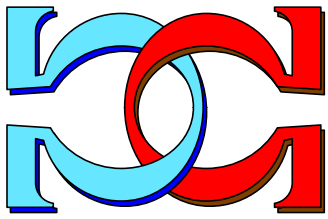
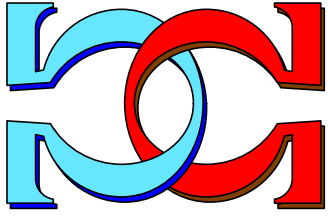
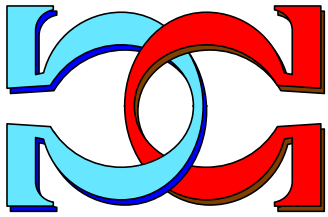


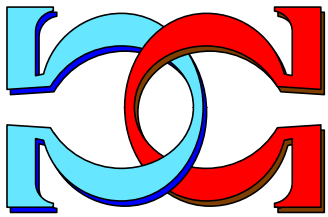
**CDMTCS
Research
Report
Series**



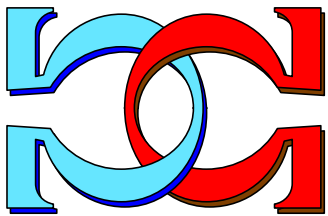
**Network Event Detection
with T-Entropy**



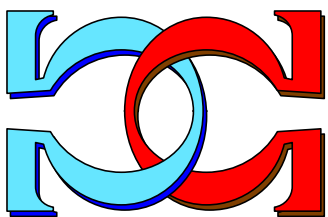
**Raimund Eimann, Ulrich Speidel,
Nevil Brownlee, Jia Yang**



Department of Computer Science
University of Auckland
Auckland, New Zealand



CDMTCS-266
May 2005



Centre for Discrete Mathematics and
Theoretical Computer Science

Network Event Detection with T-Entropy

Raimund Eimann, Ulrich Speidel, Nevil Brownlee, Jia Yang

December 2004

Abstract

This paper describes an entropy-based approach for the detection of network events. This is achieved by first converting a stream of network packets into a string and then computing its approximate average entropy rate using a computable complexity measure. Changes in the average entropy rate are interpreted as events. The computational complexity of the presented approach is nearly linear which makes this technique suitable for online scenarios. We present the results of several measurements on actual network data and show that it is indeed possible to associate actual network events with changes in entropy.

Providing a reliable network service to users is important to many Internet businesses and organizations. This requires a facility for the early detection of network events such as DDoS attacks [9] so that countermeasures can be taken. Network event detection is not a simple task on machines that are not immediately involved in the event. For instance, it may be difficult for the border gateways of a network to detect an attack against a machine inside the network.

Kulkarni, Bush and Evans [8] proposed an approach to network event detection by measuring the complexity of stream data. The authors themselves note that “*good estimation of Kolmogorov complexity is key to its usefulness*”. They further observe that the “*... complexity estimation technique used [in their paper] is not the best because empirical entropy is actually a very poor method of complexity estimation.*” Kolmogorov’s [7] observation that the complexity of a string is not computable presents a major obstacle for this type of approach. This implies that computable complexity measures can only ever be an approximation to Kolmogorov complexity. As a consequence, the question arises as to whether a particular approximation to Kolmogorov complexity provides adequate properties for network event detection. This paper proposes the use of T-entropy [4], a measure derived from a recursive parsing of strings. T-entropy has been shown experimentally to have a close correspondence with known physical entropies while its computational complexity is relatively low. We discuss its merits in network event detection.

A key component of our approach is the simplification of stream data by mapping the stream’s IP packets into symbols. This results in a string of characters whose complexity is related to the complexity of the original packet stream.

T-decomposition, T-complexity, and T-entropy

T-complexity is a complexity measure proposed by Titchener [10, 12, 11] and has also been discussed in a paper by one of the authors [5]. One fundamental property that makes T-complexity attractive for us is its computability. T-complexity serves as the basis for two further measures, T-information and its gradient, the T-entropy. Ebeling, Steuer and Titchener [4] showed that T-entropy exhibits a close relationship with known entropies such as the Kolmogorov-Sinai entropy of the logistic map. It has hence been conjectured that T-complexity represents a good approximation to Kolmogorov complexity.

The T-complexity of a string can be computed via an algorithm called T-decomposition. T-decomposition will be explained in the following paragraphs, followed by a discussion of T-complexity, T-information, and T-entropy.

