A Novel Class of Robust Covert Channels Using Out-of-Order Packets

Adel El-Atawy, *Member, IEEE*, Qi Duan, *Member, IEEE*, and Ehab Al-Shaer, *Member, IEEE*

Abstract—Covert channels are usually used to circumvent security policies and allow information leakage without being observed. In this paper, we propose a novel covert channel technique using the packet reordering phenomenon as a host for carrying secret communications. Packet reordering is a common phenomenon on the Internet. Moreover, it is handled transparently from the user and application-level processes. This makes it an attractive medium to exploit for sending hidden signals to receivers by dynamically manipulating packet order in a network flow. In our approach, specific permutations of successive packets are selected to enhance the reliability of the channel, while the frequency distribution of their usage is tuned to increase stealthiness by imitating real Internet traffic. It is very expensive for the adversary to discover the covert channel due to the tremendous overhead to buffer and sort the packets among huge amount of background traffic. A simple tool is implemented to demonstrate this new channel. We studied extensively the robustness and capabilities of our proposed channel using both simulation and experimentation over large varieties of traffic characteristics. The reliability and capacity of this technique have shown promising results. We also investigated a practical mechanism for distorting and potentially preventing similar novel channels.

Index Terms—Covert channels, data-hiding, packet reordering, packet sequence, protocol-based steganography

1 Introduction

Covert channels are defined as a communication mechanism that can evade access control policies by using a medium that normally goes by unmonitored. Using another channel as the host/overt medium, a sender can piggyback his secret along with other public pieces of information to an accomplice on the other side of the security perimeter. Network covert channels include any communication over data networks that uses network-specific mechanisms to construct such an invisible channel.

In the past few years, the Internet has exploded to include millions of users communicating with thousands of applications using hundreds of protocols. However, data transmission has been always characterized by being visible, in the open and available to anyone to collect and analyze. All data protection techniques focus on protecting the payload rather than hiding the existence of the channel itself. In other words, everyone is aware of the communication taking place even if the content itself is not readily available. Covert communications are addressed as another dimension of the information confidentiality requirement for secure systems. Knowing that a communication exists between two parties is a valuable piece of information even if the content is unknown.

Previous attempts to design network covert channels were mostly in one of two directions. The first direction is using a storage channel were actual bits in traffic packets are manipulated in order to store the required secret. For example, the use of packet identifier and other fields in the IP header to store information bits was surveyed in [20]. The downside of these techniques is that they are preventable by techniques such as packet normalization. The second family of techniques try to build a timing channel by manipulating inter-packet gaps (directly or indirectly) to encode the covert channel (CC) (e.g. [10], [29]). Many of these timing channels can be prevented by traffic shaping and jitter manipulation [14], [17], although some time-based techniques show resistance to random jitters [13], [18].

We propose a technique that can use the unconventional channel of packet order to be our covert communication medium. By manipulating the order of packets sent over the network at the sender-side, we emulate the packet reorder phenomenon which takes place naturally. Detecting the presence of the reordering based covert channel through monitoring at the network core is difficult because Interceptors or Monitors will not be able to order these packets due to the overwhelming computational cost of buffering and sorting packets at the network core.

Moreover, the reorder extent is designed not to affect the normal operation of the transport layer protocol. The transport protocol (e.g. TCP) will perform its duty in reordering the packets for delivery to the application layer transparently of the underlying hidden operations.

The covert channel communicators discussed in this paper are assumed to be the sender and receiver of the overt channel (OC). Ideally, the receiver will get ordered packets; $1, 2, \ldots, n$ (i.e., sending order). Any out-of-order packets will be handled transparently by the transport layer. This unordered sequence bares more information that we will utilize as our covert channel medium. In our approach, we intentionally send out-of-order packets where different permutations carry different values/codes. Therefore, our channel is independent from

- *A. El-Atawy is with the Google Inc.*
- *Q. Duan and E. Al-Shaer are with the University of North Carolina at Charlotte. E-mail: qiduan@gmail.com, ealshaer@uncc.edu.*

Manuscript received 15 Apr. 2013; revised 8 Apr. 2015; accepted 20 May 2015. Date of publication 10 June 2015; date of current version 15 Mar. 2017. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TDSC.2015.2443779

the packets' payloads and not very sensitive to inter-packet jitter, which means that our proposed method goes well beyond previous works that used packet payload to hide the secret messages. The proposed mechanism consists of 1) a code selection mechanism that applies coding theory to carefully select permutation patterns that will enhance robustness and error-resistance, 2) a traffic imitation component that calculates the codeword distribution in order to evade detection, and 3) an encoder component at the sender-side as well as 4) a decoder technique for retrieving the sent secret codes at the receiver end.

The goal of this work is to build a channel which is reliable and practical while maintaining a reasonable bandwidth. In such systems, there are always three factors competing: bandwidth, robustness, and stealthiness. By increasing the bandwidth, the channel used to hide information (host channel) becomes more susceptible to noise and being discovered by adversaries. Lowering the hidden bandwidth (i.e., data transmitted in the host channel) causes the transmission to be more stealthy and more resistant to change due to external noise. We will investigate the parameters that will affect each of these factors, and analyze the consequences of changes applied to each one.

Although the technique is hard to detect or eliminate completely, we will show that it is still possible to effectively disrupt the ongoing communication. The straightforward solution by complete re-sorting of packets to eliminate the channel completely is very expensive, and impractical in most cases. Therefore, we take another approach where we show how noise (i.e., extra out-of-sequence packets) can be practically injected into the traffic by packet shuffling. While the technique is far from being guaranteed to prevent the existence of such a channel, it is quite effective in reducing its possible bandwidth. A better protection (and possible detection) mechanism for this channel and other similar designs is still an open research topic. The difficulty arises from the fact that any responsive system carries the implicit potential of being a medium for covert channels, and it is theoretically impossible to eliminate it completely [8], [31].

The following sections start with an overview of the related work in the area of covert communication in Section 2, and the phenomenon of out-of-order packets and studies of its abundance and persistence in Section 3. The proposed covert channel design is then described in Section 4 including a simple prototype followed by the complete design. Implementation details and examples of potential applications are discussed in Section 5. Suggestions for hindering the channel success is shown in Section 6. The evaluation are provided in Section 7. Limitations and final discussion are in Sections 8 and 9, respectively.

2 Related Work

Previous work on covert channels in Internet traffic have focused on hiding information in either packet payload (not true network-based), or inside IP header fields. Some work used IP options fields, or source port patterns to hide the secret information. Others used timing information of packet sequences. The closest work to what we are planning to do, and one of the most recent, is the work by Shah et al. in [30], that used timing channels for leaking types passwords from and [1] that used packet sequences/order as a storage media.

Previous work on covert channels [27] in network traffic can be divided according to the storage media used: packet payload, packet header, or packet timing/behavior. Different researchers have focused on identifying the possible applications in public networks [33] and their theoretical capacity bounds [32]. Here we focus on methods that using timing channels.

Timing channels were originally suggested to attack crypto-systems and to leak information between HI and LO processes (security-clearance wise) [14]. Many researchers worked on analysis of timing information over Internet traffic to obtain worst case bandwidth variation and end-to-end delay/jitter measurements. These are necessary in the analysis of channel capacity. The work in [19] presents a new class of reliable timing channels called *Cloak* which encodes a message by the unique distribution of packets over TCP flows. Our work is more stealthier than *Cloak* since we consider the frequency distribution of different re-orderings. The work in [1] is very close to our approach where packet order is manipulated to store information. However, their approach is limited by having a specific number of patterns, and lack of generalization in the analysis. Also, code selection was not made to specifically to evade packet order metrics in the literature. There has been extensive theoretical study of this type of side channels. In [31], it was shown that for real time systems to exist, security requirements have to be relaxed as the bandwidth of covert-timing channels associated with system events can increase indefinitely. The capacity of transmission channels having a queue as the source of non-determinism is given in [2]. The authors showed that it is possible to exceed the raw capacity of the channel by augmenting this channel with a side-timing-channel. The analysis was theoretical and not directly applicable to our network environments, yet it shows the high potential of such techniques. Other works such as [8] tried to characterize the abilities of such channels in more complex multi-level systems and design a generic method for limiting its bandwidth.

The work in [34] demonstrates the theoretical limitations of low-latency anonymous communications systems, and shows that achieving anonymity in low-latency communication systems is difficult for current flow transformation based systems. The steganographic timing channel introduced in [18] is both robust and provably undetectable for network traffic with independent and identically distributed (i.i.d.) inter-packet delays. In [13], the authors present the CoCo covert channel which modulates the covert message in the inter-packet delays of the network flows, with a coding algorithm that is used to ensure the robustness of the covert message to different perturbations.

Counter-Steganography Techniques: Techniques for discovering the existence of covert channels is far less mature than the hiding techniques themselves, and the theoretical work behind them. In [15], a discussion of the relationship between covert channels and steganography is presented. In [9], techniques are presented that manipulates packet timing in order to ruin the secret channels. A classic technique was suggested before in [14], where fuzzy timing is introduced to add non-determinism into the timing channels and significantly lower its bandwidth. Their focus was information leaking between OS processes, but the same concept still applies. In [7], a framework to detect data-hiding via tunneling in web traffic was shown. By monitoring statistical properties expected to be prevalent like packet size and inter request timing, alerts can be issued when suspected tunneling occurs. Information theory and entropy measurements were used in [12] to detect covert timing channels by observing changes in the information content in the packet timing. A very effective technique but its applicability is still limited to timing channels only. In [36], a DSSS-based network flow marking technique is developed to introduce invisible marks into a target traffic flow which is persist and detectable despite a variety of anonymity schemes, making it possible to trace anonymous communications.

