NetWarden: Mitigating Network Covert Channels without Performance Loss

Jiarong Xing Adam Morrison Ang Chen
Rice University

Abstract

Network covert channels are an advanced threat to the security and privacy of cloud systems. One common limitation of existing defenses is that they all come at the cost of performance. This presents significant barriers to their practical deployment in high-speed networks. We sketch the design of NetWarden, a novel defense whose key design goal is to *preserve TCP performance* while mitigating covert channels. The use of programmable data planes makes it possible for NetWarden to adapt defenses that were only demonstrated before as proof of concept, and apply them at linespeed. Moreover, NetWarden uses a set of *performance boosting* techniques to temporarily increase the performance of connections that have been affected by channel mitigation, with the ultimate goal of neutralizing its impact on performance. Our simulation provides initial evidence that NetWarden can mitigate several covert channels with little performance disturbance. As ongoing work, we are working on a full system design and implementation of NetWarden.

1 Introduction

Network covert channels are an advanced class of security threats to cloud systems. An attacker can exfiltrate secret information from compromised VMs via channels that are not intended for carrying data. Two typical classes of network covert channels are *timing* channels [7, 8, 15, 24, 27, 34, 38], which modulate packet timing, and *storage* channels [4, 9, 16, 18, 22, 29, 33], which embed data inside packet headers. As an example of the former, an attacker could use large and small inter-packet delays (IPDs) to encode ones or zeros in a secret message [8]. As an example of the latter, an attacker could embed secret data in the TCP sequence number [9] or ACK [28, 29] fields. Covert channels can leak data without raising suspicion from a firewall that typically only inspects packet payload. Existing work has shown that both timing and storage channels can propagate over long distances [8,28], and the TCSEC criteria (Trusted Computer Security Evaluation Criteria) require protection against both of them [11].

Over the years, researchers have developed a variety of solutions to detect and mitigate network covert channels [5, 8, 9, 14, 30, 32]. For instance, in order to detect timing channels, existing detectors rely on statistical properties of legitimate traffic IPDs, such as first-order statistics (e.g., mean and variance) [8], or higher-order entropy [14]. In order to detect storage channels, existing detectors analyze packet header fields that could be used to encode data (e.g., TCP sequence number [9]) and look for anomalies. Upon detection, a range of mitigation techniques can then be applied, including adding random delays to packet transmission [5, 14], or setting a header field to a controlled value [9, 30].

It is perhaps unsurprising that no detector—whether for timing or storage channels—can achieve 100% accuracy. This is because the characteristics of network traffic can be highly non-deterministic, both in terms of timing and header values, as they depend on subtle interactions between the hosts and the network. For instance, a timing channel detector may raise a false alarm if IPDs suddenly increase, but this may be caused merely by congestion. As a trickier example, consider the partial ACK channel [28]. Suppose that a connection has transmitted N bytes of data, an attacker could send a partial ACK to acknowledge the n-th byte, leaking a secret $\delta = N - n$. As long as n is chosen to be in a valid range (i.e., between the last ACK and N), the channel will successfully hide itself in the permitted behaviors of TCP. Note that this example reveals a deeper problem—designing a perfect detector is close to impossible without having complete visibility into what actually happened on all nodes (e.g., whether the [n..N] bytes have been successfully received). However, if complete visibility were attainable, then arguably we should not have security holes to begin with. In fact, the partial ACK channel was discovered ten years ago [28], but we are not aware of a working defense even today.

To compensate for detection inaccuracy, we could be more aggressive in mitigation—e.g., applying a blanket defense to all connections that *might* contain a channel. The obvious consequence here is performance degradation. Since most connections may be benign, an aggressive defense may unduly penalize legitimate flows. For instance, one extreme defense against partial ACKs is to drop all ACK packets that do not acknowledge N; but what if data from n to N has actually been dropped or garbled? A slightly milder defense, as proposed later in this work, would be to rewind the ACK number to the last-seen value; but it may still cause unnecessary retransmissions, i.e., performance penalty. Overall, we are faced with a concrete instance of the more general phenomenon that security comes with the cost of performance. Unfortunately, performance is a non-negotiable requirement in datacenters.

