

# An Empirical Evaluation of IP Time To Live Covert Channels

Sebastian Zander, Grenville Armitage, Philip Branch  
Centre for Advanced Internet Architectures (CAIA)  
Swinburne University of Technology  
Melbourne Australia  
{szander, garmitage, pbranch}@swin.edu.au

**Abstract**—Communication is not necessarily made secure by the use of encryption alone. The mere existence of communication is often enough to raise suspicion and trigger investigative actions. Covert channels aim to side-step this problem by hiding additional information within the ‘normal’ behaviour of pre-existing communication streams. The huge amount of data and vast number of different protocols in the Internet make it ideal as a high-bandwidth vehicle for covert channels. Several researchers have proposed modulation techniques to encode covert information into the IP Time To Live field. In this paper we compare the different encoding techniques and also propose two new improved encoding schemes. We present a software framework developed for evaluating covert channels in network protocols. We use this software to empirically evaluate the transmission rates of the different TTL modulation techniques for real Internet traffic.

## I. INTRODUCTION

Often it is thought that the use of encryption is sufficient to secure communication. However, encryption only prevents unauthorised parties from decoding the communication. In many cases the simple existence of communication or changes in communication patterns, such as an increased message frequency, are enough to raise suspicion and reveal the onset of events. Covert channels aim to hide the very existence of communication. They hide within pre-existing (overt) communications channels by encoding additional semantics onto ‘normal’ behaviours and characteristics of the overt channels.

Lampson introduced covert channels as a means to secretly leak information between different processes on monolithic systems [1]. In recent years the focus has shifted to covert channels in network protocols [2]. The huge amount of data and vast number of different protocols in the Internet make it ideal as a high-bandwidth vehicle for covert communications. The capacity of covert channels in computer networks has greatly increased because of new high-speed network technologies, and this trend is likely to continue. Even if only one bit per packet can be covertly transmitted, a large Internet site could lose 26GB of data annually [3].

Covert channels are primarily used to circumvent existing information security policies, to ex-filtrate information from an organisation or country in a manner that does not raise suspicions of the network owners or operators. While covert channels may not be used very frequently today, because of

increasing measures against ‘open channels’, such as the free transfer of memory sticks in and out of organisations, the use of covert channels in computer networks will likely increase in the near future [4].

The IP Time To Live (TTL) header field limits the lifetime of IP packets, preventing packets from living forever during routing loops [5]. A packet’s TTL is set by the sender and decremented by each network element along the path processing the packet’s IP header (e.g. routers and firewalls). Packets are discarded if their TTL becomes zero while still in transit. A number of researchers have proposed different techniques to encode covert information into the TTL field [6]–[8]. Since routers and middleboxes modify the TTL fields of packets and packets can take different paths through the network from covert sender to receiver the TTL covert channel is not error-free (noisy channel) [9]. The question arises what transmission rates can the different modulation techniques achieve with realistic channel errors?

In this paper we compare the performance of existing TTL modulation techniques. Whereas previous work usually assumed that the covert sender also generates the overt traffic used as cover for the hidden information, we also consider the scenario where the covert sender uses the overt traffic of unknowing sender/receiver pairs as a carrier. Since we find some shortfalls in the existing techniques we propose two new improved encoding techniques. We also propose a simple communication-theoretic model, which can be used to compute the channel capacity and transmission rate. We have developed a software framework for empirically evaluating covert channels in network protocols and use it to evaluate the error and transmission rates for the various encoding schemes based on real overt traffic captured in the Internet.

The paper is organised as follows. In section II we explain the basic concepts of covert channels. In section III we compare the different TTL covert channels, propose two new improved encoding techniques and a simple channel model. In section IV we introduce the software framework for empirically evaluating covert channels. In section V we present an empirical evaluation of error and transmission rates for the different encoding techniques based on real Internet traffic from several network traces. Section VI concludes and outlines future work.

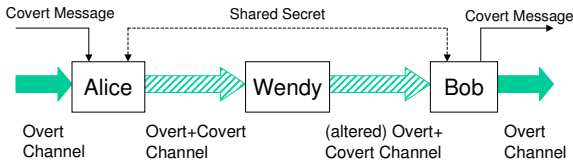


Figure 1. The prisoner problem – model for covert channel communication

## II. COVERT CHANNELS OVERVIEW

This section provides a brief overview of covert channels in computer network protocols. For a more detailed discussion see Zander *et al.* [2].