3 Packet Reordering Phenomenon

Packet reordering in network traffic is receiving packets in a different order than that used to send them. Obviously, this phenomenon is only applicable to packet switched networks. Circuit switching makes such behavior almost impossible (e.g. connection break/reestablish can be a rare cause for data to be received out of order). Previous studies [21] showed that it is mainly due to one of two reasons: 1) local built-in parallelism and connection slicing, and 2) multi-path forwarding routes [23]. The first reason was shown to be more influential. As the need for faster processing speeds and the increasing ease of adding parallel processing capabilities, the packet reordering phenomenon is not going to disappear in the near future, as shown in [6]. Therefore, we can claim we have a property that is abundant, and can be observed as a natural behavior.

3.1 Measuring the Extent of Out-of-Order Packets

Efficient, yet concise, metrics for measuring the amount of packet reordering has been the center topic for several research work. In [24], a survey of reordering metrics is presented. reorder density (RD) and reorder buffer-occupancy density (RBD) are the most applicable metrics to use in our application. They measure the reordered packets with respect to how far the received packets are away from what is expected, and how much buffering is needed for reorder, respectively. In this paper we use a variant of RD throughout the analysis and discussion. However, RBD can be used as well with no significant change to the logic or implementation of the channel. The problem with coming up with a valid metric for measuring the severity of out-of-order packets is that it can not be a single scalar value. For example, we can not simply count how many packets were out of sequence. If such a simplistic metric is used, should the sequence $< 3, 1, 2 >$ (shorthand for receiving packets in this order: pkt_3, pkt_1, pkt_2) be counted as just three coming early or one and two being late? This problem gets more complicated when we need to measure how out of order a packet is. For example, orders $< 4, 1, 2, 3 >$ and $< 1, 2, 4, 3 >$ should be result in different values for the metric to emphasize on the fact that in the first case it was shifted over three other packets rather than being swapped with an adjacent packet. Putting these factors (i.e., how many out-of-order packets and how out of order is a packet) together, we come to the conclusion that we need a vector metric. In [25] and [26] such metrics were introduced. RD (Reorder late/early Density) represents the distribution of differences between the packet original sequence number and the order by which it was received, while Obviously, a natural order will result in the distribution being concentrated around zero: packets received in their order or slightly early or late. Another property of RD is that spans both positive and negative values, making it unbounded at both ends and a bit harder to use as is in our application. On the other hand, reorder buffer-occupancy density measures the effect of reordered packets by building the frequency distribution of the receiving buffer usage. As more packets needs reorder, the more the reordering buffer will be used, and the number of elements in the buffer reflects the extent by which a packet is shifted from its ideal order. RBD has the advantage of being a single sided histogram and reflects to a greater extent the effect of out-of-order packets on the receiving end.

4 Packet Reorder Channel

To smooth the introduction of our design, we start with a very simple prototype to introduce the general idea of our channel. Afterwards, we will focus on each of the features needed to render the channel a fully functioning, stealthy, and robust channel.

4.1 Simple Prototype

In a connection-oriented session, the order of packets received is irrelevant at the application level thanks to underlying layers that sort received packets to reach the correct stream structure. In such connections (e.g. TCP flows), the original packet order is maintained by buffering out-of-order packets till intermediate (i.e., late) ones arrive at the destination. Each packet is marked with some sequence identifier (e.g. $sequence\_number$ in TCP or IPSec) whose nature and range depends on the end-to-end protocol used. As indicated before, many reasons contribute to causing this unordered reception of packets. It has always been considered a nuisance to the receiver and fixing it was one of the top priorities of such protocols. In our application domain, this phenomenon can be used or, more specifically, induced, to build a covert channel.

The normal/perfect received order of packets is $1, 2, \ldots, n$. If packets were sent in a specific order, then there is a great chance they will be received as such. Therefore, this out-of-order was always considered as either rare or an annoying property of some channels. However, if we considered the received packet order as a source of information, then we have a channel that is independent from the packets' payloads and not very sensitive to inter-packet jitter. If packets were never received out-of-order as in circuit switched networks, then this channel has zero capacity as its output is deterministic. However, this is not the case, and packet reordering is not that rare over the Internet (see [6]).

The question is: can we send information bits masqueraded in the form of packet order? If we assumed that packets in-order is one state of the channel (i.e., conveying one specific symbol, say a 0 bit), then other packet orders that deviate from the perfect order are other symbols of our alphabet. The number of different orders we can impose over the packet sequence implies the capacity. Quantitatively, the logarithm of the number of distinguishable orders equals the channel's capacity in bits. Therefore, for n packets the maximum capacity will be $\log n! \sim n \log n$. Taking a file transfer as our example: a $700\,\mathrm{MB}$ file, using 1.5 Kbyte packets. This will have $8.7\,\mathrm{Mbit} \sim 1.1\,\mathrm{MB}$ (466 K packets). Impressive as it may seem, this range is far from being practical as it literally destroys the host channel (no TCP stack can handle reordering this number of packets without failing). Moreover, this will jeopardize the stealthiness of the channel. Therefore, we will take a simple subset of this full-permutation space for demonstration: only adjacent packets can be reordered together. The coding space now is limited to 1 bit per packet pair (i.e., 0:in-order, 1:swapped), reducing the overall capacity to $233\,\mathrm{Kbit} \sim 30\,\mathrm{Kbyte}$. In other words, the entropy of this channel is 1-bit per symbol, where a symbol is represented via a packet-pair. In terms of the overt channel, the side channel provides a capacity of 0.5 bits per overt packet.

Furthermore, a TCP stream with a high rate of out-of-order packets (e.g. 50 percent of pairs on average being flipped) will look suspicious if monitored, and reducing this is a further step towards better stealthiness. Assuming a pair is swapped with a probability of 0.05, the entropy per packet-pair will be lowered to $H(0.05) = -0.05 \log 0.05 - 0.95 \log 0.95 = 0.2864\,\mathrm{bits}$ and the overall capacity will be $66.8\,\mathrm{Kbit} \sim 8.3\,\mathrm{Kbyte}$.

In this channel, we omitted some basic features that are highly needed for successful communication. Namely, error detection and correction on the symbol level. If a packet pair is reordered due to naturally occurring network behavior, our induced packet order might be canceled out causing a 1-bit to be received as a 0-bit or vice versa. It is not possible to detect this kind of errors other than relying on multi-bit code words (e.g. adding even/odd parity, or more sophisticated codes as Hamming code, Reed-Solomon, etc) or plain error detection via CRC-like mechanisms. In the following section, we will discuss how to build a real channel with enough details to support this feature in an intrinsic manner. This is achieved by extending the unit of transmission to be multiple packets for each transmitted symbol rather than a packet-pair. Moreover, better distribution and selection of packet-ordering patterns will be discussed in order to satisfy the stealthiness property as well as boosting the resilience for errors.

4.2 Refined Design

In order to extend our simple prototype to a realistic and valuable channel, some important features are needed. These features include error resistance via error detection and correction on the basic transmission level that corresponds to the physical layer in traditional channels. Also, for our specific application, the stealthiness of the channel is an important goal by definition. Therefore, the overall reorder should be kept within normal levels. This is represented via two factors: *reorder depth*, and *reorder volume*. The former indicates the farthest packets that can be swapped, or the extent by which a packet can be moved, and the latter represents the percentage of packets that are out of order. Both metrics can be represented concisely in the reorder density and the reorder buffer-occupancy distance metrics proposed by Piratla et al. in [25], [26]. The above two main features, if carefully addressed, will result in a channel that is, both robust and stealthy to a great extent. In the remaining portion of this section, we address each of our goals by: 1) defining our channel parameters, 2) selecting the codewords (i.e., permutation patterns), 3) enhancing error resilience by better codeword selection, 4) avoid detection by adjusting codeword probabilities.

4.3 Model and Notation

Let N be the total number of packets transmitted in the overt channel. Our covert channel will be built using every k packets from the overt channel to represent one codeword/symbol. A code word is a specific permutation of the k packets. Each codeword will be one of l different values selected from the set of code words L ($l = |L|$). Therefore, the upper bound on the information content of codeword is $\log_2(l)$ bits (when codewords are selected uniformly). Thus resulting in a total capacity of $N/k \times \log_2(l)$ bits. In our simple example from the previous section, we have $k = l = 2$ resulting in an overall data volume of $N/2$ bits. The two possible codewords used were "ordered" and "swapped" resulting in a plain single-bit binary coding. After introducing non-uniform use of codes for better stealthiness, the average information per codeword drops to $H(p_L)$, where p_L is the probability distribution of the L codewords the source uses to encode his hidden message.

In the case of having the transmission reorder of more than two packets exploited into a single codeword, say k packets; we will be having a maximum of $k!$ codewords. For the simple channel, this did not reflect any difference in our analysis. However, for $k > 2$, the capability of having error detection and correction is introduced, and some changes in our simple calculations are needed.

In general, an n-bit block code is denoted by (n, m) block code, if m bits of information were encoded into n-bit blocks. In that case, the coding rate R is equal to m/n. However, in our model, the difference in notation from bits per codeword to packets per codeword, and the use of permutation instead of bit values will change the resulting expressions. The block code will be denoted a $(k \log_2(k), \log_2(l))$ block code, and will have a code rate of $\log_2(l)/k \log_2(k)$. This will be needed when discussing the error correcting capabilities.