We sketch the design of a novel network primitive called NetWarden, which is a broad-spectrum defense against network covert channels in a *performance-preserving* manner. The starting observation in NetWarden is that *programmable*

data planes in modern switch hardware provides a feasible basis towards efficient and practical channel defense. Concretely, programmable data planes can perform per-packet operations over header fields, which enables NetWarden to inspect and modify headers for storage channel mitigation without stalling the traffic. Programmable data planes can also support sophisticated data structures directly in switch hardware, which NetWarden leverages to monitor each connection and discover problematic protocol behaviors (e.g., abnormal IPDs, incorrect ACKs). Moreover, both features can run at linespeed with nanoseconds of extra delay.

Building upon programmable data planes as a starting basis, NetWarden also uses a set of *performance boosting* techniques that can counteract the performance penalty due to channel defense. These techniques are inspired by a recent result in the security community, which has shown that the TCP congestion control mechanism can be manipulated by an attacker to artificially inflate the sending rate [19]. NetWarden borrows similar techniques from this attack result, but uses them for a very different goal instead—boosting the performance of connections affected by channel defense.

Concretely, NetWarden uses *ACK boosting* and *receive window boosting* to increase the sending rate of a connection. ACK boosting creates the illusion of a fast network, and receive window boosting creates the illusion of a high-performance receiver, ramping up the sending window of the data source. NetWarden also caches excess packets temporarily at the switch; should any packets be dropped on their way to the receiver, NetWarden can still serve the data to the receiver as a proxy. NetWarden then uses them in combination with defense techniques that usually lead to performance degradation—by borrowing and refining existing defenses and customizing them to programmable data planes—so that they neutralize each other's effects.

This paper is intended as a first step towards a defense that can mitigate network covert channels while preserving network performance. We sketch the initial design of NetWarden and provide preliminary evidence on its feasibility. We then summarize the open research questions and conclude with our ongoing exploration.

2 Overview

In this section, we describe more background on network covert channels, introduce building blocks for a practical defense, and then give an overview of NetWarden.

2.1 State of the art

Covert timing channels. Since Lampson proposed the notion of covert timing channels in 1970 [23], researchers have demonstrated that network covert channels can exfiltrate information over a long distance by modulating IPDs [8, 14, 27]. The simplest channel, for instance, uses large and small IPDs to transmit bits of one or zero. An attacker can further increase the stealth of a channel by mimicking the IPD characteristics of regular traffic [14].

Detection. Timing channel detectors rely on statistical tests of IPD distributions. A simple example is shape test [8], which uses first-order statistics (e.g., mean, variance) of IPDs to distinguish covert and normal traffic. More advanced detectors could also use the distribution [32], regularity [8], or higher-order entropy [14] of IPDs for channel detection. However, practically, these detectors can only be used in an offline manner, as streaming high-speed traffic through these statistical detectors in real time would cause enormous overhead.

Mitigation. In principle, mitigating timing channels is easy. A defense could be to inject random delays to network traffic, with the goal of disturbing the IPD modulation [5]. However, this is only practical if detectors can precisely pinpoint flows for delay randomization. Otherwise, false positives in statistical detectors would cause normal flows to be penalized.

Covert storage channels. The simplest storage channels (Type-I) can encode data in optional or unused TCP/IP header fields, such as Type of Service (ToS), Urgent Pointer, and IP identification (IPID) fields [12]. More advanced channels (Type-II) encode data in header fields that are essential for protocol correctness, such as the TCP initial sequence number [9]. A particularly tricky class of channels (Type-III) can hide themselves in the inherent non-determinism of network traffic, such as the partial ACK channel [28].

Detection. A common strategy for detection is to inspect all header fields, and look for the existence of header fields that are rarely used or contain suspicious values. However, the need to inspect (and potentially modify) all packet headers already makes most software-based detectors impractical.