Let $A = \{a_1, a_2, a_3, \dots, a_{n-1}, a_n\}$ denote a finite alphabet. We denote the cardinality of A by $\#A$, thus $\#A = n$. Elements $a_i \in A$ are called characters. Let A^* be the set of finite strings that can be generated by concatenating characters from A . $\lambda \in A^*$ denotes the empty string. Let $A^+ = A^* \setminus \{\lambda\}$. For two strings $x, y \in A^*$, let xy denote the concatenation of x and y . We further use x^k to denote the concatenation of k copies of x and $|x|$ to denote the length of x measured in characters.

The T-decomposition of $x \in A^+$ is carried out as follows.

1. Initialize a counter $m = 1$ and an array $k = []$.
2. Parse x left-to-right over A . Thus each character is parsed into a *token*.
3. Identify the last (rightmost) token in the current parsing of x . We will call this last token a .
4. Identify p_m as the penultimate token (the token immediately to the left of a). Identify the maximum length k_m (in tokens) of the run $p_m^{k_m}$ of $k_m \geq 1$ tokens that are copies of p_m which ends in the penultimate token. Store k_m in $k[m]$.
5. If the run identified in the previous step starts with the leftmost token, go to step 8.
6. Parse x left-to-right into a new set of tokens using the following rule: Existing consecutive tokens are merged into a single token if they form the following concatenations:
 - (a) $p_m^\ell q$, where $q \neq p_m$ and $1 \leq \ell \leq k_m$; or
 - (b) $p_m^{k_m+1}$.
7. Increment m and go to step 4.
8. Return (m, k) .

The values returned by the T-decomposition procedure are all we need to calculate the T-complexity of x . It is defined as follows:

$$C_T(x) = \sum_{i=1}^m \log_2(k_i + 1). \quad (1)$$

Example: Determine the T-complexity of the binary ($A = \{0, 1\}$) string $x = 11101010$. Dots are used as delimiters between tokens.

Initially we have $m = 1$ and $k = []$ in step 1. We parse in step 2 and obtain x as a series of single character tokens: 1.1.1.0.1.0.1.0. In steps 3 and 4 we determine $a = 0$, $p_1 = 1$ and $k[1] = 1$. The condition in step 5 is not met at this stage, so we proceed with step 6. Merging the tokens according to the rules in step 6, we obtain a new parsing into tokens for x : 11.10.10.10. In step 7, we set $m = 2$ and proceed with step 4.

In step 4 we determine $p_2 = 10$ and $k[2] = 2$. The condition in step 5 is still not met at this stage, so we proceed with step 6. Merging the tokens according to the rules in step 6, we obtain a new series of tokens for x : 11.101010. In step 7 we set $m = 3$ and proceed with step 4.

In step 4 we determine $p_2 = 11$ and $k[3] = 1$. The condition in step 5 is now met, so we proceed with step 8 where we return the result.

We have thus retrieved $k = [1, 2, 1]$ and $m = 3$ as the result of the T-decomposition of x . We may now use it to calculate the T-complexity of x :

$$\begin{aligned} C_T(11101010) &= \sum_{i=1}^3 \log_2(k[i] + 1) \\ &= \log_2(2 * 3 * 2) \approx 3.5849 \end{aligned}$$

A closed form for an upper bound for the maximum T-complexity of strings x of length $|x|$ has not been formulated yet, but Titchener conjectured that the logarithmic integral represents a good approximation of the asymptotic upper bound [10, 12, 11]. $C_T(x)$ is not a linear function. Therefore, one may define $I_T(x)$, the T-information of x , as a more linear measure for the amount of information carried in a string x . This “linearization” is achieved by using the inverse logarithmic integral:

$$I_T(x) = li^{-1}\left(\frac{C_T(x)}{\ln \#A}\right) \quad (2)$$

T-information is measured in *nats*, i.e., natural information units (base e). The gradient of the T-information is called T-entropy and is measured in nats per symbol:

$$H_T(x) = \frac{I_T(x)}{|x|} \quad (3)$$

In other words: T-entropy is a measure of the change of information as $|x|$ increases. Unlike other entropy measures, T-entropy is not normalized to 1 and values larger than 1 are common. We used T-entropy as our measure in a number of tests that are described in the following section.