From now on, packets in the overt channel will be denoted by $P_1, P_2, \ldots P_N$. Inside code block i (i.e., symbol S_i transmitted over CC), the constituent packets will be denoted by $C_{i,1}, C_{i,2}, \ldots C_{i,k}$. Then, the overall sequence will be $C_{1,1}, \ldots C_{1,k}, C_{2,1}, \ldots C_{2,k}, \ldots$. If the symbol index in understood, or the discussion is not referring to a specific symbol then the first subscript will be dropped: $C_{i,j} \sim C_j$.

The communication between sender and receiver is assumed to be established after communicating through another form of secure communication to agree on system parameters (e.g. k, l, etc) and the flows to be used. Although this information can be guessed with some extra effort by the receiver, it will not be assumed in this paper. Table 1 shows the notations and symbols used in the paper.

4.4 Codeword Selection

From a covert channel owner point of view, the perfect environment is where: 1) no one suspects the presence of a CC (detectability constraint), and 2) no reordering is introduced from the network itself (noise in transmission). The first constraint limits the code size and codewords used. The longer

TABLE 1
Notations and Symbols Used

N	total packets in OC
k	packets to represent each codeword
N/k	number of transmitted symbols in CC
L	set of valid codewords
l	size of set L, $l = \mid L \mid$
p_L	probability distribution of codewords
$H(p_L)$	bits represented by each codeword
R	code rate, $R = \log_l / k \log_2(k)$

the codeword (i.e., higher k), the higher the probability a monitor will suspect the covert channel presence. The reason is that long reorders (e.g. 10 packets in perfect reverse order) are not naturally occurring in data networks. The noise effect can be limited by introducing larger block codes and deeper swaps. Both constraints can be handled via code block size compromise and intelligent codeword selection.

Limiting the codeword size k to low values (e.g. less than 5) will be enough for most cases, and it should be adjusted not to deviate significantly from the network's naturally induced reorders. Codeword selection and their probability distribution p_L can affect both constraints. For example, we can put a constraint that no codeword should correspond to a complete reverse of more than three packets. Also, we can limit the distribution of packets out of order to a certain threshold to evade detection.

Each codeword is defined by the order of the constituent packets. We defined packets in a codeword to be $< C_1, C_2, \ldots C_k >$. These values are assigned based on the original packets to be sent in that time order. Then, $< C_1, C_2, C_3, C_4 > = < 1, 4, 3, 2 >$ means we are defining one of the codewords to be a four packet-sequence sent in such an order that the second and fourth packets are swapped. Also, we define the reverse symbol: C'_i to denote the position of packet number i. So, in this example, C'_2 will be equal to 4 (i.e., the new position of packet 2). At the receiving end, the transport layer will unknowingly resort them to be $< 1, 2, 3, 4 >$ and forward the corrected sequence to the application layer. In the same time, a lower level module will be able to see the difference before resorting and extract the coded secret information.

Obviously, we have $k!$ ways to define codewords of length k. However, in order to be able to implement error-detection and correction, only a few of those will be used. The goal is to select a few, namely l codewords, out of the $k!$ possible permutations. This gives the previously calculated code rate that is sometimes also called efficiency $R = \eta = \log_2 l / \log_2 k!$. In our original prototype, $\eta = 1$ as expected, as there were no loss in efficiency to support error detection/correction. In a more sophisticated channel, take $l = 3,\ k = 4$, we have $\eta = 0.3456$. Normally, $\eta \leq 1$, where the equality is achieved if and only if all $k!$ codewords are to be used. This should be always avoided unless: 1) the value of k is very low (e.g. ≤ 5), and 2) the host channel is known not to introduce any further reordering of its own.

4.4.1 Detecting Single Errors

To be able to detect (though, not correct) all single bit errors, all code words are selected to be even permutations. A permutation is even if it needs an even number of element-swaps to reach the desired order starting from the perfect order (i.e., all packets are send in their intended order). Thus, our valid codewords must include the identity permutations (i.e., $< C > = < 1, 2, \ldots, k >$). Any single transposition between packets will yield an error in transmission that can be detected, yet not corrected. Considering that a high portion of errors taking place in natural transmission is in the form of a single adjacent packet swap, such code can achieve good error detection rates. However, our target is to select codewords that can detect and correct multiple errors in transmissions. In other words, if one packet (or more) are shifted from their intended location as specified by the original codeword, the receiver should be able to detect and optionally correct those into the correct permutation. Note that all the permutations are even to detect single errors, which might make the channel vulnerable for detection. However, it would be trivial to add a pseudo-random sequence known to the sender and receiver that randomly switches between even and odd.

4.4.2 General Code Selection

To be able to generate more powerful coding schemes, we formulate (and map) our permutation patterns into regular linear code blocks. Any of the l ($l \leq k!$) codewords $< C > = < C_{1 \ldots k} >$ can be represented efficiently using $k \log_2(k)$ bits. Each of the packets in a codeword is represented by a number that is essentially the RBD [26] before processing the packet. In other words, each packet in the codeword is represented by the number of packets of higher index received before it. $code(C_j) = \sum u(i - j)$, where $u(x) = 1\ (if\ x > 0)$ and $u(x) = 0$ otherwise. For example, receiving a codeword $< 1, 4, 3, 2 >$ is translated to $0, 2, 1, 0$. The packet indexed by 1 was not preceded by any other packet with larger index, while packet with index 2 was preceded by another two packets with higher index, and so on. Obviously, the packet with the highest index will always be translated to 0 regardless of its order, so it will be omitted from the mapping. Therefore, we represent each of the $k - 1$ packets using $\log_2 k$ bits, resulting in an overall $(k - 1) \lceil \log_2 k \rceil$ bits. This representation is asymptotically optimal, as the overall number of permutation patterns is $k!$, and it needs $\Theta(k \log_2 k)$ bits to be encoded.

Furthermore, in order to have a correct mapping regarding the Hamming distance induced by packet shift errors; we represent these numbers via binary reflected gray codes (BRGC) [28]. This representation is the easiest to compute from normal binary representation (i.e., $G = N \oplus (N/2)$), and we guarantee having each packet shift represented as a single extra Hamming distance.

In Table 2, we demonstrate an example using $k = 3$, with L equals the whole possible set of permutations (i.e., $l = k! = 6$). Say, we need to detect single errors, then a distance of 2 is required. In such cases, the maximum number of codewords we can use (to increase the code rate) is 3. The selected code words L will be $\{C_1, C_4, C_5\}$, and the code rate will be $R = \log_2 3/3! = 0.264$. Using these selected block code, we will be able to detect any single step shift of packets in the transmission due to natural causes. As another example, for the enhanced requirement of correcting a single error we need a Hamming distance of at least 3.

TABLE 2
Example Code Using $k = 3$ and $l = k!$

codeword	block coded	RD
$C_1 = < 1, 2, 3 >$	00 00	$< 1, 0, 0 >$
$C_2 = < 1, 3, 2 >$	00 01	$< 2/3, 1/3, 0 >$
$C_3 = < 2, 1, 3 >$	01 00	$< 2/3, 1/3, 0 >$
$C_4 = < 2, 3, 1 >$	11 00	$< 1/3, 1/3, 1/3 >$
$C_5 = < 3, 1, 2 >$	01 01	$< 2/3, 0, 1/3 >$
$C_6 = < 3, 2, 1 >$	11 01	$< 1/3, 1/3, 1/3 >$

The block code is written in BRGC. The RD column shows normalized histogram on the metric after the effect of a specific permutation.

Therefore, our only solution for selecting codewords will be $\{C_1, C_6\}$. Using this block code we will correct single errors while detecting up to two errors. In general, for detecting e errors a minimum distance of $e + 1$ is needed, and for e-error correction the requirement increases to $2e + 1$.

Generally, the selection process is performed on the block code to achieve the desired characteristics. However, in our case this is not a straightforward task, as there will be some block codes that are impossible to convert back to a permutation pattern. The space defined by $(k-1)\lceil \log_2 k \rceil$ bits is an upper bound to $k!$, with $k = 5$ as the highest ratio between the two space sizes: $(k-1)\lceil \log_2 k \rceil / \lceil \log_2 k! \rceil = (5-1)\lceil \log_2 5 \rceil / \lceil \log_2 5! \rceil = 1.714$.

4.5 Codeword Distribution

After selecting which codewords are valid to be used in order to obtain the desired error correction capabilities, comes the step to choose the input codeword probability distribution to our channel. Normally, the aim of choosing input probabilities is always to achieve the capacity of the channel being used given its error model. However, in our case, the goal is different where we are willing to sacrifice bandwidth in order to follow a behavior that will keep the stealthiness of the covert channel. For maximum transmission capacity, codewords should be used equally probable resulting in a transmission capacity of $\log l$ bits per codeword, or $\log l/k$ bits per overt packet used. However, to satisfy the stealthiness requirement we should use some patterns more than others not to deviate significantly from the normal behavior and be susceptible to raising alerts and being detected. Using metrics as reordering density [4], [25] and reordering buffer-occupied density [26], we will be able to know this target behavior.

For demonstration, let us consider the easiest case of $k = 2$. Although this will limit our code selection as there will not be any error detection/correct capabilities, we can still use it as a base for discussion. Assume a study of the host networks involved showed that a maximum unobserved RD histogram will have 90 percent of packets received as expected, and 10 percent are one step earlier. Let us denote that by the sequence $RD_t = < 0.9, 0.1 >$, where RD_t is the target RD. As we have only two possible codes (i.e., $< 1, 2 >$ and $< 2, 1 >$), then we just need to calculate the probability of using each to go below the detection radar.