Mitigation. Temporarily shelving performance concerns, Type-I channels can be mitigated by setting optional header fields to controlled values. Type-II channels can also be mitigated using a similar strategy, but the defense needs to be stateful and apply the same actions to all packets in the flow to maintain protocol correctness (e.g., adding a fixed offset to all TCP sequence numbers [9]). Type-III channels are the hardest, as they exploit the non-determinism in network traffic.

Limitations of existing defenses. To summarize, although covert timing and storage channels have been studied extensively, only proof-of-concept detection and mitigation systems exist, and their deployment is hindered by a) the need to perform per-packet operations at linespeed (for detection), and b) the performance penalty a naïve countermeasure would incur (for mitigation). As a result, the feasibility of designing an efficient, practical, and performance-preserving defense remains an open research question.

2.2 Towards a practical defense

We aim to answer this research question by designing NetWarden, which uses a combination of techniques to achieve a practical covert channel defense.

Technique #1: Using programmable data planes. Our first observation is that the advent of programmable data planes provides a suitable basis for NetWarden. A recent trend in datacenters is that the networking hardware is becoming increasingly programmable, both at the hosts (e.g., smart NICs [26]) and in the network (e.g., programmable switches [6]). Programmable data planes provide a set of new features that were originally designed for better *networking*, but we recognize that the same features map surprisingly well to the requirements of covert channel defense.

First, this new type of hardware can perform per-packet header operations at linespeed. The packet processing pipeline on emerging switches can be programmed using high-level languages, such as P4 [2], to specify custom match/action behaviors and perform header modifications. This can be used as a building block for storage channel defense. Second, programmable data planes have a fine-grained timestamping facility. This was originally designed for achieving higher network visibility for diagnosis, but it also provides useful support for timing channel detection. Finally, they support sophisticated data structures that can sustain linespeed reads and writes using stateful registers. We can use this feature for per-connection monitoring and precise channel mitigation.

Technique #2: Fastpath/slowpath defense. Since programmable data planes were *not* designed with channel defense in mind, there are many needed functionalities they do not directly support. For instance, the timestamping facility only provides packet timestamps, but statistical tests over IPDs are not implementable in switch hardware. Also, as we will discuss later, the performance boosting defenses also cannot be supported in the data plane. Therefore, another design principle of NetWarden is to offload as many primitives as possible to the data plane as a fastpath defense, and then perform a slowpath defense on the switch control plane for the rest. The control plane of modern switches has general-purpose CPUs and RAMs, and it is quite powerful. As a point of comparison[1], the available CPU cycles per second and the amount of RAM on a popular switch model [3] match these of a `T3.large` VM instance on Amazon EC2 [1]!

Technique #3: Performance boosting. Finally, as we motivated before, NetWarden specifically designs for a key goal of *performance-preserving defense*. In addition to customizing existing (and performance-degrading) defenses for programmable data planes, we also design performance-boosting defenses to neutralize the overall performance impact. These defenses are more expensive to perform, because they need to involve the control plane and cannot run at linespeed. However, NetWarden only needs to apply them to a small number of connections in a "default-off" fashion.

[1] Wedge 100BF-32X switch: 4×1.6 GHz cores, 8GB RAM; Amazon EC2 T3.large VM instance: 2×2.5 GHz vCPUs, 8GB RAM.

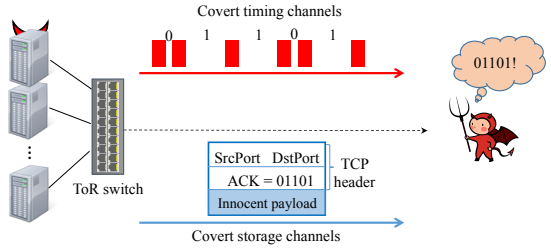


Figure 1: NetWarden can be deployed in a ToR switch as a broad-spectrum defense against network covert channels.