The de-facto standard covert channel communication model is the prisoner problem [10]. Two people, Alice and Bob, are thrown into prison and intend to escape. To agree on an escape plan they need to communicate, but Wendy the warden monitors all their messages. If Wendy finds any signs of suspicious messages she will place Alice and Bob into solitary confinement – making an escape impossible. Alice and Bob must exchange innocuous messages containing hidden information that (hopefully) Wendy will not notice.

In our case Alice and Bob use two networked computers to communicate. They run some innocuous overt communication between their computers, with a hidden covert channel. Alice and Bob share a secret, used for determining encoding parameters and for encrypting/authenticating the covert messages. For practical purposes Alice and Bob may well be the same person (e.g. a hacker ex-filtrating restricted information). Wendy manages the network and monitors the passing traffic for covert channels or alters the passing traffic to disrupt or eliminate covert channels. Figure 1 depicts the communication model (Alice sending to Bob).

In computer networks Alice and Bob do not need to be the sender and receiver of the overt communication. One or both of them may act as a middleman (see Figure 2). If Alice can observe and manipulate an existing overt communication from an innocent sender that reaches Bob, she can insert a covert channel into it. Bob does not need to be the receiver of the overt communication, but merely must be able to observe it to decode the hidden information. If Bob can also alter the overt communication, he can even remove the covert channel preventing the receiver of the overt communication from discovering it. A middleman could be located for example inside a network router or inside an end host's network stack.

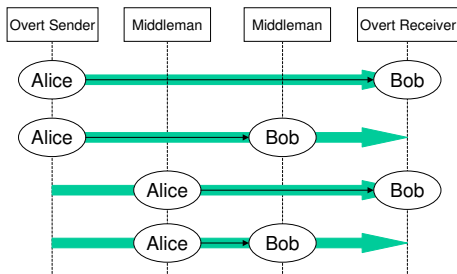


Figure 2. Communication scenarios depending on sender/receiver location

## III. COVERT CHANNELS IN THE TTL FIELD

### A. Existing Encoding Techniques

We group the existing techniques for encoding covert information into the TTL field into three classes:

- *Direct encoding* encodes covert bits directly into bits of the TTL field,
- *Mapped encoding* encodes covert bits by mapping bit values to specific TTL values and
- *Differential encoding* encodes covert bits as change between subsequent TTL values.

Qu *et al.* describe two different techniques [6]. The first method (referred to as Qu04-1) encodes bits into TTLs using mapped encoding. The original TTL value represents a logical 0 and a TTL value increased by an integer  $\Delta$  a logical 1. The second approach (referred to as Qu04-2) encodes one bit of covert data directly into the least significant bit of TTLs.

Lucena *et al.* proposed modulating the IPv6 Hop Limit field (IP TTL equivalent in IPv6) [7]. Their technique (referred to as Lu05) encodes one bit per packet pair using differential encoding. A logical 1 is encoded as TTL increase by  $\Delta$  and a logical 0 as TTL decrease by  $\Delta$ .

Zander *et al.* proposed a mapped encoding (referred to as Za06) that only decreases TTL values and therefore does not violate the IP standard [8]. In this scheme the original TTL value represents a logical 0 and a TTL value decreased by  $\Delta$  represents a logical 1.

Figure 3 shows the encoding of an example bit sequence by the existing techniques and the two new schemes we present in section III-C. Mapped and differential techniques encode the covert bits as different signal levels starting with the original TTL value. Direct techniques encode the covert information directly into bits of the TTL value (hence the representation as actual numbers). Qu04-1, Za06 and Scheme2 can encode up to seven bits per overt packet depending on the original TTL value, whereas the other techniques can only encode one bit per packet (pair).

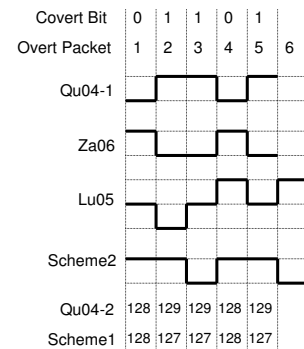


Figure 3. Example bit sequence encoded by the different modulation schemes

In this paper we consider two different scenarios:

- *Prison scenario*: Alice and Bob hide their covert communication inside their own innocent overt communication.
- *Middleman scenario*: Alice and Bob are middlemen using overt traffic of unknowing sender/receiver pairs as carrier.