Each code participate in a specific RD (RD_t) pattern RD_i which is the RD histogram caused by codeword i. As per our example, the first code $C_1 = < 1, 2 >$ will contribute with a normalized RD equals to $RD_1 = < 1, 0 >$ and the other code: $RD_2 = < 0.5, 0.5 >$. Then,

$$\begin{pmatrix} 1 & 0.5 \\ 0 & 0.5 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} = \begin{pmatrix} 0.9 \\ 0.1 \end{pmatrix}$$

where p_i is the probability of code i. Solving for p_i we find that the identity permutation should be used 80 percent of the time, and the swapped pair should appear only 20 percent in order to satisfy the stealthiness requirements of the example network.

Generally, we can write $RD \times P = RD_t$ or in detail:

$$\begin{pmatrix} RD_{1,1} & \dots & RD_{l,1} \\ \vdots & \ddots & \vdots \\ RD_{1,k} & \dots & RD_{l,k} \end{pmatrix} \begin{pmatrix} p_1 \\ \dots \\ p_l \end{pmatrix} = RD_t, \tag{1}$$

where $[RD]$ is a k *by* l matrix where each column is an RD_i k-vector, $RD_{i,j}$ specifies the contribution of the ith codeword on element j in the RD metric, and P is the l column vector of codeword probabilities to solve, and finally RD_t is the k vector with target overall RD effect of our covert channel.

A special case that can take place is having more than one codeword with the same metric vector. In such case, the calculations for calculating the distribution take place on the distinct vectors only. Formally, we solve the equation $RD \times P_{\overline{L}} = RD_t$, where $\overline{L}$ is a subset of L where only codewords with distinct RD vectors are included. Thus, we have $\overline{l} = |\overline{L}| = rank(RD)$, assuming $l \geq k$. Once the equation is solved and probability of each distinct metric value is obtained, it will be evenly split over codewords sharing this value. The uniformity of the distribution is used to provide the maximum entropy of the distribution [11]. The same effect can be obtained by including all vectors and perturbing identical vectors with insignificant values to impose independence. However, this approach might force the solving algorithm into an unstable state due to the high error of dealing with tightly related vectors.

To make the codeword stealthy, we need to pad zero codewords (codewords without any reordering and convey no information) to make the total number of reorderings appears to be natural. For a k packet codeword, the expected number of pairwise reorderings is $(k-1)k/4$. Suppose the average probability of pairwise reordering in a normal flow is q, then we need to pad $\lceil k(k-1)/(4q) - 1 \rceil$ zero codewords for every true codeword.

4.6 Performance and Capacity Calculations

Selecting the best set of codewords that satisfy the requirements of error detection and correction is a classically hard problem. The optimal solution is the largest set of codewords (i.e., maximum code rate) that satisfies the capability requirements. A generally applicable algorithm is not yet reached. In traditional linear block codes, the size of such set (L) is bounded as follows:

$$l = |L| \leq \frac{2^n}{\sum_{i=0}^{d_{min}} \binom{n}{i}}, \tag{2}$$

where d_{min} is the minimum Hamming distance required for handling the specified error rate, and n is the number of bits

TABLE 3
The Effect of Increasing the Codeword Size (in Packets) on the Maximum Detectable E_D and Correctable Errors E_C

k	2	3	4	5	6	7	8	9	10	11	12
E_C	0	1	2	3	4	5	7	8	10	12	14
E_D	0	2	4	6	8	11	14	17	21	24	28

in the block code (i.e., $n \approx k \log k$). However, this formula will not be fully applicable in our case, as some of the 2^n codewords are not mapped to permutations we can use. Moreover, this difference need to be reflected in the numerator and denominator of the given formula complicating things further.

For a k packet codeword, the resulting block code will be $O(k \log k)$ regardless of the mapping used. Therefore, the maximum number of errors we can correct E_C is bounded by $E_C \leq \lfloor (n-1)/2 \rfloor$, and the maximum number of errors detectable E_D is bounded by $E_D \leq n-1$, where n. These bounds are only achievable on the expense of having the minimum code rate R possible; $R = 1/n = 1/(k \log_2 k)$ by using only two out of the $k!$ possible permutations. Obviously, the two codewords that satisfy the maximum distance are the identity and the reversed permutations. Table 3 shows the effect of increasing the codeword size (in packets) on the maximum detectable E_D and correctable errors E_C.

For a complete view of the channel's bandwidth, one can combine the information given into a single formula that is based on k, l, p_L, and desired error-handling capabilities,

$$Capacity = \frac{H(\mathbf{p_L})}{k} \beta, \tag{3}$$

where $l = |L|$ is bounded as by Eq. (2), $H(.)$ is the entropy of a given probability distribution [11] and β is bits/packet. If maximum capacity is required given a specific error-handling capability regardless of the stealthiness requirement, the $H(\mathbf{p_L})$ term will be replaced with its upper bound $\log l$. This in turn can be equated with its limit from Eq. (2). The overall capacity is given by

$$Capacity \leq \frac{H(\mathbf{p_L})}{k} \times \log_2 \left[\frac{2^{\lceil \log_2 k! \rceil}}{\sum_{i=0}^{d_{min}} \binom{\lceil \log_2 k! \rceil}{i}} \right] \beta, \tag{4}$$

where d_{min} is given as before based on the error detection/correction capabilities.

5 Implementation

5.1 Implementation

The sender and receiver have to agree on an overt channel to host their covert communication. This includes defining the maximum allowable RD (or other similar metrics) to be used in setting the codeword usage distribution. Also, both parties have to agree on their choice of k, l, and L in order to successfully decode the sent information. In Fig. 1, the overall operation is displayed in a simple diagram showing the data flow from the sender's to the receiver's side. The sender translates his data to l distinct symbols that are encoded to permutations and send over the network. The receiver receives every k packets and translates their order into symbols that are translated back to be read.

In our implementation, we send fabricated packets over IP using the *libnet* packet handling library. The sequence number of each packet is included in the payload explicitly. In a final implementation, these packets will be sorted by an application-transparent intermediate packet filter. This can take place as a hook to the protocol stack, or simply as an intermediate box that acts as a proxy for the sender's main machine. On the other side of the channel, the receiver is currently implemented as a sniffer process implemented over the *libpcap* packet capture library, and actual packets are received by a dummy application that just reads the stream of packets without any further processing. In the final implementation, the sender can receive the covert communication by essentially the same mechanism as the current implementation, while making use of the overt channel with any application level process (e.g. FTP client). It is important to mention that the current implementation shortcuts do not change the packet behavior in the network, nor the transmission capacity. Therefore, our implementation is a valid testbed for the proposed covert channel. Fig. 2 shows the high level view of the channel deployment.

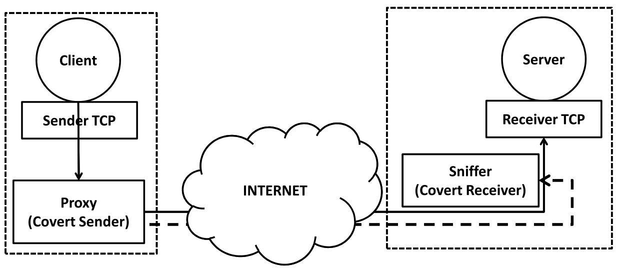


Fig. 2. The high level view of the channel deployment.

The sender buffers every k packets and sends them in the order that encodes the covert channel input data. The process at the receiver (see Fig. 3) is more complicated due to the fact that some long span ordering might take place. This might cause codewords to interleave complicating the decoding process. The receiver keeps track of the most recent incomplete codewords. Every received packet is added at the end of its corresponding codeword. These codewords are now represented only using their index, none of the packet's extra information is stored in this

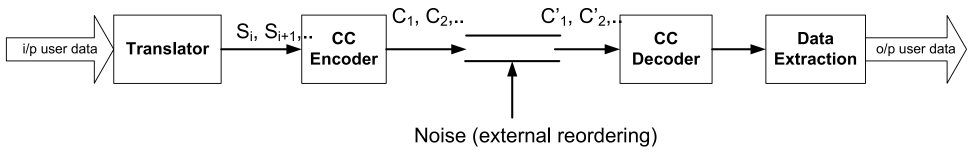


Fig. 1. Overall channel design. Starts with user data, translated to symbols that is decoded into permutation codewords to be sent to the receiver. The receiver decode the permutation into the original data symbols.

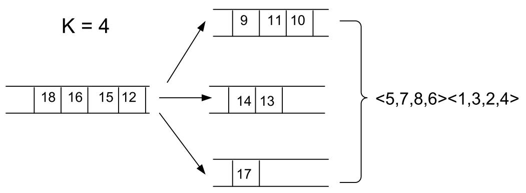


Fig. 3. The decoding process. Multiple partially received codewords are in temporary storage till completed, then forwarded to user.

phase. Once a codeword has all of its packets received, it will be checked for errors (and corrected if needed and possible), and saved into the output queue. If the completed codeword is not the earliest needed, it is kept in the queue of incomplete codewords till all earlier codewords are completed, then flushed together into the output stream. These steps are demonstrated in Algorithm 1.

Algorithm 1. Covert Channel Decode Packet

```
if First Call then
    Negotiate $k$, $L$, and $p_L$.
    Clear Packet Queues
    $indexStart \leftarrow 1$
    $codewordCount \leftarrow 0$
end if
$I \leftarrow$ Receive New Packet Index
$curCW \leftarrow \lfloor (I - indexStart)/k \rfloor$
if $curCW > codewordCount$ then
    $codewordcount += (curCW - codewordCount)$
end if
$CW[curCW] \Leftarrow ((I - indexStart) \bmod k) + 1$
while $size(CW[\lfloor indexStart/k \rfloor]) = k$ do
    flush($CW[\lfloor indexStart/k \rfloor]$)
    $indexStart \leftarrow indexStart + k$
    $codewordcount --$
end while
```

5.2 Delay Calculations

One has to observe that this channel, while it does not affect the packets themselves of the overt channel, it affects their timing in a way that might jeopardize the stealthiness of the covert session.

