransmission effect.

In order to ascertain the maximum possible impact of such an event on the entropy, the experiment also removed all packets from and to the selected address subset in a second run.

As mentioned before, the trace file used for this simulation, \( T_{20071017} \), was collected at an internal campus gateway. In order to get an overview of the subnet distribution at this gateway, Figure 8.7 presents a histogram of the third octet of internal addresses (i.e., of addresses with the prefix 130.216).

If source and destination address are both internal, the histogram gives higher counting priority to the Tamaki address range (130.216.120.0 to 130.216.123.255, 130.216.126.0/24 and 130.216.192.0 to 130.216.207.255).

\( T_{20071017} \) contains a total of 212.7 million packets. In order to simulate a major network equipment outage, traffic to or from the three most active subnets, i.e., 195, 200, and 207, was filtered out. This
was done in two ways:

- **Filter 1:** Addresses were removed largely as described in Section 3.3.1: traffic leaving the chosen subnets was removed completely, TCP traffic destined for any of these subnets was removed if the ACK flag was set. Instead of removing DNS responses only, all DNS traffic to any of these subnets was also filtered for simplicity. Removing DNS responses only would have required relatively deep packet inspection for what may be considered relatively little accuracy improvement:
  
  - Given that most of the UoA server infrastructure is located at the city campus, there should be more outgoing than incoming DNS requests and more incoming than outgoing DNS responses at the Tamaki campus gateway.
  
  - This effort with UDP traffic would only be justified if the same effort was made with other transport protocols observed at the gateway, such as ESP and PIM.

In this scenario, approximately 82.7 million packet records were removed, which is approximately 40% of all packet records in the trace file.

- **Filter 2:** All traffic to and from the chosen subnets was removed. For this test, 112.7 million packet records were filtered, which is roughly 53% of the trace file records.

The experiment uses mapping $M_{46}$, which considers the third octet of the internal IPv4 address.

**Expectation:** Depending on the filter used, certain patterns are removed from the packet stream. This should lead to entropy drops.

**Observation:** Figure 8.8 shows the effects on the average T-entropy for both simulations.
The artificial network events are clearly visible as entropy drops in the graph. For both simulation scenarios, SNR tests were carried out with a discrimination threshold of $b=1.17$ bits/byte. In both cases, this resulted in a residual noise of $\sigma(E')=0.149$ bits/byte. This is relatively high in comparison to other experiments in this thesis, even though a comparatively large sliding window with $N_S = 60000$ and $\tau = 2\,s$ was used. Because of this residual noise level, one obtains an SNR of 3.08 for the test with Filter 1 and an SNR of 3.86 for the test with Filter 2. In practice, the low SNR may make such events difficult to detect with entropy measures, even though both filters remove large portions of the traffic. However, it may be possible to increase the SNR by using an even larger sliding window at the price of a slower detection. As network equipment failures and misconfiguration problems are events that are likely to span at least several minutes, it should be safe to do so.

The comparatively strong residual noise in this simulation might be able to be explained by packet bursts from certain addresses: for instance, if a file stored remotely on one of UoA’s AFS (Andrew File System) servers at the city campus is opened in Tamaki, the IPv4 address of the client may suddenly appear quite frequently and cease to appear for a while once the file has been opened. Because of this, another experiment was run with a window size of $N_S = 300000$ and $\tau = 10\,s$. The residual noise is reduced to 0.116 bits/byte and the SNR of the artificial network events is 3.31 and 4.36, respectively, for the same discrimination threshold $b=1.17$ bits/byte.

The following experiment examines packet record filtering with respect to the transport protocol.
8.8 Simulation of network equipment failures

Experiment 8.8.2 (Fault simulation and the entropy of the IPv4 protocol field)

Description: This experiment discusses a router fault simulation by filtering packet records from $T_{20071017}$ according to their transport protocol. In practice, this type of fault may arise from misconfiguration or faulty software. As in the previous experiment, we start with a histogram to get an overview of the transport protocol distribution of this trace file, which is presented in Figure 8.9.

![Histogram of Transport Protocol Distribution](image)

Figure 8.9: Transport protocol distribution at the Tamaki campus gateway. As one would expect, TCP (protocol number 6) has the highest count, followed by UDP (protocol number 17). ESP (protocol number 50) also has a relatively high count, which may be an indicator for the use of IPsec on the monitored link.

Seven different transport protocols appear in $T_{20071017}$: TCP, UDP, ESP, ICMP, PIM, OPSFIGP, and IGMP (listed by frequency). The roughly 48 million UDP packets in $T_{20071017}$ were filtered.

This experiment uses mapping $M_{13}$, which considers the IPv4 protocol field.

Expectation: As in the previous experiment, the variety of the patterns in the packet stream was reduced with the filter. Again, this should lead to a drop in the entropy for affected samples.

Observation: Figure 8.10 presents the result of this experiment. An event was simulated for the samples between 7500 s and 7750 s.

The simulated event in Figure 8.10 is clearly visible. Its SNR is 4.94 for a discrimination threshold of $b = 0.3$, with a residual noise of $\sigma(E') = 0.088$ bits/byteS. The residual noise is lower than in the previous section although the sliding window used here is smaller: $N_S = 30000$ and $\tau = 1$ s.

At approximately 0.55 bits/byte, the background entropy for the IPv4 protocol field is relatively low. The reason for this is the small transport protocol variety on the monitored link and TCP’s clear dominance among these transport protocols.
8.9 Conclusions

This chapter demonstrated that a wide variety of network events may be detected with entropy measures. Two main event types were considered: events that add new patterns to a network stream and events that remove patterns from a network stream. Generally, the impact on the entropy appears to be stronger with added patterns than with removed patterns.
This chapter examines the sensitivity of T-entropy from two viewpoints: Section 9.1 discusses the sensitivity with respect to simulated events occurring at a variety of packet rates. This can serve as a rough estimate of the amount of event traffic required to provoke a certain entropy response. Subsequently, Section 9.2 looks at sensitivity comparisons between different information measures. A large part of this section deals with difficulties in undertaking such comparisons.

9.1 Sensitivity of T-entropy to network events at different packet rates

This section examines the sensitivity of T-entropy to SYN flood traffic as it appears in $T_{20050722}$. To do this, identical traffic was injected into this trace file at a variety of packet rates.
Experiment 9.1.1 (Sensitivity of T-entropy to SYN flood traffic)

Description: This experiment injects SYN flood traffic identical to that present in $T_{20050722}$ at a different location into this trace file. We know the average packet rate of this trace file to be roughly 8000 packets/s from previous experiments. For convenience, the experiment removes the Windows Messenger spam events in this trace file. Mapping $M_{27}$ is applied, with sliding window parameters $N_S = 5000$ and $\tau = 0.654535 \text{ s}$. The traffic injection is located after 3000 s to permit comparison with the real SYN flood attack. The duration of each artificial network event is 375 s. This experiment uses the event simulation described in Section 4.4.1.

Expectation: The effect of the artificial network event traffic on the T-entropy should increase with the injection rate.

Observation: Figure 9.1 shows the result of the experiment for injection rates between 500 packets/s (green, just noticeable) and 40000 packets/s (orange, entropy drop below that of the real SYN flood attack). Table 9.1 shows the associated signal-to-noise ratios.

![Figure 9.1](image_url)

Figure 9.1: The real SYN flood attack in $T_{20050722}$ (left) in comparison to a number of artificial SYN flood events at a variety of packet rates (right). From this plot one may conclude that an event of approximately 20000 packets/s is necessary to produce an entropy drop of about 2 bits/byte on the UoA Internet link. However, from Section 8.1 we know that the actual packet rate of the real SYN flood is only 12500 packets/s.

<table>
<thead>
<tr>
<th>Packet rate</th>
<th>SNR</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>500 packets/s</td>
<td>2.46</td>
<td>3.28 bits/byte</td>
</tr>
<tr>
<td>1000 packets/s</td>
<td>4.25</td>
<td>3.24 bits/byte</td>
</tr>
<tr>
<td>2000 packets/s</td>
<td>7.28</td>
<td>3.08 bits/byte</td>
</tr>
<tr>
<td>5000 packets/s</td>
<td>14.87</td>
<td>2.80 bits/byte</td>
</tr>
<tr>
<td>10000 packets/s</td>
<td>24.64</td>
<td>2.5 bits/byte</td>
</tr>
<tr>
<td>20000 packets/s</td>
<td>33.81</td>
<td>1.70 bits/byte</td>
</tr>
<tr>
<td>40000 packets/s</td>
<td>38.21</td>
<td>1.00 bits/byte</td>
</tr>
</tbody>
</table>

Table 9.1: signal-to-noise ratios for the artificial network events. The SNR calculation is based on the samples between 2950 s and 6000 s. As expected, the SNR values, which indicate the impact of the artificial network events on the entropy, rise as the injection packet rate is increased.
Figure 9.1 and Table 9.1 confirm the expectation: the impact on the entropy increases with the injection packet rate. Furthermore, one may get a rough impression of the impact of events with certain packet rates on a link transporting approximately 8000 packets/s, such as the one at UoA. The reason for the inaccuracy of this impression lies in the event simulation technique: as mentioned earlier, the simulator is not capable of modelling the effect of the injected packets on the real traffic. For instance, the real SYN flood event has an SNR of 35.03, based on the samples between 2000 s and 6000 s and $b = 2.5$. One might thus conclude that the packet rate of this event must be in the order of 20000 packets/s to 25000 packets/s, judging from the values in Table 9.1. However, we know from Section 8.1 that a relatively precise value for the packet rate is actually 12500 packets/s, i.e., the real event requires a lot less traffic for the same impact.