Alice sends covert information to Bob by modulating the TTL field of subsequent packets. We assume that Alice aims to be as stealthy as possible. This means in the prison scenario Alice chooses ‘normal’ initial TTL values (usually 32, 64, 128, 255 as described in [11]) and in the middleman scenario Alice alters TTL values as little as possible. Since in the middleman scenario the overt traffic consists of multiple packet flows (identified by the 5-tuple of IP addresses, port numbers and protocol) with different initial TTLs, Alice has to manipulate TTL values on a per-flow basis to avoid drastic changes (except for direct encoding). Furthermore, Alice will always try to minimise  $\Delta$  but might use  $\Delta > 1$  to increase capacity.

### B. Shortfalls of Existing Techniques

Qu and Lucena both propose encoding information by increasing the original TTL value. This violation of the IP standard [5] is problematic in the middleman scenario (if routing loops occur) and also means these techniques cannot be used if the original TTL value is high (some operating systems use an initial TTL of 255 [11]). Increasing a high TTL value causes the TTL to ‘wrap-around’ in the 8-bit number space resulting in very small values. It is then very likely that packets will be dropped because of zero TTLs before they reach their destination. Furthermore, these techniques can be easily detected if the warden is in close proximity to the covert sender. Increasing the TTL likely leads to TTL values slightly above the usual initial values, which is very suspicious.

Lu05 is problematic because there is no upper limit on how a TTL value is changed. Long series of 0 or 1 bits lead to a large decrease or increase of the TTL value (including wrap-arounds in the number space). This means that regardless of the initial TTL value it is likely that some packets will be dropped (zero TTL) or packets could stay forever in the network (routing loops). The problem can be mitigated by encrypting the data with a cipher producing a uniform random distribution (making long series unlikely) or prevented by scrambling the data (limiting the series length). A warden can easily detect Lu05 because the modified flows are likely to have more than two distinct TTL values, which is uncommon for normal flows [8].

Both mapped encoding schemes are problematic in the middleman scenario, because the receiver must learn the mapping between bits and TTL values before decoding. If the overt traffic consists of many short flows (typical Internet traffic) the probability that the receiver is unable to learn the mapping before the flow ends can be fairly high resulting in a high error rate. For the current schemes is sufficient to learn the TTL value for 0-bits. Therefore the problem can be mitigated by sending one (or more) special 0-bit(s) at the start of each flow. The receiver uses the bit(s) to learn the mapping and then removes them from the data stream.

Direct encoding schemes require the receiver to know or periodically measure the number of hops between covert sender and receiver. Alternatively, the receiver could guess the hop count assuming the overt traffic can only traverse a single path and the receiver can determine whether the

decoded information is valid (e.g. checksum). Direct encoding is less robust against noise because there is no  $\Delta$  that could be increased.

### C. New Encoding Techniques

Scheme1 directly encodes covert bits into TTL values. It has the advantage over to Qu04-2 that TTL values are always decreased and more than one bit can be encoded per packet (making the scheme tunable towards capacity or stealth). The maximum number of bits that can be encoded per packet is:

$$n_{max} = \lfloor \log_2 (I - h_{max}) \rfloor \quad (1)$$

where  $I$  is the initial TTL,  $h_{max}$  is the upper bound on the number of hops between (covert) sender and (real) receiver and  $\lfloor \cdot \rfloor$  denotes the floor operation. The sender encodes covert information by setting the TTL to:

$$TTL_S = TTL - ((TTL[0, \dots, n_{max}] - B) \bmod 2^{n_{max}}) \quad (2)$$

where  $TTL[\cdot]$  is the least significant  $n_{max}$  bits of the original TTL value and  $B$  is  $n_{max}$  bits of covert information. Assuming a packet traverses  $h$  hops between sender and receiver the TTL value at the receiver will be  $TTL_R = TTL_S - h$  and the covert information is decoded as follows:

$$B = (TTL_R + h_R)[0, \dots, n_{max}] \quad (3)$$

where  $h_R$  is the hop count known by the receiver (and without channel errors  $h_R = h$ ).

Scheme2 is based on differential coding and is similar to Alternate Mark Inversion (AMI) coding. It can be tuned towards stealth or capacity by increasing the amplitude of the signal, but it can encode only one bit per overt packet pair. It has the following advantages over Lu05: TTL values are always decreased and the TTL values in one flow never change by more than one if  $\Delta = 1$ .