In order to reorder the packets at the proxy module (i.e., the covert channel sender) some packets have to be delayed until the packet indicated by the current codeword (permutation) arrives from the overt sender. For example, if the sequence to be sent is $< 3, 1, 2 >$, then the proxy has to buffer packets 1 and 2 and wait for the third one to arrive, then send the three of them in the desired sequence. The buffering adds a delay that can reveal the channel. The delay can be calculated to have an average value depending on k. The average delay a packet will experience will be given by $k * 8 * s_p/B$, where s_p is the packet size in bytes, and B is the average flow rate in bps. Let's denote this delay by D_{ave}.The same delay calculations will be observed when we discuss prevention in Section 6.

To solve the added delay issue, we can make the proxy work on a different "time zone". Once a packet is received by the proxy, it will be buffered anyway in order to shift the whole flow with the above mentioned delay. Even if the received packet is the next to be sent according to the coding scheme, the proxy will buffer it till it matches the shifted

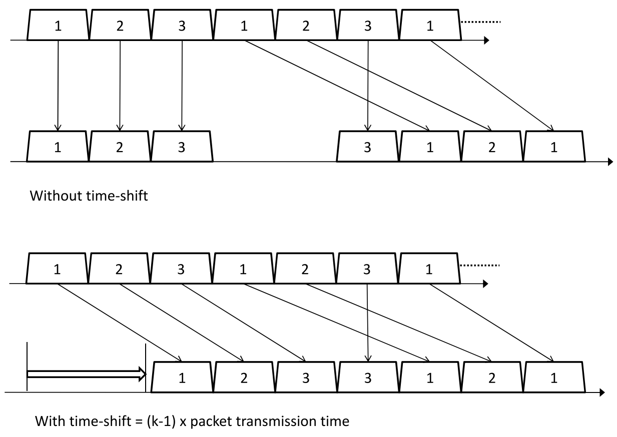


Fig. 4. Delayed/Scheduled sending for inter-packet gap removal. Example shows both with and without the added time shift for $k = 3$.

time frame. For example, assume we are building a channel that is based on $k = 3$ and using an overt channel of 1 kB packets with an initial rate of 1.6 Mbps. Sending packets 1 to 3 from the application at times 0, 5, 10 ms, will mandate a delay by the proxy for 10 ms (i.e., D_{ave}). Therefore, packets will be sent at 10, 15, 20 ms. If the covert codeword to be sent is $< 1, 2, 3 >$, then these are the delays to be applied. If we would like to send $< 3, 1, 2 >$, then we will send the third packet without delay, and the first and second packets will be delayed 15 ms each. Fig. 4 shows this example and how it maintains the back-to-back nature of the original transmission.

5.2.1 Implementing the Proxy with Delay

This tuned-up proxy module can be implemented using a simple priority queue. The key for the queue elements will be the packets' scheduled sending time. When a packet arrives at the proxy, its scheduled time will be calculated based on the current keyword, and once the time of the packet on top of the queue is less than the current time, the packet is extracted and sent onto the outgoing link.

The scheduled time of a packet i in a k-code will be

$$t_{sch} = t_{rcv} + D_{ave}\left(1 - \frac{(i \bmod k) - (i_{new} \bmod k)}{k}\right),$$

where i and i_{new} is the position of the packet in its original and after-encoding orders, respectively. As we can see, there is no need for t_{rcv} to be the exact receiving time of the packet. Measuring t_{rcv} with an error of less than one half packet transmission time is enough for correct operation.

On the long term, this time shift has the effect of giving the overt connection a perfect behavior with respect to its rate. In timing channels where inter-packet transmission time is the main media, the rate usually fluctuates slightly based on the hidden signal. This can be used against these channels to be detected, and or eliminated. The network PUMP [17], for example, will have zero effect on our channel even after adding the time shift.

6 Countermeasures and Preventions

In any proposed channel, it is hard to use any countermeasure approach without first assessing the parameter or media over which the cover transmission takes place. For example, using a technique that analyzes packet header statistics and patterns to stop a timing-channel (e.g. [29]) is guaranteed to fail. The proposed channel uses packet order, therefore, a countermeasure must either keep track and analyze such orders for each flow, or try to eliminate the media (i.e., the order-based channel).

Using a standard approach, as the network PUMP [17], will need to be extended to order as well as rate-control all in/outgoing flows to/from a certain host. Other detection techniques as those based on entropy [12] or other model-based approaches [10] will need to add extensive logging to keep track of the packet order as well. We will discuss targeted techniques to packet ordering in practical perspective, and how to implement them without a massive overhead and cost.

6.1 General Prevention Approaches

The main idea to prevent the existence of such covert channel is to make the channel too noisy to be practical or make the traffic too ideal to imitate. As our channel depend on the existence of some out-of-order packets, both approaches translates to the following. The first idea, to make the channel too noisy, necessitates adding more out-of-order packets in a way to make it very hard to send data reliably using our proposed approach. This in turn drop the channel capacity (lower values of l) to make it reliable, and in the same time increase the value of k to compensate for drop in capacity. These changes will affect the stealthiness of the channel in a way to jeopardize its initial design goal. The second approach, trying to eliminate the out-of-order phenomenon, will be more effective in eliminating the channels while being very expensive to implement. The reason is as follows. First, it can be very costly to order all passing by flows. Second, even if one want to selectively eliminate the out-of-order phenomenon for specific flows that contains the covert channel, one needs to first detect the existence of the channel. This itself is a difficult task due to the overwhelming overhead of monitoring the network traffic while the covert channel may be flooded by huge amount of "normal" traffic. If one tries to disrupt the covert channel by introducing more reordering (for example, reordering the whole sequence of packets), the TCP performance will be greatly degraded [35]. Third, if one want to avoid the heavy load on an individual device by trying to distribute the reordering load to numerous devices closer to edge nodes, one will suffer from another problem: massive deployment management and cost. Therefore, we will focus in the remaining part of our discussion on the first approach: adding more noise into the network in order to either push the covert channel parameters to being very demanding and thus lower its stealthiness, or lower its bandwidth enough to be impractical.

6.2 Implementation Options

There are several models to implement an efficient prevention mechanism for the proposed covert channel and its derivatives. In the following we will discuss a few of the possible deployment and implementation approaches for a prevention (rather than detection) mechanism.

6.2.1 Per Flow Noise Addition

In order to best add noise into targeted flows, we might need to keep track of suspected flow types (e.g. TCP is a better candidate for being used for these channels than UDP, and FTP more than HTTP due to added delay). Also, flow source-destination pairs that might be questionable can get more attention than others (e.g. sources in specific corporation, countries, ISPs, etc. Or destination that are recently opened, or targeted to hosts with a particular operating system, etc). While this approach gives the highest flexibility to discriminate between flows and allow the existence of different trust levels, and consequently, and differential behavior, it is very demanding processing and storage wise. Suppose the adversary take the following steps to add noise to the flow: 1) with probability $0 < q_1 <= 0.5$, delay current packet and send it after the next packet; 2) with a small probability $0 < q_2 <= 0.05$, randomly drop the current packet. In this case the covert codeword may be messed up. However, if one chooses an appropriate error correcting codeword, the covert channel will still work in the presence of the introduced noises. For example, if we take $k = 8$, then for different error correcting abilities and different values of q_1, Fig. 5 shows the ratio of correctable codeword based on the value of binomial probability distribution function. We can see that if the error correcting ability is high (able to correct six reorders) then the ratio of correctable codeword can reach more than 0.95. For a codeword size of $k = 8$ there are $8! = 40{,}320$ possible different values (permutations), which corresponds to a binary codeword of about 15 bits. To make the codeword to be able to correct six bit-errors will greatly degrade the bandwidth. However, this is not an issue for a covert channel. The covert channel may be only used to transfer very short messages (such as a Bot master command, which may be only tens of bytes) over a big flow. Even in the extreme case that we make the 15-bit codeword only represent one bit information, the resulting bandwidth is still acceptable in certain scenarios. For example, transferring a CD image (i.e., 700 MB or approximately 500 K packets) can easily be accompanied with 160 bits of a hash fingerprint, along with an additional 32 bits as a CRC of the hash itself or with error correction information. Note that 500 K packets can actually transfer 62,500 bits if we use $k = 8$ and assume that every codeword only represents one bit. Even if we pad the covert channel to contain 99 percent of zeros (i.e., no-reordering, to make the total reordering probability looks stealthy) and 1 percent of true information, it is still more than enough for the covert channel. To further increase the error correcting ability, we can add the redundancy in multiple codewords. For example, for every two codewords, we can make one codeword to represent true information, and

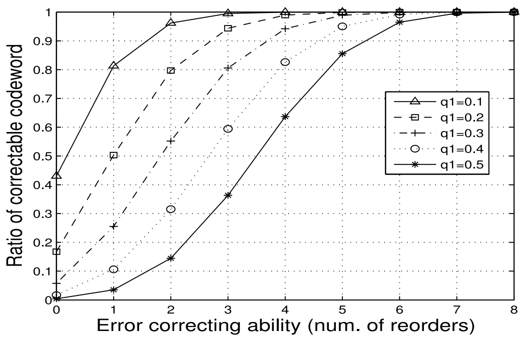


Fig. 5. Resilience of the error correcting code.

the other codeword represent error correcting code. The two-layered error-correcting (one layer inside individual codeword, and another layer across multiple codewords) can further increase the resilience of the covert channel.