Viewed from another aspect, this reinforces the conjecture that the SYN flood attack in $T_{20050722}$ actually had an effect on the other traffic on the link: it is likely that the attack “squeezed” other traffic out, thus making itself more predominant on the link. This effect cannot be modelled with the event simulator used here, and a similar effect can only be achieved with higher packet injection rates.

Steep entropy drops to minimal entropy levels during packet injection are the result of short periods of silence in the trace file and have already been described on page 69. In Figure 9.1, these occur during the artificial network events with 40000 packets/s (orange) and 10000 packets/s (magenta).

The squeeze-out issue with the event simulator described in Experiment 9.1.1 is more severe at high injection rates. One may thus safely conclude that SYN flood events with a packet rate of approximately 1000 packets/s or more satisfy Equation 5.2 in the context of $T_{20050722}$ and would thus be detectable with T-entropy.

### 9.2 Sensitivity comparisons between different measures

A first (purely visual) comparison of T-entropy, normalised complexity, and LZ76 was carried out in Section 4.2. To quantify the sensitivity of these measures for a comparison, one might consider the following properties:

- the SNR of the Windows Messenger spam events on each measure
- the SNR of the SYN flood for each measure
- the residual noise level
Generally, the measures react similarly to patterns in the data. However, there are at least two circumstances which complicate a direct comparison between the measures:

- The strength of a reaction to a particular pattern may be different. For instance, compare the high entropy peaks just before the SYN flood in Figures 4.2 and 4.4: the peaks appear far more pronounced for LZ76 than for T-entropy.

- The value ranges of the measures are different. While this does not affect SNR measurements, a comparison of the residual noise is only possible with normalised values.

These circumstances make it difficult to derive absolute statements about a measure’s sensitivity from SNR measurements alone. For instance, in Figures 4.2 and 4.4, LZ76 appears to be more sensitive to the high entropy/complexity samples just before the SYN flood than T-entropy. However, in the same plots LZ76 (visually) appears to be less sensitive to Windows Messenger spam. One might even attempt to validate this observation with SNR measurements: for example, one may choose discrimination thresholds $b=1.5$ bits/byte (T-entropy) and $b=14000$ production steps (LZ76) to find the corresponding SNRs as 19.74 and 18.37, respectively.

How vulnerable such SNR comparisons between different measures really are becomes clear as soon as one chooses slightly different discrimination thresholds: the sensitivity to Windows Messenger spam looks considerably better for LZ76 if the discrimination thresholds are closer to the background entropy, $b=2.5$ bits/byte (T-entropy) and $b=21000$ production steps (LZ76), for instance. The corresponding SNRs here are 15.39 and 19.74, respectively.

Concerning the SYN flood event, the same observations were made: for discrimination thresholds of $b=1.36575$ bits/byte (T-entropy) and $b=10900$ production steps (LZ76), the SNRs are 21.77 and 19.21, respectively. If one chooses the discrimination thresholds as 2.0 bits/byte and 16000 production steps, the sensitivity of LZ76 with an SNR of 28.82 appears better than that of T-entropy, whose SNR is 27.50.

Again, it depends on the discrimination thresholds whether T-entropy or LZ76 appears to be more sensitive, although for all tests the values were chosen in a relatively “fair” way: after setting a discrimination threshold for one measure, the corresponding discrimination threshold for the other measure was chosen such that its position was comparable with respect to certain key samples. For instance, the discrimination thresholds $b=1.5$ bits/byte (T-entropy) and $b=14000$ production steps (LZ76) were chosen to be just above the entropy/complexity of the least pronounced Windows Messenger spam event (for both measures that is the Windows Messenger spam event just after the SYN flood).
These observations demonstrate that SNR measurements are not a valid criterion for the comparison of different information measures. This leaves only the residual noise level for consideration.

A simple normalisation strategy for the residual noise might consider the quotient $\sigma(E')/\mu(E')$ as an indicator for the relative level of the residual noise. For the entropy samples plotted in Figures 4.2 and 4.4, the values of this quotient are similar: 0.151962 for T-entropy and 0.153432 for LZ76, indicating that LZ76 exhibits marginally more noise for mapping $M_5$ and a window size of $N_S = 2000$. However, these differences are probably too small to be meaningful. The proximity of the two values suggests that even the simple quotient $\sigma(E')/\mu(E')$ works reasonably well as an indicator.

Instead of directly comparing the residual noise level of the two measures, one might compare the relative residual noise behaviour with respect to different sliding window sizes $N_S$ for the measures. In this case, one may normalise the residual noise to 100% for a certain window size. The following experiment adopts this strategy.

**Experiment 9.2.1 (Residual noise comparison for LZ76, LZ78, normalised complexity and T-entropy)**

**Description:** This experiment examines the relative residual noise levels of LZ76, LZ78, Szczepański’s normalised complexity (cf. [97] and Section 2.3.4), and T-entropy for sliding window sizes $N_S \in \{32, 64, 128, 256, 512, 1024, 2048, 4096, 8192\}$. The noise levels are normalised to 100% for all measures at $N_S = 8192$. The experiment uses 100 s of traffic samples from $T_{20050722,15:00:00}^{15:01:40}$ and the fixed-volume mapping $M_{13}$.

**Expectation:** According to the law of large numbers, the residual noise level should rise as $N_S$ is decreased. If one measure synchronises more quickly into patterns in the traffic samples, the normalised noise of this measure should rise less quickly than that of the others.

**Observation:** Figure 9.2 shows that the residual noise level of T-entropy rises less quickly for shorter $N_S$ than all other measures considered here.

The residual noise of LZ76 rises most quickly as $N_S$ is decreased. This is likely to be related to the last production step dilemma inherent in LZ76: in most cases, LZ76 terminates on a non-exhaustive production step rather than an exhaustive production step. At parsing position $n$, the typical step size is approximately $\log(n)$. Consequently, the maximum error associated with the last step is also in the order of $\log(n)$. For short inputs, this error is usually larger than for large inputs: consider two inputs of 64 bytes and 1000000 bytes. For the first input, the error is in the order of $\log(64)$ bytes=6 bytes,
which is almost 10%, and for the second input the error is in the order of $\log(1000000)$ bytes $\approx 20$ bytes, which is only about 0.002% of the input length.

Szczepański’s entropy estimator behaves much like T-entropy for window sizes down to approximately 128 bytes. Below this size, it also deviates from T-entropy, an effect which is presumably also related to the last production step inaccuracy in LZ76 described above.

Although one may conclude that the sensitivity of all measures becomes more similar for larger $N_S$ (cf. Experiment 4.2.1, for instance), LZ76 and normalised complexity (which is derived from LZ76) in particular become quickly less attractive as $N_S$ increases, due to the time complexity of the LZ76 algorithm used here. This is documented experimentally in Figure 9.3, which shows the average running times per traffic sample for LZ76 and T-entropy. LZ production complexity and normalised complexity would become useful measures for large $N_S$, if and when an implementation of the algorithm described by Rodeh et al. [87] becomes available.

9.3 Conclusions

The sensitivity tests carried out in Section 9.1 suggest that the lowest SYN flood packet rate detectable in the context of $T_{20050722}$ is roughly 1000 packets/s. The tests for higher packet rates in Section 9.1 reveal a problem with the events simulation technique used in this thesis: as the event generator used here cannot model congestion effects, entropy changes caused by artificial network events require significantly higher packet injection rates than real events.
9.3 Conclusions

Figure 9.3: Running times for traffic sample sizes $500 \leq N_S \leq 20000$ with a step size of 500 for LZ76 and T-entropy: with $ftd$, the T-entropy computation terminates in $O(N_S)$ as long as the pointer size required to address substrings in a traffic sample does not exceed the pointer size of the machine running $ftd$. At the time of writing, the latter is 32 bits, i.e., as long as $N_S < 2^{32}$ bytes, $ftd$ will terminate in $O(N_S)$ time. The running times shown here were experimentally produced via the Linux/Unix `time` command. The running times for LZ76 are associated with an implementation of Algorithm 2.2.1. According to Rodeh [87] et al., faster execution is possible, but no implementation of the algorithm Rodeh et al. describe was available for experimentation.

Section 9.2 revealed that SNR comparisons between different information measures depend somewhat on the choice of the discrimination thresholds $b$. For this reason, such comparisons are at least problematic, if not invalid. However, it is possible to make statements about one factor that influences the sensitivity of information measures: the level of residual noise. We know from Sections 5.1 and 7.5.1 that the residual noise is a function of the sliding window size $N_S$. For this reason, Experiment 9.2.1 compared the residual noise levels for T-entropy and LZ76 for various $N_S$. The result of this experiment suggests that the residual noise level of LZ76 rises more quickly than that of T-entropy as $N_S$ is decreased.
10

Multi-Dimensional Traffic Analysis

10.1 Motivation

The past chapters considered one-dimensional traffic analysis only, i.e., for each measurement, a single mapping or information measure was considered. However, it is conceivable to run several mappings or measures in parallel for traffic analysis. This may yield more detailed information about events at the cost of additional computing resources. This chapter examines these two scenarios: Section 10.2 examines the topic of multiple mappings. Subsequently, Section 10.3 discusses multiple measure traffic analysis. These techniques are meant to complement, not replace, existing techniques such as those described by Cilibrasi and Vitányi [29], for instance.