Network Event Detection with T-Entropy

In order to perform event detection, we needed data about the network traffic at our monitoring site. This data was captured by a DAG [1, 3] card on a monitoring machine. DAG cards usually have two network interfaces, so that network traffic can be directed through them. For reasons of privacy and volume, a DAG card only produces a fingerprint of each packet that travels through it by capturing the first n bytes of the packet. Typical values are $n = 64$ and $n = 84$. Thus no application layer payload is recorded. The packet fingerprints contain, among other information:

- precise timing information,
- the link layer packet header,
- the complete IP header, and
- the transport protocol header or a part of it, dependent on the transport protocol used. The most common transport protocols, TCP and UDP, are fully covered if no IP or TCP options are present.

The data produced by the DAG cards can be used in two different ways:

- It can be stored as *tracefile records*. Tracefiles are frequently used in the development of analysis software as they permit software fine-tuning by replay of interesting scenarios.
- Real-time analysis of data retrieved from a live interface. Network event detection software runs on live interfaces.

In our experiments, we used tracefiles in order to be able to replay scenarios. Each record from a tracefile was mapped into a single *symbol* according to selected properties of the IP and/or TCP/UDP headers. We used simple ASCII characters as symbols. Higher symbol resolutions may require multi-byte symbols.

Example: Table 1 shows a simple mapping based on the *protocol* field from the IP header:

IP protocol ID	Symbol
0x01 (ICMP)	'i'
0x04 (IP)	'I'
0x06 (TCP)	't'
0x11 (UDP)	'u'
0x29 (IPv6)	'6'
0x84 (SCTP)	's'
other	'.'

Table 1: A simple mapping for IP packets: packets are mapped to symbols according to the packet's *protocol* header field.

Mapping tracefile records into symbols rather than using their plain data content has two advantages for our experiments:

- It is possible to focus on selected properties of a record. Other information carried in the record can be stripped away. Mapping hence permits viewing the data stream from different angles.
- Mapping records into symbols reduces the amount of input data by a factor $f = \frac{RecordSize}{SymbolSize}$. As T-decomposition requires almost linear time to complete, the speed-up factor is almost equal to f .
- Redundant information is removed, e.g., the network portion of the destination IP addresses of inbound packets.

After mapping, the symbols were concatenated into a string in order of packet arrival time. This string was then broken up into *fragments* of equal size. The average T-entropy was determined for each fragment. We call the value of a fragment's average T-entropy a *sample*. We chose the size of the fragments to be 500 symbols. This number was chosen under two considerations:

- It is small enough to provide for several thousand samples per tracefile.
- It is large enough not to cause excessive jumps between samples because of too low granularity.

This produced a series of samples that permitted observation of the average T-entropy.

The mapping part of our approach requires an algorithm with linear time complexity. The subsequent T-decomposition step is performed by an algorithm that was developed by Yang and Speidel [6]. This algorithm runs in order $O(n \log n)$ time. The $\log n$ component of this upper bound is due to the addressing scheme used by the algorithm. In real implementations with limited n this component can be neglected. In our approach n is constant. Thus the overall computational complexity of approach is linear and therefore suitable for online scenarios.

Experimental Results

For our experiments we needed input data. The data for our tests originates from DAG tracefiles recorded at the University of Auckland in December 2003 by Jörg Micheel from NLNR [2]. Each tracefile reflects one hour of network traffic at the monitoring point.

We used four different mappings to convert tracefile records into symbols:

- *Size mapping*: packets mapped into 43 symbols according to overall IP packet size.
- *Protocol mapping*: packets mapped into 10 symbols according to selected transport protocols.
- *Flag mapping*: packets mapped according to the three flag bits from the IP header, resulting in 8 symbols.
- *Port mapping*: packets mapped into 10 symbols according to TCP ports (only for TCP packets, the smaller of *source-* and *destination port*).

We present the results of our experiments as graphs. Where appropriate, we also provide alternative statistical data from our tracefiles.

The first tracefile we analyzed, T_1 , consists of 2022922 records. It contains three events we were interested in. In Figure 1 we present a graph of T_1 's packet rate to provide some background for the entropy computations carried out with this file. The three events are marked as A, B and C in the graph:

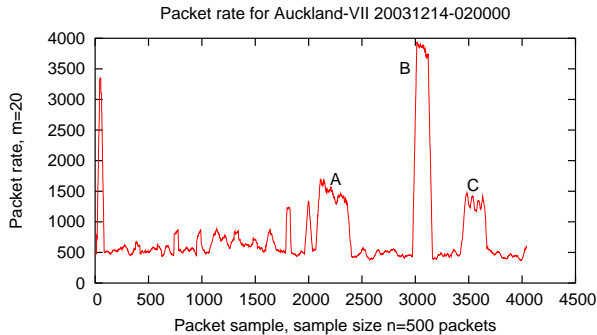


Figure 1: The packet rate of tracefile T_1 .