If the adversary randomly drop some packets, then the effect is similar as introducing reordering. For example, in the case of $k = 8$, dropping the fourth packets is equivalent to adding four reorderings in the sequence since packet 4 will appear after packets 5, 6, 7 and 8. In our implementation, we always consider the first appearance of a packet as the criteria to determine reordering if a packet is received multiple times. The analysis of the error correcting ability in this case is similar. If the adversary intentionally inserts bogus packets that are simply duplicators of other packets and affect the packet order, then it can be handled by error correcting code as described above. If the bogus packets have wrong headers (specifically sequence numbers) then they can be discarded as the case in the protocol. However, if these packets are not distinguishable by the protocol, they will mess-up the application as well as our covert channel.

6.2.2 Overall Traffic Noise

The same idea can be applied regardless of flow. In other words, two consecutive packets might be swapped together even if they belong to different flows. This might seem in vain, but if it is applied close enough to the edge, and with a deep enough buffer, it will be effective with a simpler implementation as no per-flow information, or flow tables need to be created. A hybrid of this approach and the previous one can be built by splitting the aggregate flows into rough granularity bins. For example, we can just extract flows with connection-oriented protocols (e.g. TCP), while leaving all audio/video/udp traffic go smoothly without buffering or noise-adding.

6.2.3 Burst Noise for Delay Tolerant Flows

Random flows can be selected for extra security screening. The prevention process/module at the gateway/router will take aside a number of packets, possibly with sampling, to be investigated. An optional study can be put in place, for their order and whether they pose any statistical significance in this regard will be recorded for future manual analysis. These packets will be fully sorted, then forwarded towards their destination. This effectively will destroy the signal in a number of consecutive codewords in this flow (in case it was used for covert communication in the first place).

6.2.4 Delay Analysis of Prevention Implementation

- *Individual packet noise mechanisms.* Affected flows will suffer a delay for only those packets being swapped/ordered. The analysis will be the same as shown in the delay calculations in Section 5. We do not have the restriction to perform under the radar. However, we should not impose enough delay to the extent that will it affect user experience. For long term flows, like FTP data connections and web-download requests, we have the freedom to add relatively significant delays without jeopardizing the overall network performance.

TABLE 4
Analysis of SeR Rates in a Packet Index Trace with 14 Percent Reordering

Errors in codeword	0	1	2	3	4
prob. of errors	0.8732	0.0808	0.0376	0.0082	0.0002
CDF	0.8732	0.954	0.9916	0.9998	1.0

Codewords sent using $k = 4$.

- *Burst noise mechanism.* Assuming b packets have been withdrawn from a flow sending at B bits per second on a C bps link, we can reach the following figures. With a packet size of 1 K as a typical value, we see that there will be approximately a capture time of $b * 1\,\mathrm{K}/(B/8)$ seconds. If ordering the packets took negligible time, we reach a possible delay of $b * 1\,\mathrm{K}/(B/8)$ seconds affecting the first captured packet, and gets lower as we proceed with later captured packets. While this look similar to the previous case, we can always tune the capture mechanism to bypass the first few packets if they are already in order. For example, if 100 packets are to be sorted, and the first 30 packets were perfectly sorted then we will only delay packets from 31-100.

7 Evaluation

To evaluate the proposed covert channel, the main metrics to evaluate are the covert bandwidth per overt cost (e.g. packet, bit, or unit time), and the error rate of the covert transmission. The bandwidth can be calculated based on the channel parameters k, l, and the target reorder metric we have to stay below. This was addressed in Section 4.6. The error rate on the other hand, need evaluation experiments to imitate the behavior over each codeword transmitted not the aggregate effect as given by previously mentioned metrics (e.g. RD, RBD, etc).

The channel's most important design parameter is k, the codeword size in packets. As in Table 2, the effect of k on the maximum number of errors that can be detected is obvious. However, there shall be no need to go beyond the error correcting capabilities needed to successfully transmit through a specific channel, as that means a direct loss of bandwidth.

A single error in transmission is denoted by number of *shift error rate* (SeR) per codeword. SeR is equivalent to the single bit error rate used in ordinary channel analysis. It represents the probability by which a packet might shift one step. It can also be modeled as the probability of a single swap in position between two packets. An important property to verify is that multiple errors within the same codeword is a rare event. A trace with about 14 percent of out-of-order packets was used, and only the first 5,000 packets were analyzed for this experiment. The results show a sharp decline in the number of errors within the same codeword (for this experiment, $k = 4$). In Table 4 these results are shown. The RD of the trace used was $< 0.8545, 0.1343, 0.0105, 0.00054, \dots >$. This specific trace was chosen for its heavy reordering where its out-of-order measures given even exceeds those provided in [24]. This ensures that our conclusions will be conservative estimates

TABLE 5
Undetected Errors Using Only Two Codewords for Transmission ($l = 2$)

code word size (k)	probability of undetected errors	probability of uncorrected errors
2	0.084654	0.084654
3	1.233 E-03	0.019830
4	1.08 E-04	0.011152
5	3.50 E-05	0.007875
6	2.40 E-05	0.004656
7	7.00 E-06	0.000154

The trace used has an RD of 0.877, 0.114, 0.0074, 3.32E-4, 1.64E-5.

about the success of the channel in resisting external noise. The results show that if the code used is able to correct a single error (which is feasible for all $k \geq 3$) then 95.4 percent of codewords will be received successfully.

Another important dimension is how accurate can our transmission be if we sacrificed the bandwidth for maximum reliability. In other words, if we used only two permutations from the $k!$ possible ones, what will the percentage of undetected errors be. For the first few values of k[1], Table 5 shows the percentage of errors goes undetected when applied for the same trace. Although the probabilities of undetected errors are considerably low, only detecting the error will require retransmission which is not possible in all cases. Obviously, the probability of errors that will go uncorrected (using the same trace and coding) will be higher. However, the values shown in Table 5 still show very high reliability despite the strict requirements of correcting errors rather than stopping at detection.

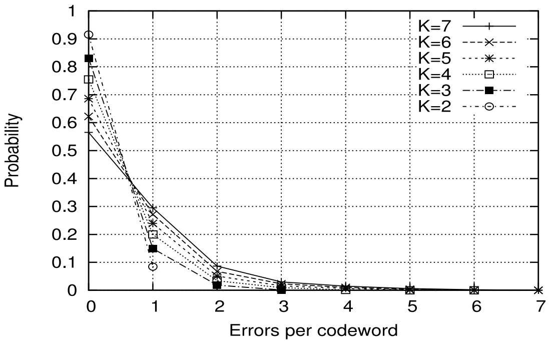
Fig. 6a shows the effect of increasing k on the distribution of errors (SeR). Lower values of k give the highest probability of error-free codewords, but suffers low decay rate for the SeR distribution. Therefore, if moderate error-correction capability is provided, higher k's will prove useful. The cumulative error distribution, illustrated in Fig. 6b, shows this effect. For example, if a code with $d_{min} = 3$ is used (feasible for $k \geq 3$), the reliability of the transmission increases to above 90 percent for $k = 3, 4, 5$ and increases to higher than 97 percent if three or less errors are corrected.

The effect of increasing the codeword size while fixing the error handling capabilities is presented in Fig. 7. We can see that the capacity per packet can increase from almost no information transferred (~ 0 bits/packet) to as high as 2.5 bits/packet by increasing the value of k. Each curve represents a specific error capability represented in the minimum distance between codewords (d_{min}). On the other hand, the effect of increasing d_{min} on the capacity is shown in Fig. 8. The drop is clear but gradual enough to give a wide range of valid design settings.

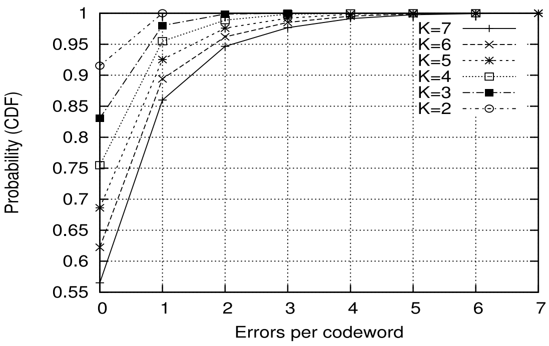
7.1 PlanetLab Deployment

We deployed our implementation over PlanetLab, which provides a faithful approximation of Internet. We use

1. From previous studies on extent of reordering, and its effect on TCP protocol buffers; values of $k > 10$ are not encouraged as it will severely degrade the stealthiness as well as the performance of the host application.



(a) Probability of different errors (SeR) for various values of K.



(b) Cumulative probability of different errors (SeR) for various values of K.

Fig. 6. At $k = 2$, the channel will not be able to correct/detect any errors but is included here for completeness.

PlanetLab nodes covering the six inhabited continents. Every continent was represented by at least one node, with a total of 10 nodes in the whole experiment. In our experiment, every pair of PlanetLab nodes send TCP packets with size of 1,500 bytes to each other, with the rate of 100 packets per second (which results in a data rate of 1.2 Mbps). The sending and receiving nodes apply the algorithm described in Section 5. Although, the deployment was world-wide, the results did not represent any out of the expected results. The focus was on evaluating the error rate of our covert channel for various values of k when used for pairs varying geographical proximity. The codeword selection model was based on the statistics we collected from those pairs, as well as those learned from studies in the literature [22], [24].