NetWarden. Combining these techniques, NetWarden can be easily deployed on a Top-of-Rack (ToR) switch to protect a rack of machines that host sensitive data (e.g., file servers). NetWarden is also a broad-spectrum defense in that it can serve as a general-purpose platform to support a wide range of existing and new defenses. Figure 1 shows a concrete example scenario for using NetWarden.

3 Solution Sketch

Next, we sketch a tentative design of NetWarden, describe how it adapts existing defenses for programmable data planes, and introduce its performance boosting techniques. Figure 2 shows the fastpath and slowpath components of NetWarden, which collaboratively detect and defend against covert channels while preserving network performance.

3.1 The fastpath defense

The fastpath consists of three components: a) connection monitoring, b) IPD characterization, and c) storage channel defenses (that degrade performance).

Connection monitoring. NetWarden uses a per-connection monitoring data structure in the fastpath, which is inspired by the Dapper [13] system that performs TCP monitoring for performance diagnosis. The data structure organizes TCP connections in a key/value store, where the key is a TCP connection's four tuples (i.e., source/destination IPs and ports), and the value is an index to a set of register arrays. Using this index, we can further write into or read from stateful registers that record the TCP state for each direction of this connection, such as a) the highest ACK and SEQ numbers, b) the receive window sizes, and c) IPDs observed for this connection. Every time NetWarden receives a packet, it uses its four tuples as key to index this table. If NetWarden finds an entry in the table, it uses the packet's header fields and timestamp to update the monitored values in the register arrays. Upon a miss, NetWarden needs to send this packet to the control plane for entry installation.

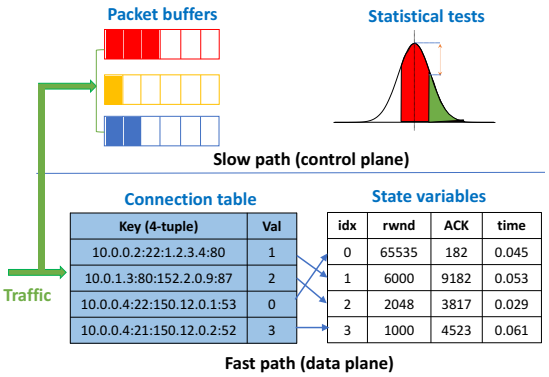


Figure 2: NetWarden consists of a fastpath defense on the data plane, and a slowpath defense on the control plane.

IPD characterization. If NetWarden keeps all IPDs for all connections, it would soon run out of memory and need to evict data to the control plane. Monitoring systems such as Marple [31] use this design, because they passively observe the traffic and only perform analysis offline. However, as an online defense, NetWarden needs to run at linespeed, so it should avoid such heavyweight evictions as much as possible. To address this, a possible approach is to keep k counters for a fixed numbers of IPD intervals $[0,t_2)$, $[t_2,t_3)$, $\cdots$, $[t_k,\infty)$. When a packet comes in, NetWarden computes the IPD between its timestamp and the last-seen packet timestamp from the same flow, and increments a counter for that IPD interval. Such a data structure *approximates* the IPD distribution on the data plane, and trades off some accuracy for efficiency.

Storage channel defenses. NetWarden also adapts a set of existing defense techniques for programmable data planes. Type-I channel defenses are the easiest, as they are simply header modifications (e.g., setting fixed values to ToS, Urgent Pointer, and IPID fields). Type-II channels require stateful defenses. For instance, consider the TCP initial sequence number channel [9]. NetWarden can replace a sequence number s with s', but then it also needs to apply the same offset $s'-s$ to subsequent packets in order to preserve correctness. The needed state is recorded in the connection table, which supports linespeed reads and writes. The main benefit NetWarden provides for these defenses is efficient hardware support.