The covert sender encodes a logical 0 by repeating the last TTL value. A logical 1 is encoded by a TTL change, alternating between the two possible values (see Table I). The receiver decodes a constant TTL as logical 0 and a TTL change as logical 1.

Table I  
CURRENT TTL BASED ON COVERT BIT AND PREVIOUS TTL

Encode	Previous TTL	Current TTL
0	TTL	TTL
0	TTL - $\Delta$	TTL - $\Delta$
1	TTL	TTL - $\Delta$
1	TTL - $\Delta$	TTL

### D. Channel Model and Capacity

We assume one bit of covert data is encoded per packet (pair) to maximise stealth (and keep the channel model simple). However, our model can be extended to channels where multiple bits are encoded per packet (pair).

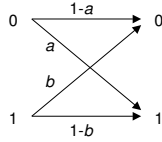


Figure 4. TTL covert channel model

The channel capacity is the maximum rate at which communication with arbitrary small error is possible [12]. The capacity of a channel is affected by channel errors. There are three possible sources of errors for the TTL covert channel:

- Deletions of bits caused by packet loss,
- Bit errors caused by reordering of packets and
- Bit errors caused by TTL modifications and path changes.

If covert sender/receiver are able to utilise transport protocol information, the first two errors depend on the properties of the transport protocol i.e. whether it provides reliable in-sequence packet delivery (e.g. TCP) or not (e.g. UDP). For simplicity and given the space limitations of this paper we assume negligible transport errors and focus solely on the noise caused by modification of the TTL field on the path between covert sender and receiver and path changes (see [8], [9]). We model the TTL covert channel as discrete memoryless channel assuming the current output of the TTL covert channel only depends on the current input and the error, but not on any previous input.

If the error is asymmetric the binary asymmetric channel (BAC) is used (see Figure 4). The BAC has two input/output symbols where the first symbol is changed to the second with probability  $a$  and the second symbol is changed to the first with probability  $b$ . The capacity is [13]:

$$C = \frac{-b \cdot h(1-a) + (1-a) \cdot h(b)}{b+a-1} + \log_2 \left( 1 + 2^{\frac{h(1-a)-h(b)}{b+a-1}} \right) \quad (4)$$

where  $h(\cdot)$  is the binary entropy. If the error is symmetric then the simpler binary symmetric channel (BSC) is used. The BSC is a channel with two input/output symbols where each input symbol is changed to the other with error probability  $p = a = b$ . The capacity is [12]:

$$C = 1 - h(p) = 1 + p \cdot \log_2(p) + (1-p) \cdot \log_2(1-p). \quad (5)$$

Both models assume that the input is uniform random, which can be achieved for example by encrypting the data. Whether the channel is symmetric or asymmetric depends on the modulation technique and  $\Delta$ . With the channel model we can compute the channel capacity if the error probabilities are known. In section V we empirically measure the error rates for all encoding schemes based on real overt traffic.

The capacity  $C$  is in bits per overt packet (pair). If  $f$  is the average frequency of overt packet (pairs) per second the average transmission rate  $R$  in bits/second is:

$$R = C \cdot \frac{1}{f}. \quad (6)$$

## IV. SOFTWARE FRAMEWORK

We have developed a flexible software framework for empirically evaluating covert channels in network protocols called Covert Channels Evaluation Framework (CCHEF) [14]. CCHEF runs under Linux and can be used in real networks with real overt traffic. Usually testing with real traffic is restricted to controlled testbeds where it is almost impossible to generate a realistic traffic mix from a larger number of hosts. Therefore, CCHEF also runs on single hosts emulating covert channels based on overt traffic from traces.

The central component of CCHEF is the *Channel* module that interfaces with multiple device modules. Covert data to be sent ( $B$ ) is read from the *Covert In* device, while received covert data ( $B^*$ ) is written to the *Covert Out* device. The *Overt In/Out* device taps into a stream of IP packets to be used as the carrier for the covert data. At the sender, suitable overt packets ( $O$ ) are intercepted and passed to the Channel module. The Channel module encodes the covert data and passes the modified packet ( $O^*$ ) back to the device, which will re-inject it into the network. If an overt packet ( $O^*$ ) arrives at the receiver the Channel module decodes any covert information and removes the covert channel (if possible) before re-injecting the packet. (CCHEF also supports passive receivers that use copies of overt packets and do not delay the actual traffic, if removing the covert channel is not necessary.)