From our measurements, we found two basic parameters that affect our channel's success and reliability. The first

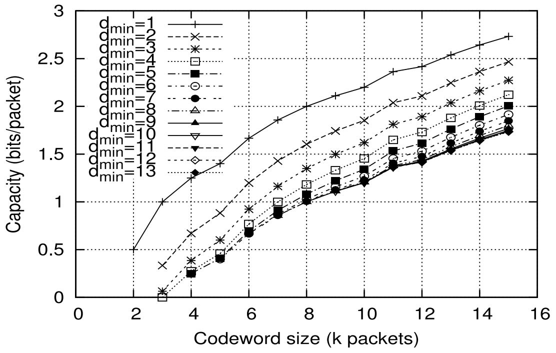


Fig. 7. Effect of increasing codeword size (k) while fixing minimum distance between codewords (d_{min}). ($d = E_D + 1$ or $d = 2E_C + 1$).

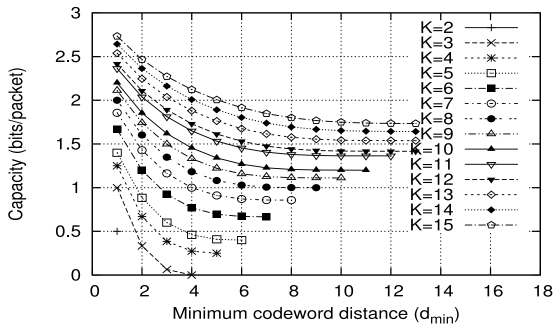


Fig. 8. Effect of increasing minimum codeword distance on the channel capacity per packet.

parameter is the number of hops between the two nodes: the more the hops in the connection the longer the out-of-order packet sequences. However, this does not necessarily guarantee an error in transmission. For example, if a block of 20 packets have been shifted 30 locations in the overt flow due to hiccup in routing status (and assuming the TCP connection did not fail due to the excessive delay), we will find errors only in those codewords that crossed the boundaries of the shifted block. If a codeword has all of its packets shifted together, then the covert channel receiver will see this as an error-free transmission. A higher number of hops, usually, translates to a higher RTT. This higher RTT scales up all out-of-order mishaps causing shifts to be in the multi-codeword range. In our observations, shifts affecting blocks of more than 10 packets happened exclusively in connections including a node in Africa and/or Australia.

Other parameters were parallelism, load-balancing and traffic slicing that take place in intermediate nodes. Although, we do not have access to the configuration and hardware capabilities of all nodes, we found that some routes have suffered of these effects more than others (an order of magnitude more out-of-order packets is quite possible). Such high reordering rate took place with connections between pairs like (Virginia tech-Princeton university). In [5], [6], [21], these device properties were the main reason accused of causing short term packet reordering, and it is hypothesized that their effect will increase as high-end routers make more use of new advances in multi-core processors, and load balancing mechanisms.

7.2 Effectiveness of Prevention Mechanism

The prevention mechanism presented in Section 6, was evaluated using the per-flow perturbation implementation. Clearly, this is the most demanding implementation resource-wise, but it is the easiest to analyze, understand and model. By varying the depth of the shift, and its probability, we arrived at the following results.

The three main factors selected in the evaluation are k, α, and d. The first parameter is the codeword size in packets. The other two are the probability by which a packet is selected to be shifted in time (i.e., moved to a later order), and the distance by which such a packet can be shifted (i.e., measured in number of packets it jumps over), respectively. The results were obtained by sending 10^6 packets with each combination between varying nodes. The results shown are a sample of those obtained between east-west coast node

TABLE 6
Effect of Prevention Depth on Error Rate of Covert Channel

$d \downarrow \setminus k \rightarrow$	2	3	4	5
1	0.982%	1.974%	2.935%	3.930%
2	1.002%	1.935%	2.930%	3.907%
3	1.004%	1.983%	2.975%	3.904%
4	0.987%	1.915%	2.872%	3.910%
5	0.969%	1.945%	2.898%	3.862%
10	0.969%	1.942%	2.880%	3.83%

With α equals 0.01 and $l = k!$ (i.e., $E_D = E_C = 0$).

pairs (UCLA and Princeton). We opted not use extreme ordering links, or long-range ordering pairs to be conservative on our estimates of the success of our channel in real life. For the range of values selected for each parameter, we choose to have a reasonable k as can be used by the channel designer/user, as well as responsible values for d and α. A higher value for d would have affected the performance of the receiving end. Besides, a higher value of d and α will reveal the existence of the prevention technique as well as force it to be beyond the range of naturally occurring out-of-order packets. We chose not to go beyond a constant factor of what was measured by other researchers as in [3], [4], [6], [16], [26]. The value selected for α ranged from 0.1 to 5 percent. This enabled the implementation to be realizable without excessive overhead on the node in which the prevention mechanism will be deployed.

In Table 6, we can see that given a fixed α, the probability of an error being forced at the receiving end increases linearly with the codeword size. This observation matches our expectations, because the probability for a codeword being hit will necessarily increase as its span increase. The effect was quite dramatic due to the lack of use of any error detection or correction ($l = k!$). Also, note that there was no effect for the depth d, as it does not matter how far a packet is shifted when the smallest shift will cause an error (again due to the lack of error detection or correction).

We focus on the relation between d and k in Table 7. By enabling some error detection and correction capability, namely $E_C = 1$, we saw the effect of error correction capabilities in circumventing our noise-adding technique. We can see that for a depth of 1, the success-rate of our prevention technique was very low regardless of the codeword size ($< 0.05\%$). For higher values of d, the effect was as we saw in the previous table, where the error introduced into the channel was linear with code-word size. Therefore, we conclude that the channel owners can significantly lower the effect of

TABLE 7
Effect of Prevention Power on Error Rate of Covert Channel

$d \downarrow \setminus k \rightarrow$	2	3	4	5
1		0.005%	0.02%	0.041%
2		0.472%	1.006%	1.475%
3	N/A	0.6525%	1.323%	1.966%
4		0.75%	1.526%	2.170%
5		0.7625%	1.586%	2.425%

With α equals 0.01 and $E_C = 1$. $k = 2$ does not permit E_C to have a value > 0.

TABLE 8
The Effect of α on the Success of the Prevention Mechanism

l	$\alpha \downarrow \setminus d \rightarrow$	2	3	4	5	10
$k!$	0.001	0.283%	0.260%	0.276%	0.294%	0.245%
	0.005	1.483%	1.450%	1.473%	1.450%	1.356%
	0.01	3.037%	2.834%	2.976%	2.896%	2.847%
	0.05	13.42%	13.573%	13.689%	13.756%	13.626%
2	0.001	0.0%	0.0%	0.033%	0.05%	0.053%
	0.005	0.0%	0.0%	0.146%	0.273%	0.4%
	0.01	0.0%	0.01%	0.323%	0.5%	0.7%
	0.05	0.01%	0.306%	1.739%	2.573%	3.723%

In this experiment, $k = 4$, and l is taken to be $k!$ and 2.

our prevention mechanism by choosing $E_C > d$. However, they will suffer from the serious drop in bandwidth as shown in Fig. 8. Another problem, is that the stealthiness of the channel will be affected as they try to increase k in order to accommodate higher correction power.

In the following table (Table 8), we show how increasing α will affect the success of the prevention system. The first section of the table, we show the results without any error detection/correction capabilities, while in the second, we show the results with the highest possible E_C and E_D by using $l = 2$. All entries in the table were obtained for a moderate setting of $k = 4$. As we can see, and as expected, the higher the value of α the stronger the effect. However, there is an important observation, which is the cap we get on our success rate regardless of the value of the depth parameter (i.e., d). This can be explained by the way the decoding mechanism is implemented, which removes any effect of shuffling packets belonging to different codewords. For example, delaying the last packet of a codeword for any arbitrary duration does not affect the decoding of the codeword, as relatively to its original codeword, the packet still comes last. Also, we see that for a very cautious channel operator, it will still be very hard to completely destroy his packet. For example, if he/she selected $k = 10$ and used a very skewed distribution of codewords (by assuming a very conservative reordering metric for which to solve RD_t, as shown in Section 4), as well as imposing a very high error correction capabilities (e.g. $E_C = 5$), it will be definitely impossible to succeed in completely preventing the channel from bypassing our prevention mechanism. If such cases exist, and a complete solution is indeed needed, then a complete reorder of traffic will be essential.

8 Limitations

The proposed covert channel has the following limitations. First, it can only be used to transmit small amount of auxiliary data, but cannot be used to transfer data of large size. Transferring big amount of data in the covert channel is very inefficient and easy to be detected. Second, the proposed covert channel cannot replace other cryptographic approaches such as encryption. As long as that there is no pre-agreed encryption key between the sender and receiver, a powerful eavesdropper will eventually recover all covert channel information, since the eavesdropper can just repeat the decoding procedure. However, this applies to any covert channel that does not use pre-agreed keys. Third, the stealthiness of the covert channel exists in certain limited aspect. The covert channel significantly increases the bar for detection, but cannot prevent any powerful eavesdropper or monitor to discover the covert channel. Our covert channel mimics natural reordering by making the reordering probability similar to that of the natural reordering. If the estimation of probability of natural reordering is not precise, it is possible for the adversary to detect the existence of the channel after observing the channel long enough. Also, since we only consider the metric for overall reordering, it is possible to detect the covert channel after statistical analysis for patterns of "bursting" reorderings in the codeword. We plan to further investigate this in future research. Another limitation of the scheme is that it cannot be applied for applications with large delay deviation since the added delay will cause too much performance degradation.

9 Conclusion and Future Work

In this paper, we pursue the possibility of building covert channels that use the packet order as the medium for transmitting hidden signals. Packet reordering is a normal behavior in the Internet and it is too hard to closely monitor and unlikely to raise suspicions. Moreover, efforts to eliminate it will not be easily implemented due to high cost and lack of incentive. Also, the main causes behind this phenomenon are not likely to vanish nor decrease in the near future.