A closer look at the graph of Figure 1 and at the raw data of T_1 revealed that the “normal” packet rate was about 500 packets per second. It also revealed more details about the events A, B, and C:

- Event A is caused by a burst of medium-sized ($psize = 1076^1$) IP packets with a packet rate of about 1500 *packets/s*. TCP was used as transport protocol. The source and destination ports used were 34134 and 59863 respectively.
- Event B is caused by a sudden burst of small ($psize = 40$) IP packets with a packet rate of ≈ 3800 *packets/s*. A more detailed analysis of T_1 showed that this event was a scan on port 9999.
- Event C is caused by a burst of small ($psize = 48$) IP packets with a packet rate of about 1300 *packets/s*. For this event the in-depth analysis of T_1 also revealed scanning activity, this time on port 445.

Our first T-entropy measurement with T_1 used size mapping. The result of this measurement is presented in Figure 2. Note that the horizontal axis states the sample number, not the time.

The events A, B, and C from Figure 1 are clearly visible in Figure 2 as drops in the average T-entropy. The drop in T-entropy does not come as a surprise: As we have already mentioned, the respective sections of the tracefile contain a large number of very similar packets - a scenario that one would classically associate with low entropy.

Protocol, port, and flag mappings for T_1 are shown below in Figure 3, 4, and 5.

Again, the events A, B, and C are noticeable in Figure 3 to 5 as drops in the respective T-entropies. Our other experiments confirm that T-entropy measurements of a single tracefile

¹All packet sizes are given in bytes.

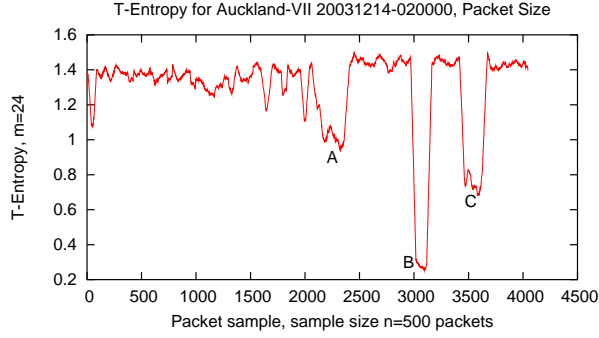


Figure 2: T-entropy graph of the size-mapped tracefile T_1 . The events A, B, and C from Figure 1 are visible.

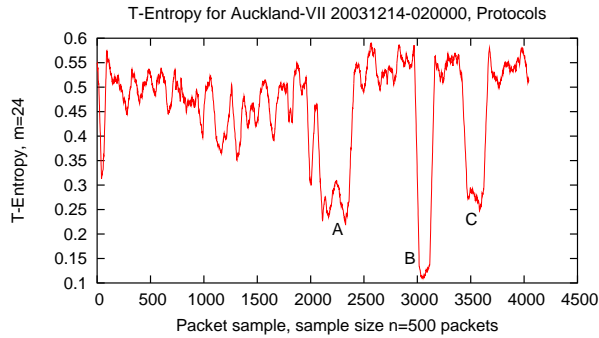


Figure 3: T-entropy graph of the protocol-mapped tracefile T_1 . Again, the three events are visible and marked as A, B, and C.

often exhibit similar characteristics, even for different mappings. This is consistent with the observation that there is significant correlation between different packet properties. For example, packets relating to a particular application may not only use the same ports, but may also be similar in size. Network events that involve packets sent by similar source and destination applications may thus be detectable via the mapping of different properties.

In order to determine the average non-event T-entropy \overline{H}_T for each mapping, we computed the histogram of the samples and determined \overline{H}_T as the T-entropy value of the largest peak of the histogram. The results of these calculations are presented in Table 2. For comparison, the table also shows average T-entropies \overline{H}_R taken from random sources with the same number of symbols. We may thus observe that the average T-entropies measured in the tracefiles were all reasonably low already. Given the generally highly correlated nature of network traffic, this had to be expected.