The Channel module has various sub-modules responsible for modulation, framing, reliable transport, encryption etc. We implemented modules for all TTL encoding techniques.

Figure 5 shows CCHEF transmitting covert information over a network from Alice to Bob. The figure shows a unidirectional channel but in general, channels in CCHEF are bi-directional (depending on the available overt traffic).

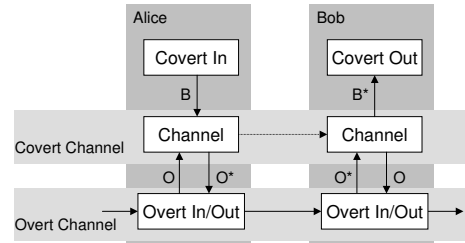


Figure 5. CCHEF used for transmitting information across the network (Alice sending to Bob)

Figure 6 shows how CCHEF is used with trace files. Covert data is encoded into overt traffic read from a packet trace and then decoded immediately (Alice and Bob are one entity).

A number of devices exist. The *Random* device generates uniform random covert data (useful to avoid bias towards specific input data). The *Text* device reads/writes textual covert data from/to text files. The *Trace* device reads overt packets from various trace file formats (e.g. libpcap, Endance record format) or from a live network device. The *NetfilterQueue* device intercepts packets via a Netfilter queue inside the Linux kernel and re-injects the packets back into the kernel [15].

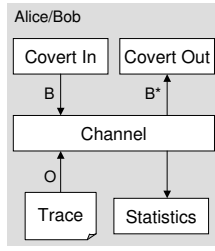


Figure 6. CCHEF used for evaluating covert channels based on overt traffic from trace files

CCHEF keeps track of the state of overt traffic flows so that modulation modules can be applied on a per-flow basis and rules can limit the overt traffic used to certain flows (based on IP addresses, port numbers and protocol).

## V. EVALUATION

### A. Datasets and Methodology

The overt traffic is taken from five public packet traces collected in the Internet (described in more detail in [9]). We use 24 hours of the Bell, Twente and Waikato traces, and six hours of the NZIX and Leipzig traces (each containing between 20 and 261 million packets). We use uniform random covert data to avoid any bias towards specific input data.

For direct encoding schemes we assume perfect knowledge of the true hop count at the receiver. For mapped schemes we consider both cases: 1) the receiver knows the mapping (**prison scenario**) and 2) the receiver learns the mapping from the extra 0-bit at the flow start (**middleman scenario**).

We define  $A$  as the peak-to-peak amplitude of the encoding schemes (difference between the signal level of 1-bit and 0-bit). Then for direct schemes  $A = 1$ , for Lu05  $A = 2\Delta$  and for all other techniques  $A = \Delta$ . We vary the amplitude within a limited range to investigate its influence on the error rate, but avoid large changes that would compromise stealth.

Because of the random input data we evaluate all encoding schemes five times for each trace and report the mean error rate. Since the standard deviations are small we do not include errors bars in the graphs for better readability.

### B. Error Rate

Figure 7 and 8 show the error rates for the Leipzig and Waikato dataset depending on the different modulation schemes and amplitude (note the y-axis is logarithmic). For space reasons we cannot show graphs for all datasets, but the overall trends are similar across all datasets.

For the Leipzig and Waikato traces the average TTL change rates are approximately  $5^{-3}$  and  $4^{-3}$  changes per packet pair according to [9] and the actual error rates experienced by the different modulation schemes (except mapped in the middleman scenario) are fairly similar for  $A = 1$ . The error rate does not decrease quickly with increasing  $A$  because the empirical error distributions have long tails [9].

Mapped encoding schemes in the middleman scenario have significantly higher error rates than the other schemes, because

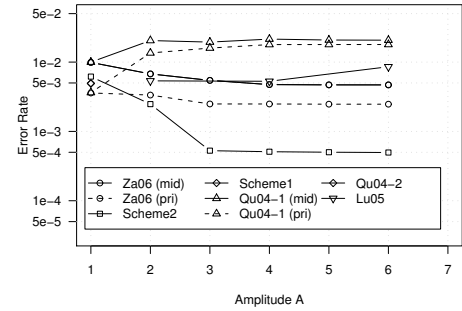


Figure 7. Error rate for different modulation schemes and amplitudes (Leipzig dataset)