Similar experiments to those presented in the following section were carried out by Xu, Zhang, and Bhattacharyya [117], Kulkarni and Bush [61], and Lakhina, Crovella, and Diot [63]. The novel approach here is the use of mappings and T-entropy as an information measure.
10.2 Traffic analysis with multiple mappings

The consideration of $n$ mappings in traffic analysis opens an $n$-dimensional sample space for entropy samples, where each axis is assigned the entropy of one mapped observable. Of course, the same traffic samples must be used for all mappings contributing to a multi-dimensional entropy sample. In addition to the entropy dimensions, a time dimension may be added if one is interested in the behaviour of the selected observables over time.

This chapter limits the number of dimensions/mappings under consideration to two, because the arrangement of samples in space is not comprehensible in hardcopies of 3D entropy diagrams.

The following experiment is a first attempt at multi-dimensional entropy analysis.

**Experiment 10.2.1 (Plot of the entropy of the TCP SYN flag against the entropy of the TCP/UDP destination port)**

**Description:** This experiment uses two mappings of $T_{20050722}$ for a two-dimensional entropy analysis. The observables for the two mappings are the TCP SYN flag and the destination port (both TCP and UDP). $N_s = 5000$ and $\tau = 0.654535s$ (in accordance with Equation 4.1).

**Expectation:** The arrangement of the entropy samples in the two-dimensional plane may lead to clustering of certain event-related samples.

**Observation:** Figure 10.1 presents a plot of the SYN flag entropy against the entropy of the TCP/UDP destination ports. Indeed, some event-related samples form sample clusters in the plane.

The background entropy in Figure 10.1 forms a sample cluster around the coordinate (0.225,3.4). The green sample cluster is related to the SYN flood event in $T_{20050722}$ that has already been an object of study in previous chapters. From Section 8.1 we know that this event increases the entropy of the SYN flag field; from Section 4.3 we know that packets belonging to this event access TCP destination port 5406. It is therefore not surprising that the entropy of the TCP/UDP destination port observable decreases during the event. As a result, the samples associated with this event are located in the lower right section of the diagram.

The dark blue sample cluster in Figure 10.1 is related to the Windows Messenger spam events in $T_{20050722}$, which also have been discussed before. Windows Messenger spam lowers the entropy of the SYN flag field as already mentioned in Section 8.1. Like the SYN flood event, Windows Messenger spam
10.2 Traffic analysis with multiple mappings

Figure 10.1: $T_{20050722}$: T-entropy of the TCP SYN flag ($M_{50}$) against T-entropy of the TCP/UDP destination ports ($M_{31}$). The large red sample cluster around (0.225,3.4) is the background entropy. Some entropy samples associated with events have been coloured, e.g., SYN flood samples (green), Windows Messenger spam samples (dark blue). Furthermore, $T_{20050722}$ contains some low rate port scans (light blue) which appear concurrently during the last hour of the trace file, leading to more SYN flags and increasing the entropy of this observable. The two-dimensional arrangement of the samples thus permits spatial discrimination of various event types. The arrows roughly indicate the impact of each event type on the entropy with respect to the background entropy. The radii of the ellipse around the background entropy are four $\sigma(E')$ for mappings $M_{50}$ (horizontal) and $M_{31}$ (vertical). Hence, this ellipse is the two-dimensional equivalent of Equation 5.2, discriminating outliers from regular samples.

uses fixed destination ports, usually 1026 or 1027, lowering the entropy of the TCP/UDP destination port observable. Consequently, the samples associated with this event type are located in the lower left section of the diagram.

Apart from these previously discussed events, another event was found in $T_{20050722}$ with mapping $M_{50}$: during the last hour of the trace file, several low rate port scan events appear in parallel, scanning TCP for open ports 135, 445, 5556 and 9898. Therefore, more SYN flags appear during this period, increasing the otherwise low entropy of this field, while there is no obvious impact on the entropy of the destination port observable. The corresponding (light blue) samples appear in the right upper section of the diagram.

To ensure that the coloured samples really belong to the events mentioned above, the same samples were coloured identically in an entropy plot of the SYN flag ($M_{50}$) field against the time, shown in Figure 10.2.

Experiment 10.2.1 demonstrates an advantage of entropy analysis of network traffic with multiple mappings, arranged in multiple dimensions: the arrows in Figure 10.1 indicate that the spatial separation of the various event samples may guide the classification of events.

The two mappings/dimensions chosen for Example 10.2.1 would not permit the discrimination between
high rate port scans on a single port and SYN floods. If one wishes to discriminate between these events, one might want to add another mapping/dimension for the entropy of the internal IPv4 address.

This concept can be extended arbitrarily by adding more dimensions. Ultimately, one could have mappings/dimensions for all $n$ fields provided by a particular type of packet record and for additional observables as described in Section 6.5.

Not only multiple mappings permit multi-dimensional traffic analysis. The following section uses several complexity/entropy measures instead.

### 10.3 Traffic analysis with multiple measures

Computable information measures tend to overestimate the information content of strings to varying degrees (cf. Section 2.2.1). This property may be useful for the discrimination of events.

By assigning different information measures to the axes in a multi-dimensional plot, characteristic sensitivities of each measure to certain patterns may yield sample clusters in the sample space.

The measures considered in the present section are T-entropy, LZ production complexity, LZ78, and 1-gram Shannon entropy. All these measures ultimately behave like Shannon entropy: as the unpredictability of patterns in a string increases, the return value of an information measure for that string rises. Hence, one may expect sample distributions from combinations of these measures to be arranged roughly along the diagonal of a multi-dimensional sample space.
10.3 Traffic analysis with multiple measures

The following experiments all use $T_{20050722}$, due to the number of events present in this trace file. All experiments use identical sampling parameters of $N_S = 5000$ and $\tau = 0.654535$ s. Furthermore, all experiments use mapping $M_5$, as it permits all available fields to influence the information measures.

As in the previous section, the following set of experiments assigns distinct colours to sample clusters in plots. In particular, sample clusters related to the SYN flood and Windows Messenger spam events use the same colour codes as in the plots of Section 10.2. As the most predominant events in $T_{20050722}$ cause entropy drops, the background entropy forms a red sample cluster in the upper right section of the plots discussed in the experiments below. The colour codes are consistent for all four experiments, permitting observations about the impact of the different information measures examined here.

**Experiment 10.3.1 (1-gram Shannon entropy versus LZ production complexity)**

**Description:** This experiment plots the LZ production complexity versus the 1-gram Shannon entropy. The 1-gram Shannon entropy uses probabilities derived from individual symbol frequencies. It thus has a rather limited ability to recognise larger patterns in a data stream.

**Expectation:** One may expect a stronger overestimation from 1-gram Shannon entropy than from LZ production complexity, because of its limited ability to recognise patterns. If both measures are plotted against each other, the likely result is a convex sample arrangement.

**Observation:** The plot in Figure 10.3 has a convex shape, indicating overestimation of the 1-gram Shannon entropy for low entropies. The overestimation increases as the entropy/complexity of the samples decreases. A possible explanation for this lies in the nature of 1-gram Shannon entropy: low entropies/complexities – broadly speaking – manifest themselves in long repeated patterns. The 1-gram Shannon entropy misses these patterns completely. On the other hand, the 1-gram Shannon entropy misses less detail for high entropies/complexities, which – again broadly speaking – manifest themselves in relatively short patterns. LZ76 is able to recognise longer patterns, permitting this measure to estimate the information content more accurately, independent of the input’s complexity.

Apart from this, the plot exhibits a number of distinct sample clusters: while the SYN flood (green) and Windows Messenger spam (blue) samples are both located in the lower left section of the plot, they occupy clear-cut areas in the sample space.

With mapping $M_5$, the length of the packet records associated with Windows Messenger spam is 66 bytes, which is more than the 40 bytes of the SYN flood packet records. For this reason, the number of different values in the Windows Messenger spam packet signature is likely to be larger than that of
Figure 10.3: This plot examines the 1-gram Shannon entropy versus the LZ production complexity. Five different sample clusters are highlighted in different colours. These are SYN flood samples (green), high packet rate Windows Messenger spam samples (blue), low packet rate Windows Messenger spam samples (magenta), samples containing peculiar e-mail-related TCP ACKs (black), and samples affected by a high-rate HTTP download in the UoA DMZ (yellow). In this plot, only the yellow samples do not form a clear-cut sample cluster. However, they do if a different combination of information measures is considered, as in Experiment 10.3.3.

The SYN flood packet signature which would explain the slightly higher 1-gram Shannon entropy of the Windows Messenger spam samples. Compared to the SYN flood samples, LZ76 assigns a clearly higher complexity to the Windows Messenger spam samples. This may be explained with two observations:

- The packet rate of the Windows Messenger spam events is lower than that of the SYN flood. For this reason, sequences of Windows Messenger spam packets are more likely to be interspersed with packets belonging to regular traffic.

- Due to address scanning inherent in Windows Messenger spam, the destination IPv4 address usually changes with each Windows Messenger spam packet, which also has an effect on the IPv4 and UDP checksums. For this reason, LZ76 cannot exploit entire packet signatures in a single step. This leads to higher production step counts and thus to a higher LZ production complexity. For comparison: during the SYN flood attack, addresses, sequence numbers, and ports did not change, which meant that the IPv4 and TCP checksums were also all the same. As a consequence, the entire packet signature repeats frequently and LZ76 is able to exploit these (relatively long) signatures.