Our proposed channel is designed based on the idea of assigning different symbols to the different permutations a number of consecutive packets can have. By using only a subset of the permutations, we were able to add error detection and correction capabilities. Subsets used are rotated to evade detection. Any adversary needs a huge cost to detect the covert channel due to the tremendous overhead to buffer and sort the packets among huge amount of background traffic. The codewords themselves are selected based on the traffic characteristics to follow closely the innate reordering characteristics of the host channel. The outcome was almost identical to the target as given by the reordering metric (RD in our evaluation). Also, by varying the codeword input distribution to the channel, as well as the subset of permutations to use, our channel can be extremely robust and resilient to external reordering effects. The channel's built-in error detection/correction capabilities increased the correctly received codewords to more than 98 percent in typical operating parameters. The evaluation using packet traces showed error rates as low as 0.1 percent if the appropriate coding scheme is selected. As one example for the capacity and error correcting ability of the channel, 30 packets are needed per bit of covert communication, if we require that one error is correctable for every four bits, with codeword size $k = 2$, and "natural" probability of pairwise reordering $q = 0.1$. Many additions can be used to enhance the proposed channel. First, other coding schemes, e.g. cyclic codes, can be used to enhance the channel's resilience to errors. Second, we plan to investigate more dynamic environments where channel properties need to be continuously communicated between both parties. This requires adding some learning/monitoring capabilities to the logic at both ends. Third, we can use more than one technique so that we can select the most optimal one for the target channel will enhance the channel resilience and stealthiness. Fourth, we can apply hybrid metrics of packet reordering to better evade detectors. Fifth, we plan to evaluating the possibility of detecting the channel by monitoring the transport layer protocol performance. Sixth, we can design more efficient and effective protection techniques as well as detection mechanisms.

REFERENCES

[1] K. Ahsan and D. Kundur, "Practical data hiding in TCP/IP," in *Proc. Workshop Multimedia Security ACM Multimedia*, 2002.

[2] V. Anantharam and S. Verdu, "Bits through queues," *IEEE Trans. Inf. Theory*, vol. 42, no. 1, pp. 4–18, Jan. 1996.

[3] C. M. Arthur, A. Lehane, and D. Harle, "Keeping order: Determining the effect of TCP packet reordering," in *Proc. 3rd Int. Conf. Netw. Serv.*, Washington, DC, 2007, pp. 116–122.

[4] T. Banka, A. A. Bare, and A. P. Jayasumana, "Metrics for degree of reordering in packet sequences," in *Proc. 27th Annu. IEEE Conf. Local Comput. Netw.*, Washington, DC, 2002, pp. 0333–0340.

[5] J. Bellardo and S. Savage, "Measuring packet reordering," in *Proc. 2nd ACM SIGCOMM Workshop Internet Meas.*, 2002, pp. 97–105.

[6] J. C. R. Bennett, C. Partridge, and N. Shectman, "Packet reordering is not pathological network behavior," *IEEE/ACM Trans. Netw.*, vol. 7, no. 6, pp. 789–798, Dec. 1999.

[7] K. Borders and A. Prakash, "Web tap: Detecting covert web traffic," in *Proc. 11th ACM Conf. Comput. Commun. Security*, New York, NY, 2004, pp. 110–120.

[8] R. Browne, "An architecture for covert channel control in realtime networks and multiprocessors," in *Proc. IEEE Symp. Security Priv.*, 1995, pp. 155–168.

[9] S. Cabuk, C. E. Brodley, and C. Shields, "Ip covert timing channels: Design and detection," in *Proc. 11th ACM Conf. Comput. Commun. Security*, New York, NY, USA, 2004, pp. 178–187.

[10] S. Cabuk, C. E. Brodley, and C. Shields, "Ip covert channel detection," *ACM Trans. Inf. Syst. Security*, vol. 12, no. 4, pp. 22:1–22:29, Apr. 2009.

[11] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. New York, NY, USA: Wiley, 1991.

[12] S. Gianvecchio and H. Wang, "Detecting covert timing channels: An Entropy-based approach," in *Proc. 14th ACM Conf. Comput. Commun. Security*, New York, NY, USA, 2007, pp. 307–316.

[13] A. Houmansadr and N. Borisov, "Coco: Coding-based covert timing channels for network flows," in *Proc. 13th Int. Conf. Inf. Hiding*, 2011, pp. 314–328.

[14] W. Hu, "Reducing timing channels with fuzzy time," in *Proc. IEEE Comput. Soc. Symp. Res. Security Priv.*, 1991, pp. 8–20.

[15] M. Owens, "A discussion of covert channels and steganography," *SANS Institute InfoSec Reading Room*, 2002.

[16] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley, "Measurement and classification of out-of-sequence packets in a tier-1 IP backbone," *IEEE/ACM Trans. Netw.*, vol. 15, no. 1, pp. 54–66, Feb. 2007.

[17] M. H. Kang, I. S. Moskowitz, and D. C. Lee, "A network pump," *IEEE Trans. Softw. Eng.*, vol. 22, no. 5, pp. 329–338, May 1996.

[18] Y. Liu, D. Ghosal, F. Armknecht, A. Sadeghi, S. Schulz, and S. Katzenbeisser, "Robust and undetectable steganographic timing channels for i.i.d. traffic," in *Proc. 12th Int. Conf. Inf. Hiding*, Berlin, Germany, 2010, pp. 193–207.

[19] X. Luo, E. W.W. Chan, and R. K.C. Chang, "Cloak: A Ten-fold way for reliable covert communications," in *Proc. 12th Eur. Conf. Res. Comput. Security*, 2007, vol. 4734, pp. 283–298.

[20] S. J. Murdoch and S. Lewis, "Embedding covert channels into TCP/IP," in *Proc. 7th Int. Workshop Inf. Hiding*, 2005, pp. 247–261.

[21] V. Paxson, "End-to-end routing behavior in the internet," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 5, pp. 41–56, Oct. 2006.

[22] N. M. Piratla, A theoretical foundation, metrics and modeling of packet reordering and methodology of delay modeling using inter-packet gaps, Ph.D. thesis, Dept. of Electrical and Comput. Eng., Colorado State Univ., Fort Collins, CO, USA, 2005.

[23] N. M. Piratla and A. P. Jayasumana, "Reordering of packets due to multipath forwarding-an analysis," in *Proc. IEEE Int. Conf. Commun.*, 2006, vol. 2, pp. 829–834.

[24] N. M. Piratla and A. P. Jayasumana, "Metrics for packet reordering: A comparative analysis," *Int. J. Commun. Syst.*, vol. 21, no. 1, pp. 99–113, Jan. 2008.

[25] N. M. Piratla, A. P. Jayasumana, and A. A. Bare, "Reorder density (RD): A formal, comprehensive metric for packet reordering," in *Proc. Int. Federation Inf. Process. Netw. Conf.*, 2005, pp. 78–79.

[26] N. M. Piratla, A. P. Jayasumana, A. A. Bare, and T. Banka, "Reorder Buffer-occupancy density and its application for evaluation of packet reordering," *Comput. Commun.*, vol. 30, no. 9, pp. 1980–1993, 2007.

[27] N. Provos and P. Honeyman, "Hide and seek: Introduction to steganography," *IEEE Security Priv.*, vol. 1, no. 3, pp. 32–44, May 2003.

[28] C. Savage, "A survey of combinatorial gray codes," *SIAM Rev.*, vol. 39, pp. 605–629, 1996.

[29] S. H. Sellke, C. Wang, S. Bagchi, and N. B. Shroff, "TCP/IP timing Channels: Theory to Implementation," in *Proc. IEEE INFOCOM*, 2009, pp. 2204–2212.

[30] G. Shah, A. Molina, and M. Blaze, "Keyboards and covert channels," in *Proc. USENIX Security Symp.*, 2006, pp. 59–75.

[31] S. H. Son, R. Mukkamala, and R. David, "Integrating security and real-time requirements using covert channel capacity," *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 6, pp. 865–879, Nov. 2000.

[32] S. Voloshynovskiy and F. Deguillaume, "Information-theoretic data-hiding: Recent achievements and open problems," *Int. J. Image Graph.*, vol. 5, no. 1, pp. 1–31, 2005.

[33] S. Voloshynovskiy, F. Deguillaume, O. Koval, and T. Pun, "Information-theoretic data-hiding for public network security, services control and secure communications," in *Proc. 6th Int. Conf. Telecommun. Modern Satellite, Cable Broadcasting Serv. (TELSIKS)*, 2003, pp. 1–17.

[34] X. Wang, S. Chen, and S. Jajodia, "Network flow watermarking attack on low-latency anonymous communication systems," in *Proc. IEEE Symp. Security Priv.*, 2007, pp. 116–130.

[35] W. Wu, P. Demar, and M. Crawford, "Sorting reordered packets with interrupt coalescing," *Comput. Netw.*, vol. 53, no. 15, pp. 2646–2662, Oct. 2009.

[36] W. Yu, X. Fu, S. Graham, D. Xuan, and W. Zhao, "Dsss-based flow marking technique for invisible traceback," in *Proc. IEEE Symp. Security Priv.*, 2007, pp. 18–32.

Adel El-Atawy received the PhD degree in computer science from Depaul University. He is currently with Google, Inc. His research interests include network configuration verification and information theory. He is a member of the IEEE.

Qi Duan received the PhD degree in computer science from University at Buffalo. He is currently a senior research associate in the University of North Carolina at Charlotte. His research interests include network security and information theory. He is a member of the IEEE.

Ehab Al-Shaer received the PhD degree in computer science from Old Dominion University. He is currently a professor in the University of North Carolina at Charlotte. His research interests include network security and formal methods. He is a member of the IEEE.