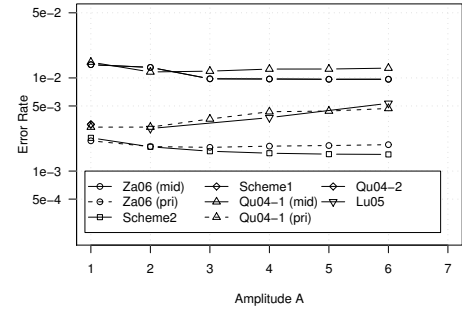


Figure 8. Error rate for different modulation schemes and amplitudes (Waikato dataset)

the probability that the first special 0-bit is wrong is higher than the average error rate as TTL changes occur more frequently at the flow start [9]. Mapped encoding schemes perform much better (especially for  $A = 1$ ) in the prison scenario where the covert sender controls the initial TTL and no special 0-bit is needed. The error rate for Qu04-1 and Lu05 actually increases because wrap-arounds in the number space happen for high original TTL values even for small  $A$  (illustrating the problem described in section III-B).

A method to decrease the error rate for mapped and differential encoding schemes is to use hop count differences instead of TTL differences. The receiver converts all TTL values to hop counts by subtracting each TTL value from the nearest typical initial TTL value before decoding. This eliminates errors when the TTL was changed by middleboxes (e.g. firewalls) but the hop count is the same [9]. For example, the TTLs 56 and 120 are different but the hop count is 8 for both assuming the usual initial TTL values of 64 and 128.

However, this technique does not work if modulated TTL values cross boundaries between different initial TTL regions (e.g. increasing a TTL value of 62 by 4 would change the estimated hop count from 2 to 62). This is the case for Lu05 and Qu04-1 even for very small  $A$ , because they increase TTL and often TTL values are close to the initial TTL value. Za06 and Scheme2 show some notable improvement across all datasets for  $A \leq 3$  but for larger amplitudes the same negative effect occurs. Table II shows the percentage by which the error rate is reduced averaged over all traces for different  $A$ .

Table II  
ERROR RATE REDUCTION WHEN USING HOP COUNT DIFFERENCES  
AVERAGED ACROSS ALL TRACES

Amplitude	1	2	3
Za06 (mid)	1.7%	2.5%	6.4%
Za06 (pri)	25.6%	37.5%	41.5%
Scheme2	5.4%	10.9%	34.4%

### C. Capacity and Transmission Rate

Figure 2 shows an example of the capacity of BSC and BAC (with  $a$  equal to  $1/x$  of the error rate and  $b = p - a$ ) depending on  $A$  for Za06 (middleman scenario) and Scheme2 for the Leipzig trace.

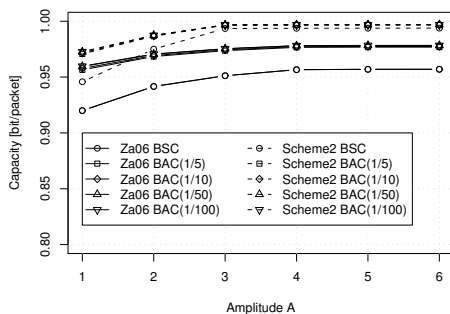


Figure 9. Capacity of BSC and BAC for Za06 and Scheme2 (Leipzig dataset)

The capacity of the BAC is higher for larger error rates (smaller  $A$ ) but for small error rates there is almost no difference. The BSC model always provides a lower bound.

Table III shows the average transmission rates in bits per second (Equation 6) for all encoding schemes and traces assuming 1 bit/packet encoding,  $\Delta = 1$ , the BSC model and hop count differences are used for Za06 and Scheme2. The transmission rates are the sum of the rates in both directions.

Table III  
AVERAGE TRANSMISSION RATES IN (KILO) BIT/SECOND

Dataset	Direct	Qu04-1		Za06		Lu05	Scheme2
		pri	mid	pri	mid		
Twente	481	483	437	483	437	439	444
Waikato	1396	1398	1095	1426	1104	1197	1206
Bell	173	212	203	229	213	208	207
NZIX	2037	2378	1935	2390	1961	2259	1515
Leipzig	11.6k	10.7k	10.2k	11.7k	10.2k	10.5k	10.5k

Overall the transmission rates vary between several hundred bit/s for medium volume links (Bell, Twente) and several kbit/s for high-speed large volume links (NZIX, Leipzig, Waikato) and the differences between the encoding schemes are not very large. Besides being standards-compliant and stealthier Za06 and Scheme2 also have equal or higher transmission rates (with the exception of Scheme2 for NZIX, because of many flows with frequent TTL changes [9]).