Apart from the Windows Messenger spam and the SYN flood, two more events form sample clusters which are distinguishable from the background entropy: the magenta sample cluster is related to low-rate Windows Messenger spam events (also visible in Figure 10.1, between the dark blue sample cluster and the background entropy sample cluster) and the black sample cluster appears to be related to sequences of TCP ACKs for e-mail traffic. Table 10.1 presents a number of packet properties of such
10.3 Traffic analysis with multiple measures

a sequence.

<table>
<thead>
<tr>
<th>Timestamp [s/µs]</th>
<th>Src socket</th>
<th>Dest socket</th>
<th>Protocol</th>
<th>IPv4 length</th>
<th>ACK flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>1121997744:511712</td>
<td>202.74.207:25:25</td>
<td>130.216.190.13:35419</td>
<td>TCP</td>
<td>52</td>
<td>True</td>
</tr>
<tr>
<td>1121997744:512471</td>
<td>202.74.207:25:25</td>
<td>130.216.190.13:35419</td>
<td>TCP</td>
<td>52</td>
<td>True</td>
</tr>
</tbody>
</table>

Table 10.1: The black samples in Figure 10.3 appear to be associated with sequences to SMTP TCP ACK packets as presented above. During the event, the IPv4 address 202.74.207.25 was never observed as destination. For this reason, this event may be backscatter of an attack against 202.74.207.25 which uses 130.216.190.13 as a forged source address.

The yellow samples form a sample cluster, but they are located within the background entropy and are thus not distinguishable. These samples are shown here mainly as a reference for Experiment 10.3.3, in which they form a distinct sample cluster. The property that characterises these samples is a high rate HTTP download between a DMZ host and a host inside the UoA network, a packet sequence of which is shown in Table 10.2.

<table>
<thead>
<tr>
<th>Timestamp [s/µs]</th>
<th>Src socket</th>
<th>Dest socket</th>
<th>Protocol</th>
<th>IPv4 length</th>
<th>ACK flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>1121999502:484774</td>
<td>130.216.1.27:80</td>
<td>130.216.191.183:35928</td>
<td>TCP</td>
<td>1500</td>
<td>True</td>
</tr>
<tr>
<td>1121999502:484897</td>
<td>130.216.1.27:80</td>
<td>130.216.191.183:35928</td>
<td>TCP</td>
<td>1500</td>
<td>True</td>
</tr>
<tr>
<td>1121999502:485020</td>
<td>130.216.1.27:80</td>
<td>130.216.191.183:35928</td>
<td>TCP</td>
<td>1500</td>
<td>True</td>
</tr>
<tr>
<td>1121999502:485143</td>
<td>130.216.1.27:80</td>
<td>130.216.191.183:35928</td>
<td>TCP</td>
<td>1500</td>
<td>True</td>
</tr>
<tr>
<td>1121999502:485266</td>
<td>130.216.1.27:80</td>
<td>130.216.191.183:35928</td>
<td>TCP</td>
<td>1500</td>
<td>True</td>
</tr>
<tr>
<td>1121999502:485389</td>
<td>130.216.1.27:80</td>
<td>130.216.191.183:35928</td>
<td>TCP</td>
<td>1500</td>
<td>True</td>
</tr>
<tr>
<td>1121999502:485512</td>
<td>130.216.1.27:80</td>
<td>130.216.191.183:35928</td>
<td>TCP</td>
<td>1500</td>
<td>True</td>
</tr>
</tbody>
</table>

Table 10.2: Properties of seven consecutive packets taken from a traffic sample that is related to a yellow entropy sample in Figure 10.3: although the payload (which is not listed here) changes for each packet, there are several protocol header fields with very stable contents during this event.

**Experiment 10.3.2 (1-gram Shannon entropy versus T-entropy)**

**Description:** This experiment plots 1-gram Shannon entropy against T-entropy. Experiments such as 4.2.1 suggest that T-entropy behaves similar to LZ76, especially for large \( N_S \), as is the case in the present series of experiments. In particular, T-entropy recognises multi-symbol patterns, like LZ76.

**Expectation:** One may expect the plot to look quite similar to that of the previous experiment, as the 1-gram Shannon entropy, which cannot recognise multi-symbol patterns is again plotted against a measure that recognises such patterns. This might result in a convex sample arrangement again.

**Observation:** Figure 10.4 largely confirms the expectation, as the transition from LZ76 to T-entropy on the horizontal axis preserved most of the features present in Figure 10.3.
Figure 10.4: Plot of the 1-gram Shannon entropy versus T-entropy: again, the overall arrangement of the samples is convex, as the 1-gram Shannon entropy tends to overestimate the actual entropy. Compared to LZ76, the location of the individual sample clusters is largely unchanged. Two exceptions are visible: the black samples associated with peculiar e-mail traffic do not form a coherent cluster with T-entropy, and T-entropy estimates the entropy of the yellow samples to be lower (with respect to the background entropy sample cluster) than LZ76 did. Yet the yellow samples are still not distinguishable from the background entropy.

As expected, the overall sample arrangement is still convex. However, three notable differences exist between Figures 10.3 and 10.4:

- The black samples associated with peculiar e-mail traffic no longer form a sample cluster, indicating that the sensitivity of T-entropy to this event is not stable.

- The yellow samples related to the DMZ HTTP download have shifted, but are still located inside the sample cluster belonging to the background entropy.

- The sample clusters related to the SYN flood and Windows Messenger spam appear to be closer.

Events such as the SYN flood (green), high packet rate Windows Messenger spam (blue), and low packet rate Windows Messenger spam (magenta) still form distinct sample clusters in the sample space. The convex arrangement of the samples in the plane is also preserved.

**Experiment 10.3.3 (T-entropy versus LZ production complexity)**

**Description:** This experiment plots T-entropy versus LZ production complexity. Experiments 4.2.1 and 9.2.1 already compared certain characteristics of the two measures, suggesting that they react similarly to identical input, especially for the relatively large $N_S$ used here. Apart from sample cluster observations, this experiment will enable comparison of entropy- and complexity-specific overestimation in both measures.
**Expectation:** In this experiment, both information measures are able to recognise multi-symbol patterns. Because of this, one might expect the overall sample arrangement to be more linear than it was in the previous experiments.

We also know from the previous two experiments that the black samples form a sample cluster with LZ76, but not with T-entropy. As this experiment uses the vertical axis for T-entropy, these samples should thus spread out vertically.

**Observation:** LZ76 and T-entropy indeed exhibit a relatively linear relationship for lower and higher entropies/complexities. Nevertheless, event specific sample clustering is observable in Figure 10.5.

![Figure 10.5: T-entropy plotted against LZ production complexity: both information measures are capable of recognising multi-byte patterns. For this reason, the overall sample arrangement stretches out largely along the diagonal. Nevertheless, overestimation effects of both measures lead to distinct sample clusters for Windows Messenger spam (blue/magenta), SYN flood (green), and DMZ HTTP download samples (yellow). With this combination of measures, the yellow samples form a distinct sample cluster for the first time in this series of experiments.](image-url)

As expected, the black samples do not form a sample cluster. However, for this combination of measures, the yellow samples associated with the DMZ HTTP download form a distinct sample cluster in the plot presented in Figure 10.5. Also, the SYN flood, the high packet rate Windows Messenger spam, and the low packet rate Windows Messenger spam samples still form distinct sample clusters. However, the sample cluster associated with low packet rate Windows Messenger spam is located closer to the background entropy in comparison to Experiments 10.3.1 and 10.3.2.

The SYN flood samples and the Windows Messenger spam samples still form two distinct sample clusters, suggesting that between T-entropy and LZ76, there is event-specific overestimation.
Experiment 10.3.4 (LZ78 versus T-entropy)

Description: This experiment plots the LZ78 complexity against the T-entropy. The number of dictionary entries in LZ78 may be interpreted as an approximation of LZ production complexity (cf. Section 2.2.4).

Expectation: LZ78 recognises multi-symbol patterns. Because of this one may expect the overall symbol arrangement to be roughly linear, like in the previous experiment. However, compared to LZ76, LZ78 overestimates by a larger amount, as discussed in Examples 2.2.1 and 2.2.3.

Observation: Similar to the previous experiment, the overall sample arrangement is rather linear, indicating that the overestimation for both measures is largely identical for lower and higher entropies/complexities. Note that the x-axis has shifted compared to Experiment 10.3.3, which is the result of the increased overestimation in LZ78. The smaller distance between the sample clusters associated with the Windows Messenger spam and SYN flood indicates that LZ78’s reaction to the patterns of both event types is more similar than it was with LZ76. The sample clusters associated with the events are still distinguishable, however.

![Graph showing LZ78 plotted against T-entropy](image)

Figure 10.6: LZ78 plotted against T-entropy: compared to LZ76, LZ78 tends to overestimate by a larger amount, resulting in a shift of the samples along the x-axis, compared to Figure 10.5. The overall sample arrangement is linear, because both measures recognise multi-byte patterns. LZ78 associates a larger range of complexities with the black samples than LZ76. For this combination of measures, the yellow samples are again not distinguishable from the background entropy. The samples associated with the Windows Messenger spam and the SYN flood still form clear-cut sample clusters.