In the following paragraphs we present the results of four more experiments with different tracefiles, $T_2 - T_5$. These files contain 2513861 (T_2), 11357132 (T_3), 2029780 (T_4), and 4005695 (T_5) records respectively. A short description of the cause of the event(s) is given for each graph.

The graph in Figure 6 shows an event between samples 2000 and 3100 in tracefile T_2 . The graph was generated by size mapping. A closer inspection of the tracefile data revealed

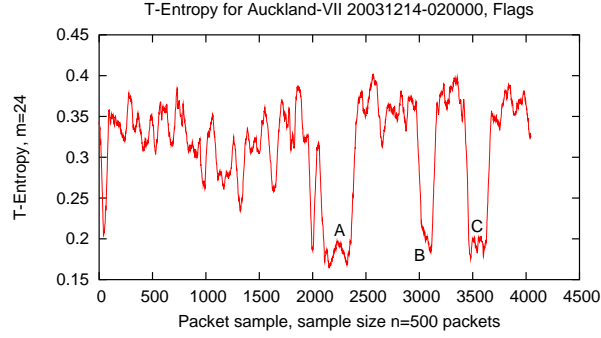


Figure 4: T-entropy graph of the flag-mapped tracefile T_1 .

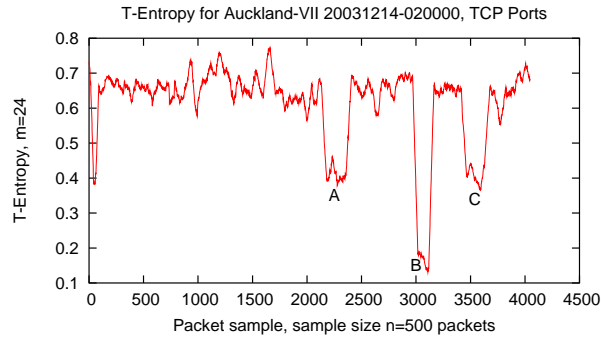


Figure 5: T-entropy graph of the port-mapped tracefile T_1 .

that two consecutive events were responsible for the drop in T-entropy:

1. An FTP transfer which caused the packet rate to leap from ≈ 500 *packets/s* to ≈ 2300 *packets/s* between samples 2000 and 2900. Most of the packets involved had a size of 1500 bytes.
2. An unspecified network event using source and destination TCP ports 36640 and 41320, approx. between samples 2900 and 3050 at a packet rate of about 1500 *packets/s*. Most of the packets involved had a size between $1001 \leq psize \leq 1200$ bytes and were mapped into a single symbol.

Mapping	symbols	$\overline{H_T}$	$\overline{H_R}$
Flag	8	0.281	1.630
Port	10	0.559	1.792
Protocol	10	0.209	1.792
Size	43	1.422	2.720

Table 2: Characteristic values for the symbol mappings used: number of symbols, most predominant T-entropy value $\overline{H_T}$ and a reference value $\overline{H_R}$ for the T-entropy of random strings with the same number of symbols.

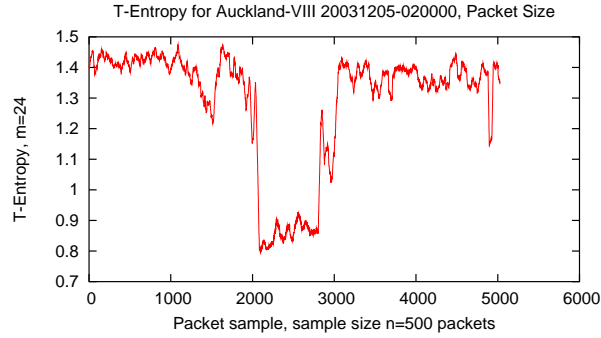


Figure 6: Two consecutive network events in tracefile T_2 between samples 2000 and 3050: The T-entropy of T_2 's size-mapped samples dropped by up to 40% compared to $\overline{H_T}$.