For the mapped encoding schemes and Scheme1 it is possible, although less stealthy, to encode multiple bits per TTL to further increase the transmission rate.

## VI. CONCLUSIONS AND FUTURE WORK

We compared the performance of four existing and two new improved TTL covert channel encoding techniques. We introduced a software framework for empirically evaluating covert channels in network protocols and used this framework to evaluate transmission rates of the different TTL covert channels for real Internet traffic. We find that the emulated covert transmission rates vary between approximately 200bit/s and 12kbit/s depending on the overt traffic and the differences between encoding techniques are not very large. In comparison with previous techniques our novel schemes are stealthier, do not violate the IP standard and provide equal or better transmission rates.

We are working on extending our channel model to include packet loss and reordering, and evaluate the impact of these on the transmission rates. Also we are developing analytical solutions for the error probabilities of different encoding schemes in order to compare theoretical error rates with empirical findings. Byte or frame synchronisation is a necessity for the TTL covert channel and we plan to evaluate and compare several different techniques. Furthermore, we aim to develop formal methods for evaluating the stealth of different encoding techniques.

## REFERENCES

- [1] B. Lampson, "A Note on the Confinement Problem," *Communication of the ACM*, vol. 16, pp. 613–615, October 1973.
- [2] S. Zander, G. Armitage, P. Branch, "A Survey of Covert Channels and Countermeasures in Computer Network Protocols," (*accepted for publication in*) *IEEE Communications Surveys and Tutorials*, 2007.
- [3] G. Fisk, M. Fisk, C. Papadopoulos, J. Neil, "Eliminating Steganography in Internet Traffic with Active Wardens," in *Proceedings of 5th International Workshop on Information Hiding*, October 2002.
- [4] M. Van Horenbeeck, "Deception on the Network: Thinking Differently About Covert Channels," in *Proceedings of 7th Australian Information Warfare and Security Conference*, December 2006.
- [5] J. Postel, "Internet Protocol," RFC 0791, IETF, Sept. 1981. <http://www.ietf.org/rfc/rfc0791.txt>.
- [6] H. Qu, P. Su, D. Feng, "A Typical Noisy Covert Channel in the IP Protocol," in *Proceedings of 38th Annual International Carnahan Conference on Security Technology*, pp. 189–192, October 2004.
- [7] N. B. Lucena, G. Lewandowski, S. J. Chapin, "Covert Channels in IPv6," in *Proceedings of Privacy Enhancing Technologies (PET)*, pp. 147–166, May 2005.
- [8] S. Zander, G. Armitage, P. Branch, "Covert Channels in the IP Time To Live Field," in *Proceedings of Australian Telecommunication Networks and Applications Conference (ATNAC)*, December 2006.
- [9] S. Zander, G. Armitage, P. Branch, "Dynamics of the IP Time To Live Field in Internet Traffic Flows," Tech. Rep. 070529A, CAIA Technical Report, May 2007. <http://caia.swin.edu.au/reports/070529A/CAIA-TR-070529A.pdf>.
- [10] G. J. Simmons, "The Prisoners' Problem and the Subliminal Channel," in *Proceedings of Advances in Cryptology (CRYPTO)*, pp. 51–67, 1983.
- [11] C. Jin, H. Wang, K. Shin, "Hop-Count Filtering: An Effective Defense Against Spoofed DoS Traffic," in *Proceedings of the 10th ACM International Conference on Computer and Communications Security (CCS)*, pp. 30–41, October 2003.
- [12] T. Cover, J. Thomas, *Elements of Information Theory*. John Wiley & Sons, Inc., 1991.
- [13] R. Silverman, "On Binary Channels and Their Cascades," *IEEE Transactions on Information Theory*, vol. 1, December 1955.
- [14] S. Zander, "CCHEF - Covert Channels Evaluation Framework," 2007. <http://caia.swin.edu.au/cv/szander/ccchef/>.
- [15] H. Welte, "Netfilter Queue Library." [http://www.netfilter.org/projects/libnetfilter\\_queue/](http://www.netfilter.org/projects/libnetfilter_queue/).