LZ78 computes a variety of complexities for the black samples associated with the peculiar e-mail traffic. Because of this, these samples are also scattered horizontally in Figure 10.6. As in the first two experiments, the yellow sample cluster associated with the DMZ HTTP download falls into the sample cluster of the background entropy, preventing a detection of this event. The magenta sample cluster associated with low packet rate Windows Messenger spam is close to the background entropy cluster, but still clearly distinguishable.
LZ78 yields relatively high complexities for seven samples that appear just below the background entropy sample cluster in Figure 10.6. These samples did not stick out as clearly in any of the previous three experiments.

For all experiments discussed in this section, the SYN flood samples and the Windows Messenger spam samples formed distinct sample clusters in the plane. The black samples only form a cluster for the 1-gram Shannon entropy in combination with LZ76, indicating that a stable entropy/complexity arises for both of these measures from the associated patterns. The yellow samples always form a cluster, but this sample cluster usually appears inside the background entropy. Only if LZ76 is combined with T-entropy, this event is clearly distinguishable from the background entropy.

10.4 Concluding remarks

This section discussed network traffic analysis with multiple mappings and multiple information measures. In a multi-dimensional sample space, analysis with multiple mappings permits spatial discrimination of different events: mappings exhibit an event-specific response, which at least sometimes leads to distinct sample clusters in the sample space.

For multi-dimensional analysis with different measures, sample clusters were also observed. Here, the clusters arise from characteristic overestimation of the information content for the different measures.

For both methods, the spatial clustering of event samples may be helpful for event detection.
Conclusions

This thesis proposes the use of information measures for network event detection. While a variety of different information measure were presented in Chapter 2, this thesis is generally focused on a relatively new measure, T-entropy. This measure is appealing for network event detection for several reasons:

1. T-entropy has a close relationship with known entropies such as the Kolmogorov-Sinai entropy of the logistic map, as shown by Ebeling et al. [32].

2. The computational complexity of T-entropy is low. With \( f_{td} \), a workable \( O(n \log n) \) implementation of the algorithm is available, which terminates in \( O(n) \) time if the length of the inputs is bounded by a constant. In this thesis, the window size \( N_T \) provides a constant bound.

3. Compared to measures such as LZ production complexity and normalised complexity, T-entropy does not suffer from last step inaccuracies (cf. Section 9.2).
In particular Point 2 of this list makes T-entropy an interesting measure for network event detection: the amount of detection hardware required grows linearly with the sample size $N_S$. Rodeh et al. [87] claim that an equally quick implementation of LZ production complexity should be possible, but no such algorithm was available for experimentation at the time of writing. If such an implementation becomes available in the future, T-entropy might still produce more accurate results (Point 3). Due to their relative inaccuracy or high computational overhead, other measures such as LZ78 were mainly used for reference purposes in this thesis.

11.1 Results

A number of novel results were obtained in the course of this thesis project:

- A clear cause/effect relationship between network events and the T-entropy of affected samples was established in Chapter 4. Concerning the network events, LZ production complexity and normalised complexity showed a behaviour largely identical to T-entropy.

- Chapter 5 applied a signal-to-noise ratio to quantify the impact of network events on information measures. This measure was also used for comparisons between mappings and the optimisation of the sliding window size.

- The concept of packet mapping was introduced in Chapter 6, followed by a comprehensive information-theoretical review of the header fields in common network protocols. Furthermore, Chapter 6 introduced the concepts of high and low entropy monitoring to avoid the problem of entropy cancellation effects with multiple observables.

- The suitability of several sampling schemes for live interface monitoring and trace file analysis was reviewed in Chapter 7. In addition to this, Chapter 7 proposed a method to determine a suitable size for a sliding window associated with sampling, which depends on the duration of the shortest events one wishes to detect.

- T-entropy was used for the detection of several types of network events in Chapter 8. Note that each event type discussed here is also detectable with event-specific statistical methods. The particular strength of the information-theoretic approach over narrow statistical methods is its general nature, i.e., it is possible to detect a wide range of events without necessarily having to tweak detection parameters.
11.2 Open problems

- Chapter 9 reviewed the sensitivity of T-entropy for network events with a variety of different packet rates. For the first time, a direct comparison of the estimation errors (as a function of the input size) associated with different measures was made.

- Chapter 10 proposed the use of multiple mappings and multiple measures for improved detection and possibly classification of network events.

- A comprehensive set of software tools to analyse libpcap trace files was developed in the preparation of this thesis.

11.2 Open problems

- As with all computable information measures, T-entropy suffers from entropy estimation errors. This may obscure events with a small impact on the T-entropy.

- The network event detection approach discussed in this thesis suffers from false positives and false negatives. I.e., regular traffic might be mistaken for an event (consider flash crowds) while other traffic belonging to events may not be detected because it only causes a small deviation from the background entropy. Note that the latter case is rather unlikely for malicious events, because an attacker would need intimate knowledge about the background entropy of the monitored observables and mimic this entropy level in an attack.

- The traffic analysed in this thesis originates from an ordinary network card. Standard network cards do not provide hardware timestamps for incoming packets, therefore the timestamps have been added by the tcpdump software and may thus be inaccurate. For this reason, only basic interarrival time measurements were carried out in this thesis.

- The network considered in this thesis permitted processing of all packets on the monitored link. On backbone links it may only be possible to process a small fraction of the traffic.

11.3 Future work

The insights gained from this thesis may be used as the foundation for future projects. Some suggestions are provided below.

- In this thesis all packets on the monitored links were considered. On busy backbone links, it may not be possible to capture every packet. Instead, it may only be possible to capture one in 100 packets,
for instance. It would be interesting to examine the impact this has on network event detection with T-entropy, especially in the light of the publications by Brauckhoff, Tellenbach, Wagner, May, and Lakhina [22] and Mai, Chuah, Sridharan, Ye, and Zang [75], which question whether sampled data is sufficient for network event detection.

While incomplete data sampling would make flow analysis impossible, the approach presented in this thesis might still produce useful results, as it does not consider flows.

- Interarrival time measurements with highly accurate GPS timestamps provided by a DAG network monitoring card (see DAG project website [8] and Endace company website [2]) would be an interesting area for further research.

- The software developed during this thesis could be improved significantly. At this stage, mappings and packet signatures for artificial network event injection are hard-coded, for example. Apart from this, the software is not able to process trace file formats other than the `libpcap` format. WAND [10] published a general trace file API called `libtrace` [30]. The use of this API should permit processing of many more trace file formats, such as that produced by DAG cards.

Another commonly used software for traffic analysis is CoralReef [1] provided by the Cooperative Association for Internet Data Analysis (CAIDA). CoralReef provides many interesting data representation extensions and, like `libtrace`, it can process a variety of trace file formats. Hence, the software developed in the course of this thesis project may benefit from using functions that `libcoral`, the C API of CoralReef offers.

- Event classification in more than two dimensions using multiple mappings would also be an interesting area for future work.

Overall it can be said that information measurement of network streams, especially with T-entropy, is a viable method of network event detection. However, additional development work is necessary to implement a complete network event detector based on the findings in this thesis.

The detection technique discussed in the thesis can in principle be used in conjunction with existing solutions: For instance, signature-based intrusion detection systems could possibly limit their activity to times with unusual traffic entropy.
This chapter describes some details about the software and mappings used for this thesis. Readers intending to use the software or interested in mapping details may find this an interesting read. Readers interested in the theory of this project only may want to skip this chapter.

A.1 Software

This section briefly describes the software used and/or developed during this thesis project. The software was needed for trace file acquisition, trace file transformation, and trace file processing. The following three sections briefly discuss third party software used for this project. The remaining sections present a number of trace file processing tools developed by the author in the course of the project, listed in the order in which they were developed.
A.1.1 Third party software used for trace file capture and processing

tcpdump

For data acquisition tcpdump [7] was used. The name wrongly suggests that this program captures traffic at the transport layer; in fact, tcpdump records network streams at the data link layer. tcpdump is closely related to the libpcap project, a C library for packet capture\(^1\). tcpdump is able to generate libpcap trace files. This permits easy access to data captured by tcpdump via libpcap further in the process. Both tcpdump and libpcap are distributed as source code under the BSD license.

bzip2

Trace files captured during busy hours can easily become very large, even if only a small part of each packet is captured. There is usually much redundancy, such as similar IPv4 address prefixes in trace files, permitting compression ratios of approximately 1:3. In order to be capable of capturing and storing more trace files for this project, the author decided to compress the trace files with bzip2, an efficient text compression program. bzip2 was chosen, because in test runs it produced smaller output files than gzip, which would have been an alternative. bzip2, its counterpart bunzip2 and the corresponding C library are available freely as open source under a BSD license.

ftd

ftd is a program which computes the T-complexity, T-information, and T-entropy of a string passed to it. ftd was developed by Speidel and Yang [95] and is described in Section 2.2.5 on page 32. ftd is an improvement of tcalc which was developed by Titchener and Wackrow [107] and terminates in $O(n^2)$ time. Some of the programs described in the following section use functions from ftd for the computation of T-entropy.

A.1.2 Software developed in the course of the project

The programs described below were developed by the author of this thesis in the course of the project. The author plans to release these programs under the GNU General Public License after thesis completion.