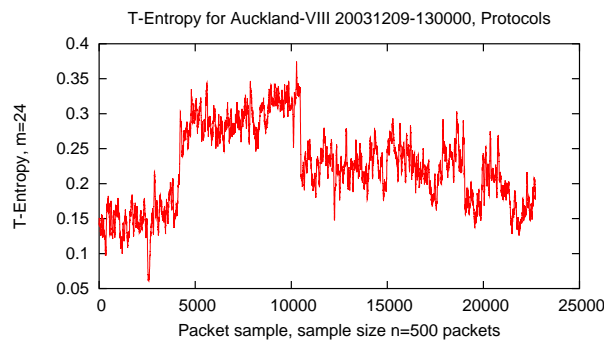


Figure 7: Events are not necessarily marked by a drop in T-entropy: This graph of tracefile T_3 shows a scenario where the T-entropy of the protocol-mapped samples rose by approx. 50% due to a temporary increase in the UDP packet rate between samples 4500 and 10500.

Figure 7 shows a graph of the T-entropy of the protocol-mapped tracefile T_3 . A rise in T-entropy by about 50% occurs between samples 4500 and 10500. This increase is caused by a more complex arrangement of the records in T_3 with respect to their transport protocol field. An examination of T_3 revealed that the affected fragments contain additional UDP symbols at a rate of approx. 250 *packets/s*. These symbols originate from UDP records using port 16384 – presumably VoIP traffic. Figure 8 shows the fraction of UDP packets on the link at the time of the event in percent. The rise in UDP traffic between 4500 and 10500 seconds is clearly visible.

The event presented in the graph of Figure 9 was most likely a port scan on machines inside the network of the University of Auckland. It involved thousands of TCP SYN packets attempting to open connections to port 4000 of various destination machines inside the university network. The following excerpts of nine consecutive records² from tracefile T_4 show that the destination IPs were scanned in almost sequential order. The out-of-sequence arrivals may have been caused by router queues or different network paths:

12.34.56.78:3311 → 130.216.33.60:4000

²Source IP address anonymized.

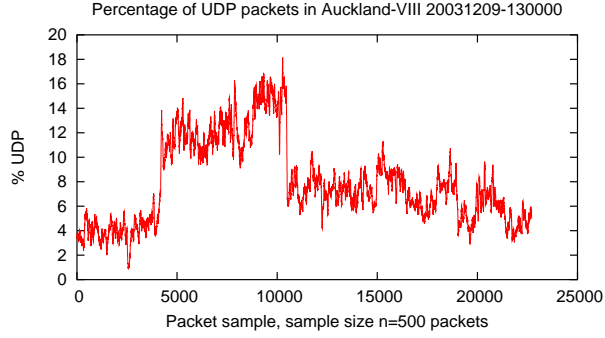


Figure 8: UDP Packet rate of T_3 in %.

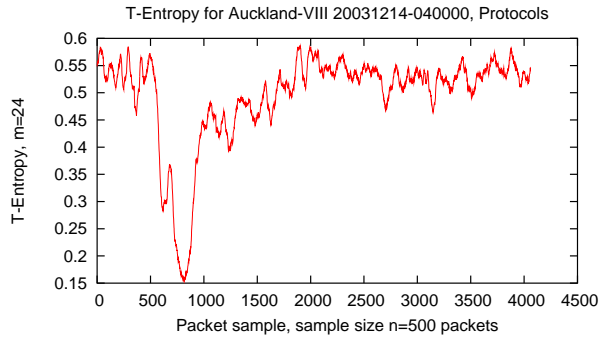


Figure 9: A port scan in tracefile T_4 .

```

12.34.56.78:3308 → 130.216.33.57:4000
12.34.56.78:3309 → 130.216.33.58:4000
12.34.56.78:3310 → 130.216.33.59:4000
12.34.56.78:3310 → 130.216.33.59:4000
12.34.56.78:3312 → 130.216.33.61:4000
12.34.56.78:3314 → 130.216.33.63:4000
12.34.56.78:3315 → 130.216.33.64:4000
12.34.56.78:3313 → 130.216.33.62:4000

```

This port scan caused the T-entropy to drop from $\overline{H_T} = 0.517$ to about 0.16, constituting a drop of 69%.

Figure 10 also shows a graph of a port scan in tracefile T_5 . An investigation of T_5 showed a similar situation as we found for the event presented in Figure 9:

```

87.65.43.21:4707 → 130.216.195.108:4899
87.65.43.21:4705 → 130.216.195.107:4899
87.65.43.21:4708 → 130.216.195.109:4899
87.65.43.21:4709 → 130.216.195.110:4899
87.65.43.21:4710 → 130.216.195.111:4899
87.65.43.21:4712 → 130.216.195.112:4899

```

While this port scan only had a moderate effect on the link's packet rate, the T-entropy dropped by about 55% from $\overline{H_T} = 0.256$ to 0.115.