\(^1\)In fact, both projects are hosted on the same website.
bpf\_slicer

Locating a particular trace file record in a non-indexed trace file may take a long time. A binary search for trace file is not easy to implement, as re-synchronisation with the stream would be required for each search step, due to the variable record length.

bpf\_slicer breaks a monolithic tcpdump trace file into a sequence of slice files (or simply slices). By default, each slice contains one minute of the original trace file’s data. After processing an input file, bpf\_slicer writes the following information to an slice index file:

- Total number of slices
- Number of the first trace file record in each slice
- Total number of packet records in each slice
- Timestamp of the first packet record in each slice

Programs such as bpf\_map (see Section A.1.2) or bpf\_dumpsample (see Section A.1.2) use this index to locate specific traffic samples quickly: the slice index file provides information permitting a binary search through the slices; once the slice containing a particular packet record is found, the requested packet record is searched in a linear fashion.

Synopsis:

\[ \text{bpf\_slicer [option]}... \text{trace\_file} \]

Options:

- \(-h\) Prints a help message.
- \(-t \text{ integer}\). Specifies slice duration (in \(\mu s\)); default is 60000000 (= 60 s).
- \(-v\) Prints version info and exits.

Example: bpf\_slicer \(-t\) 1000000 input.bpf.bz2 generates a slice index file and a set of slice files, covering 1 s worth of traffic each, from the input trace file input.bpf.bz2.

bpf\_map

Packet record mapping (cf. Chapter 6), sampling (cf. Chapter 7) and entropy/complexity computation are the main tasks performed by bpf\_map. bpf\_map takes a trace file (either sliced or unsliced) and
maps its packet records according to mapping rules (currently hard coded) into a sequence of symbols. Whenever sufficiently many symbols have been collected for a traffic sample, `bpf_map` computes the entropy/complexity of the sample using the preferred information measure and writes the result to an entropy/complexity sample file. The name of this file is automatically generated from the command line parameters.

Apart from mapping, sampling and entropy/complexity computation, `bpf_map` performs two additional tasks:

- Packet record filtering: users may filter certain packet records from the stream. `bpf_map` does not consider filtered packets in traffic samples. Filter rules are specified in plain ASCII files, referred to as filter rule files in this thesis. The structure of such filter rule files will be described in the next section.

- Packet record injection: artificial network events can be simulated by injecting additional packet records into the packet record stream provided by the input trace file. At the time of writing, two packet record types are available for injection: SQL Slammer and packet records identical to those of the SYN flood in \( T_{20050722} \).

Synopsis:

```
bpf_map [option]... trace_file
```

Options:

- `-a` Sets the information measure used for entropy/complexity computation. Accepted values are:
  - 6: LZ production complexity
  - 7: LZ77
  - 8: LZ78
  - s: 1-gram Shannon entropy
  - t: T-entropy (default setting)
- `-b` Sets the sampling method (cf. Chapter 7). Accepted values are:
  - c: Combo sampling (default setting)
  - i: Interval-based sampling
  - v: Volume-based sampling
- `-E integer`. Processing end Unix timestamp in (\( \mu s \)); default is \( 4.294967295 \times 10^{15} \).
- `-e integer`. Injection end timestamp in (\( \mu s \)); default is \( 2.5 \times 10^{9} \).
- `-F integer`. \( N_S \) (in symbols); default is 500.
- `-h` Prints a help message.
-i integer. Injection rate in packets/s; default is 0, i.e., no injection.

-m integer. Mapping Id; default is 0, i.e., no mapping.

-S integer. Processing start Unix timestamp (in µs); default is 0, i.e., midnight of 1st January 1970.

-s integer. Injection start timestamp in (µs); default is 1.5 × 10^9.

-T integer. Precede entropy/complexity values in the entropy/complexity sample file with timestamps.
   Accepted values are:
   a: Timestamp relative to first packet in trace file (default setting)
   e: Timestamp relative to midnight at the beginning of 1st January 1970
   r: Timestamp relative to first processed packet’s timestamp

-t integer. Sample interval τ (in µs); default is 1.0 × 10^5 µs = 100 ms.

-v Prints version info and exits.

-w integer. Window size for LZ77 (in bytes); default is 500 bytes.

-x string. Defines a filter rule file (cf. next section for details).

Example: bpf_map -at -bc -m27 -t100000 -F5000 -Ta input.bpf.bz2 computes the T-entropy of the input trace file input.bpf.bz2, using combo sampling with sliding window parameters N_S = 5000, τ = 100 µs, and mapping M_{27}. Each in the output file entropy samples is prefixed with a timestamp relative to the first packet in input.bpf.bz2.

bpf_dumpsample

bpf_dumpsample dumps information about the packets of specific traffic samples to stdout. One may use this program to investigate the specific conditions that lead to a particular entropy-/complexity sample.

For convenience, the program largely uses the same parameters as bpf_map. The version current at the time of writing permits users to dump any combination of the data listed under the -D option listed below.

Synopsis:

    bpf_dumpsample [option]... trace_file

Options:

    -a Sets the information measure used for entropy/complexity computation. Accepted values are:
      6: LZ production complexity
7: LZ77
8: LZ78
s: 1-gram Shannon entropy
t: T-entropy (default setting)

-b Sets the sampling method. Accepted values are:
c: Combo sampling (default setting)
i: Interval-based sampling
v: Volume-based sampling

-D string. Packet fields to be dumped; Accepted is any combination of the following:
n: Sample number/packet number (within sample)
t: Packet timestamp
i: Interarrival time
a: Source and destination IPv4 addresses
l: IPv4 length field
c: IPv4 TOS field
T: IPv4 TTL field
p: IPv4 protocol field
s: Symbol resulting from mapping (as hexdump)
P: Source and destination TCP/UDP ports
Y: TCP SYN flag
A: TCP ACK flag
w: TCP Advertised window field
d: Payload (as hexdump)
k: UDP checksum field

-e integer. Number of the last dumped traffic sample; default is 0.

-F integer. N_S (in symbols); default is 500.

-h Prints a help message.

-i integer. Injection rate in packets/s; default is 0, i.e., no injection.

-m integer. Mapping Id; default is 0, i.e., no mapping.

-s integer. Number of the first dumped traffic sample; default is 0.

-T integer. Precede entropy/complexity values in the output with timestamps.
Accepted values are:
a: Timestamp relative to first packet in trace file (default setting)
e: Timestamp relative to midnight at the beginning of 1st January 1970
r: Timestamp relative to first processed packet’s timestamp

-t integer. Sample interval $\tau$ (in $\mu$s); default is $1.0 \times 10^5 \mu s = 100 ms$.

-v Prints version info and exits.

-w integer. Window size for LZ77 (in bytes); default is 500 bytes.

-x string. Defines a filter rule file (cf. next section for details).
Example: **bpf_dumpsample** -s10 -e20 -DtaP -bc -F5000 input.bpf.bz2 dumps data from packets associated with traffic samples 10 to 20 from input trace file input.bpf.bz2 to stdout, using combo sampling with sliding window parameters $N_S = 5000$, $\tau = 100\text{ms}$. The data dumped is the timestamp, the source and destination IPv4 addresses, and the source/destination ports (where applicable).

**bpf_stats**

*bpf_stats* uses an entropy sample file generated by **bpf_map** as input and computes statistical data about it, such as the residual noise and the SNR. In order to separate event samples from non-event samples (or low values from high values), **bpf_stats** requires a discrimination threshold $b$ as additional input. **bpf_stats** provides four different methods to specify this discrimination threshold. If **bpf_stats** is called without any options for the specification of a discrimination threshold, only global statistical data such as global mean or global standard deviation are computed.

**Synopsis:**

```
bpf_stats [option]... trace_file
```

**Options:**

- `-f float` Fixed specification of the discrimination threshold.
- `-h` Prints a help message.
- `-p float` Used to specify the discrimination threshold in terms of a fixed percentage of low values.
- `-s float` Used to specify the discrimination threshold in terms of a fixed number of global standard deviations from the global mean.
- `-v integer` Used to specify the discrimination threshold in terms of a fixed number of low values.

Example: **bpf_stats** -v 100 entrsamples.txt will compute $b$ so that it discriminates exactly the 100 lowest values from all entropy samples appearing in the file entrsamples.txt.

**bpf_filter**

*bpf_filter* copies packets matching certain filter rules from an input trace file into an output trace file. The filter mechanism is identical to that of **bpf_map** and **bpf_dumpsample**. Details about it can be found in the next section.

**Synopsis:**

```
bpf_filter [option]... trace_file
```
Options:

- `-h` Prints a help message.
- `-n` Inverts the filter rules set in filter rule file.
- `-o string` Used to define the output file name.
- `-x string` Defines the filter rule file to be used (cf. next section for details).

Example: assume the filter rule file `smtpfilter.txt` contains the filter rules defined in the example of Section A.2. `bpf_filter -x smtpfilter.txt -o output.bpf.bz2 input.bpf.bz2` generates a `libpcap` trace file named `output.bpf.bz2` which only contains TCP packet records with source or destination port 25 from `input.bpf.bz2`. If the `-n` command line parameter is also used, `output.bpf.bz2` contains all packet records from `input.bpf.bz2 except` those with TCP source or destination port 25.

`bpf_fileinfo`

`bpf_fileinfo` is a C program that scans a trace file generated by `tcpdump` and computes statistical information about the file. In its current version, the program is able to provide the following information:

- Timestamp and human-readable date/time of the first trace file record
- Timestamp and human-readable date/time of the last trace file record
- Duration of the trace file
- Total number of captured packets
- Average packet rate
- $\tau$ (for a given window size $N_S$)
- Shortest time between two adjacent trace file records
- Longest time between two adjacent trace file records

**Synopsis:**

```
  bpf_fileinfo [option] trace_file...
```

**Options:**

- `-F integer` Used to provide $N_S$ to `bpf_fileinfo` for the computation of $\tau$.
- `-h` Prints a help message.

Example: `bpf_fileinfo -F 5000 input.bpf.bz2` computes the statistical data mentioned above for the trace file named `output.bpf.bz2`.
bpf_search

*bpf_search* may be used to scan a *libpcap* trace file for packets matching certain criteria. These criteria are expressed sequences of *terms*. Each term consists of a variable, a comparison operator, and an unsigned numerical value. The variables are used as references to specific fields in the packet records. At the time of writing, the following variables are supported:

- **IPVERS**: IPv4 version field
- **IPHLEN**: IPv4 header length field
- **IPTOS**: IPv4 type of service field
- **IPLEN**: IPv4 length field
- **IPIREDIT**: IPv4 ident field
- **IPFLAGS**: IPv4 flags field
- **IPOFFSET**: IPv4 offset field
- **IPTTL**: IPv4 time to live field
- **IPPROTOCOL**: IPv4 protocol field
- **IPCHECKSUM**: IPv4 checksum field
- **IPSRC**: IPv4 source address
- **IP2BSRC**: first two octets of the IPv4 source address
- **IPSRC2B**: last two octets of the IPv4 source address
- **IP2BDST**: first two octets of the IPv4 destination address
- **IPDST**: IPv4 destination address
- **IP2BDST2B**: last two octets of the IPv4 destination address
- **IP2BINT**: first two octets of the internal IPv4 address
- **IPINT**: internal IPv4 address
- **IP2BINT2B**: last two octets of the internal IPv4 address
- **IP2BEXT**: first two octets of the external IPv4 address
- **IPEXT**: external IPv4 address
- **IPEXT2B**: last two octets of the external IPv4 address
- **ADDR**: IPv4 source or destination address
- **TCPSRC**: TCP source port
- **TCPSRCDST**: TCP destination port
- **TCP2PORT**: TCP source or destination port
- **TCPSEQ**: TCP sequence number
- **TCPACKSEQ**: TCP acknowledgment number
- **TCPD0FF**: TCP data offset
• TCPCWR: TCP congestion window reduced flag
• TCPECE: TCP ECN ECHO flag
• TCPURG: TCP URGENT flag
• TPCACK: TCP ACK flag
• TCPPUSH: TCP PUSH flag
• TCPRST: TCP RESET flag
• TCPSYN: TCP SYN flag
• TCPFIN: TCP FIN flag
• TCPWIN: TCP advertised window field
• TCPCHECK: TCP checksum field
• TCPURG PTR: TCP urgent pointer field
• UDPSRC: UDP source port
• UDPDST: UDP destination port
• UDPPORT: UDP source or destination port
• UDPLEN: UDP length field
• UDPCHECK: UDP checksum field
• DPORT: TCP/UDP destination port field
• SPORT: TCP/UDP source port field
• PORT: TCP/UDP source or destination port field

Available operators for comparison are =, <, >, <=, >=, and !=. Except for =, their meaning is identical to the respective operators in the C programming language. The meaning of the = operator corresponds to == in C.

An example for a typical term passed to `bpf_search` is `IPPROTO=17`, which matches all UDP packets. The variables IPSRC, IPDST, IPINT, IPEXT, and ADDR also accept the "usual" decimal dotted notation of IPv4 addresses, i.e., the terms IPEXT=130.216.197.81 and IPEXT=2195244369 are identical, for instance.

For the definition of more complex matching criteria, several terms can be joined with an & sign. For example, `ADDR=130.216.197.81&TCPPORT>=22&TCPPORT<=80` matches all TCP packets from or to 130.216.197.81, which use a port in the range 22 to 80.

A selection of different screen dump formats is available for any matching packets.

Synopsis:

`bpf_search [option]... trace_file`
Options:

- **c string**: Used to define the matching criteria.
- **-E integer**: Processing end Unix timestamp in (µs); default is $4.294967295 \times 10^{15}$.
- **-h**: Prints a help message.
- **-i**: Inverts the filter rule set in filter rule file.
- **-o char**: Used to define the output format. Accepted values are
  - s: short dump. A single line is used for each packet (default setting).
  - v: verbose dump. A verbose packet dump using several lines is generated.
- **-S integer**: Processing start Unix timestamp (in µs); default is 0.
- **-x string**: Defines a filter rule file (cf. next section for details).

Example: `bpf_search -c "IPSRC=130.216.1.36&IPPROTO=17" input.bpf.bz2` dumps all UDP packets originating from host 130.216.1.36 in trace file `input.bpf.bz2` to stdout.

### tsh2bpf

Trace files from the NLANR Special Traces Archive [12] are stored in TSH files (Time Sequenced Header). This format has no file header to store a link type or time zone information. Instead, it consists of a sequence of 44 byte records, where each record is composed of the following fields:

- A 56 bit timestamp with a time resolution of 1 µs.
- A 8 bit interface number.
- The complete IPv4 header (20 bytes, without options).
- The first 16 bytes of the transport protocol header.

The trace file processing tools written for this project do not support the TSH format natively. In order to still be able to process NLANR TSH files for Section 8.4, the small C program `tsh2bpf` was implemented to convert TSH files into libpcap files. `tsh2bpf` fills any extra fields that the libpcap format provides over the TSH format with constant dummy values. Provided with a TSH file as argument, `tsh2bpf` generates an identically named libpcap trace file, with "\*.bpf" as file name extension.

**Synopsis:**

`tsh2bpf TSH_file`
A.2 Filter mechanism used in the software developed during this project

Some programs described in the previous section use filter rule files to filter certain packet types during trace file processing. The format of such files is briefly described here. Filter rule files are simple ASCII files. A simple example will be used to describe the format of filter rule files.

Consider the following filter rule file:

TCP 00 19
TCP XX XX 00 19

Each line in a filter rule file defines one filter rule, i.e., two filter rules are defined in the example above. Each filter rule consists of a filter type named at the beginning of the rule (TCP in the example) and a filter signature (XX XX 00 19 in the second filter rule of the example). The filter type and the filter signature must be separated by a single space character.

The filter type defines the position in the protocol stack where the filter is applied. Valid values are:

- **PCAP**: Applies the filter at the beginning of the PCAP packet header.
- **PCAP,PL**: Applies the filter at the first byte following the PCAP packet header.
- **ETH**: Applies the filter at the beginning of the Ethernet header.
- **ETH,PL**: Applies the filter at the first byte following the Ethernet header.
- **IP**: Applies the filter at the beginning of the IPv4 header.
- **IP,PL**: Applies the filter at the first byte following the IPv4 header.
- **TCP**: Applies the filter at the beginning of the TCP header.
- **TCP,PL**: Applies the filter at the first byte following the TCP header.
- **UDP**: Applies the filter at the beginning of the UDP header.
- **UDP,PL**: Applies the filter at the first byte following the UDP header.
- **PL**: Applies the filter at the first byte following any recognised headers.

Filter signatures consist of 8 bit hexadecimal words, also separated by single space characters. The special word **XX** is used as a wildcard: it matches any octet.

We can now interpret the filter in the example above: the first filter rule matches all TCP packets with a value of 25 (=0x0019, i.e., SMTP) as source port. The second rule matches all TCP packets with a value of 25 in the destination port field, not caring about the source port, because of the two wildcards.
Words in a filter signature may be preceded with a - (minus) sign, indicating that the corresponding word matches every value except the one stated. For instance -19 matches all octets except 0x19.

There is no limit to the number of filter rules in a filter rule file. Unix line breaks are used to separate filter rules, i.e., each filter rule is followed by a \texttt{0x0a} character. The number of line breaks must be identical to the number of filter rules.

### A.3 Details of mappings used in this thesis

The following table provides detailed information about the mappings used in this thesis.