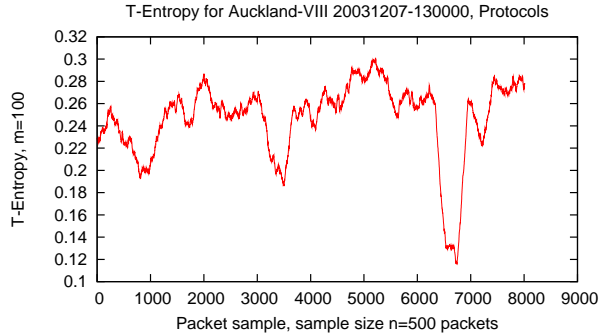


Figure 10: A port scan between samples 6300 and 7000 in tracefile T_5 .

Conclusions

Our experiments show that T-entropy is a sensitive measure for the detection of sudden changes in network traffic patterns. They demonstrate that various network events have an impact on T-entropy, even if only very simple mappings are used. The measurements in general confirm that network traffic has a low entropy. Events such as port scans tend to cause a drop in T-entropy. However, as demonstrated by the UDP example, this is not universally so - events may also result in an increased T-entropy. The average T-entropy $\overline{H_T}$ of “normal” traffic depends on the observation point, on the mapping used, and may also change with the time of day, week, etc.

Event detection by T-entropy offers two distinct advantages: Firstly, it permits events affecting individual hosts inside a network to be detected at the gateway rather than at the victim host, which may be overloaded or disconnected. Secondly, T-entropy detection is able to look at arbitrarily many packet parameters simultaneously. It thus offers an opportunity for detecting events about whose characteristics one has only a limited a priori knowledge, reflected by the chosen symbol mapping.

The data we had available for our experiments did not contain actual attacks. We expect attacks to have an even stronger impact on the T-entropy than the port scan experiments we presented here.

Our future work will focus on the detection of network events that are difficult to detect statistically. We also aim to develop more complex and powerful mappings. The eventual goal is the development of a monitoring application that will automatically detect network events and raise corresponding alarms.

Acknowledgements

Our thanks go to Mark Titchener for his comments and suggestions, to Jörg Micheel for making the tracefiles available, and to Jamie Curtis for his assistance with the tracefile format.

References

- [1] <http://dag.cs.waikato.ac.nz/> (last visited on June 29th, 2004).
- [2] <http://pma.nlanr.net/Special> (last visited on January 20th, 2005).
- [3] <http://www.wand.net.nz/> (last visited on June 29th, 2004).
- [4] W. Ebeling, R. Steuer, and M. R. Titchener. Partition-based Entropies of Deterministic and Stochastic Maps. *Stochastics and Dynamics*, 1:45 – 61, March 2001.
- [5] Ulrich Günther. T-Complexity and T-Information Theory – an Executive Summary. In *CDMTCS Research Report No. 149*, February 2001.
- [6] Jia Yang and Ulrich Speidel. A T-decomposition Algorithm with $O(n \log n)$ Time and Space Complexity. Technical report, Centre for Discrete Mathematics and Theoretical Computer Science (CDMTCS) of The University of Auckland, 2005.
- [7] A. N. Kolmogorov. Three Approaches for Defining the Concept of “Information Quantity”. *Problems of Information Transmission*, 1:3–11, 1965.
- [8] A. B. Kulkarni, S. F. Bush, and S. C. Evans. Detecting Distributed Denial-of-Service Attacks Using Kolmogorov Complexity Metrics. In *Technical Information Series*. GE Development & Research Center, February 2002.
- [9] Felix Lau, Stuart H. Rubin, Michael H. Smith, and Ljiljana Trajković. Distributed Denial of Service Attacks. In *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics, Volume: 3, 8-11 Oct. 2000*, pages 2275 – 2280.
- [10] M. R. Titchener. A Deterministic Theory of Complexity, Information and Entropy. In *Proceedings of IEEE Information Technology Workshop*, page 80, 1998.
- [11] M. R. Titchener. A Novel Deterministic Approach to Evaluating the Entropy of Language Texts. In *Proceedings of 3rd Conference in Information-Theoretic Approaches to Logic, Languages, and Computation*, June 1998.
- [12] M. R. Titchener. Deterministic Computation of Complexity, Information and Entropy. In *Proceedings of IEEE International Symposium in Information Theory*, page 326, 1998.