<table>
<thead>
<tr>
<th>Mapping Id</th>
<th>Fields used</th>
<th>Symbol width $\omega$ (bits)</th>
<th>Used in Chapter/Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_5$</td>
<td>Complete IPv4 data, i.e. IPv4 header and its payload</td>
<td>$160 \leq \omega \leq 726$</td>
<td>4, 5.2, 7.5.1, 7.5.2, 10.3</td>
</tr>
<tr>
<td>$M_{13}$</td>
<td>Protocol field from IPv4 header</td>
<td>8</td>
<td>6.1, 8.4, 8.8</td>
</tr>
<tr>
<td>$M_{26}$</td>
<td>Truncated 8 bit sum of the payload (TCP or UDP payload where possible, IPv4 payload for other transport protocols)</td>
<td>8</td>
<td>6.3</td>
</tr>
<tr>
<td>$M_{27}$</td>
<td>Truncated 8 bit sum of the last two internal IPv4 octets, bits $2^{10}$ to $2^{3}$ of IPv4 length field, $M_{26}$</td>
<td>24</td>
<td>6.6, 7.5.2, 7.4, 8.4, 8.5, 9.1</td>
</tr>
<tr>
<td>$M_{28}$</td>
<td>Host identifier of internal UoA IPv4 addresses</td>
<td>8</td>
<td>6.5</td>
</tr>
<tr>
<td>$M_{31}$</td>
<td>TCP/UDP destination ports</td>
<td>16</td>
<td>10.2</td>
</tr>
<tr>
<td>$M_{37}$</td>
<td>Interarrival time intervals; for interval details see Table 6.1</td>
<td>8</td>
<td>6.4.1</td>
</tr>
<tr>
<td>$M_{40}$</td>
<td>TCP/UDP destination port</td>
<td>16</td>
<td>8.4</td>
</tr>
<tr>
<td>$M_{42}$</td>
<td>IPv4 length field intervals; for interval details see Figure 6.6</td>
<td>8</td>
<td>6.4.3</td>
</tr>
<tr>
<td>$M_{46}$</td>
<td>Third internal UoA IPv4 address octet</td>
<td>8</td>
<td>8.8</td>
</tr>
<tr>
<td>$M_{50}$</td>
<td>SYN flag from TCP header</td>
<td>8</td>
<td>8.6, 10.2</td>
</tr>
<tr>
<td>$M_{51}$</td>
<td>Last two octets of IPv4 source address</td>
<td>16</td>
<td>6.1</td>
</tr>
<tr>
<td>$M_{52}$</td>
<td>Last two octets of IPv4 source address and IPv4 protocol field</td>
<td>24</td>
<td>6.1</td>
</tr>
</tbody>
</table>

Table A.1: Mappings used in this thesis.


[86] Edward W. Ray: SQL Slammer and other UDP Port 1434 Threats. Technical report, SANS Institute, 2003. This article seems to have disappeared from the website of the SANS Institute (http://www.sans.org/) and may need to be requested from there, or from the author.


Index

1-gram Shannon entropy, 17, 146–150, 153

address scan, 51, 96, 125–127, 148

aireplay-ng, 88

AIX, 92

alphabet, 23, 29

artificial network event, 57, 65, 67, 68, 132, 136,
137, 140, 158, 162

background complexity, 62

background entropy, 61–63, 69, 74, 80, 84, 86, 90,
92, 95, 102, 105, 120, 121, 133, 134, 138,
144, 145, 147–153, 157

Boltzmann entropy, 35

Boltzmann, Ludwig, 34, 35

border gateway, 42–46, 51, 53, 55, 62, 65, 79, 85,
87, 92, 93, 96, 120, 121, 125, 127–129

bpf_dumpsample, 161, 163, 165

bpf_fileinfo, 166

bpf_filter, 165, 166

bpf_map, 161–163, 165

bpf_search, 167–169

bpf_slicer, 161

bpf_stats, 165

BSD, 58, 92, 93

license, 160

bunzip2, 160

bzip2, 20, 160

Chaitin, Gregory J., 16, 20, 22–24, 36

Clausius, Rudolf, 34

complexity, 14, 15, 17, 18, 22–24, 28, 29, 33, 36, 37,
41, 61–63, 68, 69, 72, 73, 138, 146–148,
150–153, 155, 161–164

measure, 3, 14, 17, 22, 23, 28, 29, 33, 36, 38,
40, 57, 73, 78

normalised, 36, 38, 61–63, 137, 139, 140, 155,
156

sample, 60, 61, 162, 163

computable complexity measure, 24

DAG card, 58, 86, 158

data link layer, 58, 59, 82, 104, 160

denial-of-service, 47

tack, 48, 89

low volume attack, 47, 48

description language, 23, 24

distributed Denial-of-Service

tack, 17, 18, 48, 49, 96, 121

Ebeling, Werner, 40, 155

Endace, 58, 158

dentropy, 3, 5, 15–20, 22, 33–37, 39–41, 46–49, 51, 52,
57, 59–65, 68, 69, 72, 73, 75, 77–83, 85–92,
94–108, 110, 112–115, 117, 118, 120–125,
127–138, 140, 144–147, 150–153, 155–158,
161–164

tempirical, 17–19

estimation error, 72, 78, 106, 109, 110, 113, 157

timator, 36, 62, 140

183
measure, 14, 16, 17, 19, 33, 34, 40, 46, 58, 59, 72, 73, 82, 84, 88, 98, 99, 106, 113, 119, 132, 134, 146
rate, 22, 36, 37, 39
sample, 16, 60, 61, 63, 73–75, 106, 108, 109, 112, 139, 144, 145, 149, 163, 165
file, 165
ESP, 94, 131, 133
Ethernet, 44, 58, 59, 87, 88, 90, 104
header, 58, 59, 87, 170
event, see network event
event sample, 73, 74, 115–117, 145, 165
filter rule, 162, 165, 166, 169–171
file, 162–164, 166, 169–171
ftd, 26, 32, 39, 141, 155, 160
GNU GPL, 160
Gulliver, Aaron, 37
gzip, 20, 28, 160
High entropy monitoring, 47, 73, 80, 81, 83, 88, 90, 92, 99, 100, 102–105, 122
high entropy monitoring, 78, 97, 98, 156
HOPOPT, 94
ICMP, 54, 92, 94, 96, 122, 124, 128, 129, 133
header, 54
ICMP flood, 48, 54, 85, 128, 129
IGMP, 133
information, 3, 14–16, 19, 21, 22, 35, 48, 71, 72, 100, 146, 147, 153, 156
rate, 47, 51, 53–55
source, 3
theory, 22, 34
interarrival time, 58, 66, 85–87, 157, 158, 164, 171
interdeparture time, 66
intrusion detection system, 13, 19, 158
IP header, see IPv4 header
header, 54, 58, 59, 78, 84, 88, 92, 95, 96, 169–171
IPv6, 88, 89, 94
Kolmogorov, 3, 16, 17, 20, 22–24, 36, 40, 155
Kolmogorov-Chaitin complexity, 16, 20, 22–24, 36
Lempel, Abraham, 3, 16, 22, 24–29, 36, 37
libpcap, 58, 59, 85, 86, 157, 158, 160, 166, 167, 169
Linux, 52, 58, 85, 88, 89, 91–95, 127, 141
live interface monitoring, 109, 110, 156
low entropy monitoring, 47, 51, 73, 78, 80, 81, 88, 89, 91, 92, 94, 97–99, 102, 156
LZ production complexity, 17, 22, 24–26, 28, 36, 40, 61–63, 137–141, 146–153, 155, 156, 162, 163
LZ76, see LZ production complexity
LZ77, 25–28, 37, 162–164
LZ78, 20, 27, 28, 37, 40, 73, 139, 140, 146, 152, 153, 156, 162, 164
MacOSX, 91
mapping, see packet mapping
N-gram entropy, 34–36, 39
detection, 3, 5, 14, 15, 17, 19, 20, 85, 155–158


network layer, 58, 59, 63, 68, 87

network stream, 3, 14–17, 41, 42, 53, 54, 57, 59, 60, 78, 80, 100, 106, 122, 126, 129, 130, 134, 158, 160

network traffic, 13, 16, 42, 57, 58, 80, 145, 153

capture, 57, 58

Nicolescu, Radu, 31, 37

NLANR (U.S. National Laboratory for Applied Network Research), 122, 123, 169

noise, 16, 71–73, 75, 84, 105, 114–116, 118, 139, 140

residual, 16, 74, 75, 84, 105, 128, 132, 133, 137–141, 165
types of, 71

non-event sample, 73, 74, 115, 165

OPSFIGP, 133


complex, 84, 102, 104, 105, 116, 117, 128

fixed volume, 79, 80, 82, 106, 139

identification number, 83, 120, 163, 164, 171

interval, 85–87, 91

multi-byte, 102, 103, 105, 126, 129

multiple, 143–146

simple, 81–85, 90, 100–102, 120, 121

variable volume, 79, 82, 116

PCAP packet header, 58, 59, 85, 170

Pesin, Yakov, 40

PIM, 94, 131, 133

ping, 54

port scan, 51, 60, 96, 145, 146

prefix-free code, 29

RSVP, 94

sample cluster, 18, 144–153

sampling, see traffic sampling, 18, 20, 61, 108–111, 113, 118, 147, 156, 158, 161, 162, 164

SCTP, 94

Shannon entropy, 16–19, 34–38, 47, 146–150, 153, 162, 164

Shannon, Claude E., 3, 22, 34–37, 39

signal-to-noise ratio, 15, 16, 73–75, 79, 80, 84, 102, 105, 106, 114–118, 121, 128, 132–134, 136–139, 141, 156, 165

Sinai, Yakov Grigor’evich, 17, 40, 155

slice, 161

file, 161

index file, 161

Snort, 13

Solaris, 91, 95

Speidel, Ulrich M., 5, 23, 32, 37, 52, 85, 160

SQL Slammer worm, 46, 47, 52, 59, 60, 94, 100, 102, 119, 122–125, 162

dictionary attack, 48, 52, 53, 99, 125–128

Staiger, Ludwig, 37

statistical mechanics, 34

strings

concatenation of, 23

finite, 23

substring, 23

Sun OS, 92

symbol, 79, 80, 82, 84–87, 91, 104, 107, 108, 116–118, 120, 121, 127, 147, 149, 151, 152, 162,

Unix, 20, 52, 93, 98, 127, 141, 162, 163, 169, 171

von Neumann, John, 22

Wackrow, Scott, 32, 160

Windows, 91–93, 96, 122, 127


Yang, Jia, 32, 160

zip, 28

Ziv, Jacob, 3, 16, 22, 24–29, 36, 37