Type-III channels exploit the non-determinism of network traffic, such as the partial ACK channel [28]. To the best of our knowledge, NetWarden is the first system that can defend against this channel, using a technique that we call *ACK rewinding*. If NetWarden sees an ACK packet that acknowledges an offset that is lower than the highest sequence number seen in this connection, it rewinds the ACK value to a smaller value. Such a value could be a previous ACK number before the last burst of packets were received, or it could be a previous ACK number offsetted by a controlled amount; either way would mitigate the partial ACK channel. This comes at the cost of unnecessary retransmissions, but such inefficiency would be addressed later by performance boosters.

3.2 The slowpath defense

The slowpath defense runs on the switch control plane. In addition to installing match/action entries upon a table miss, it has three more modules for channel defense: a) statistical IPD tests, b) timing channel defense, and c) performance boosters.

Statistical IPD tests. This module queries the IPD intervals for selected connections, and performs statistical tests for timing channel detection. As a defense platform, NetWarden can easily support existing detectors (e.g., Kolmogorov-Smirnov test [32] or regularity test [8]), or new detectors that may be developed in the future. The main novelty here is to offload IPD characterization to the data plane, which is heavyweight and needs to be performed per packet, and to perform statistical tests in software on aggregated IPD data. As ongoing work, we are also exploring whether certain simple tests (e.g., detecting skewed distributions) can be done on the data plane.

Timing channel defense. This module injects random delays to packets in suspicious connections to disrupt potential timing modulation. NetWarden temporarily buffers such packets in the control plane, and sends them to the destination in any desired order and at a time of its own choice. This obviously may lead to packet reordering and increased delay, but our performance boosters would counteract this impact.

Performance boosters. The performance boosters in NetWarden can temporarily increase the performance of certain TCP connections. However, this may cause data packets to be transmitted at a time when the receiver is not yet ready to process them, so the control plane needs to be involved for buffering. We discuss them in detail in the next subsection.

3.3 Performance boosting

TCP performance depends on a) the amount of available data to send, b) the network condition, and c) the receiver's ability to process new data. NetWarden cannot control a), but it can create the illusion that b) and c) are better than they actually are, and increase the TCP performance as a result.

ACK boosting. This technique *prefetches* data from the sender by generating ACK packets from NetWarden. When NetWarden sees an incoming packet from the sender, it forwards the packet to the receiver, and at the same time, immediately generates an ACK on behalf of the receiver. This hides the latency for a) the data packet to propagate to the receiver, b) the receiver to process the data and generate the real ACK, and c) for the ACK to propagate back to the sender. In effect, it creates the illusion of a shorter RTT as perceived by the sender, thus ramping up the sending rate faster. Although NetWarden forwards all data packets to the receiver, it still needs to buffer the prefetched data in the control plane temporarily, in case it needs to serve the data to the receiver again if packets are dropped from NetWarden to the receiver. The buffered data can be gradually removed when the actual ACKs from the receiver arrive at NetWarden. Such ACKs do

not need to be forwarded to the sender, since from the sender's perspective, the corresponding data packets have already been successfully received. This technique works more effectively when NetWarden is deployed close to the sender (data source), which corresponds to typical scenarios where NetWarden is co-located with a rack of servers that host sensitive data.

Receive window boosting. This technique generates an ACK packet on behalf of the receiver with a large receive window, creating the illusion of a high-performance receiver. This aims to remove any limit posed by the actual receive buffer size when the sender is computing the sending window (i.e., the minimum between the congestion window and the receive window). As before, the prefetched data is buffered in the control plane, and NetWarden serves the data as a proxy.

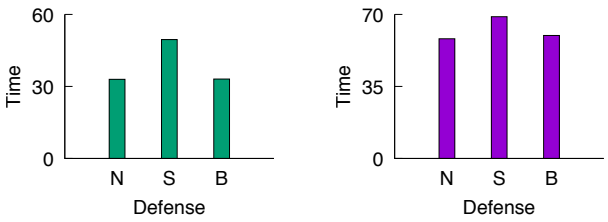
4 Initial Validation

We have built a preliminary NetWarden prototype for initial validation. Our prototype is written in P4 (for the fastpath) and Python (for the slowpath), and runs in a software-based simulator [3]. It implements a) a simple defense against a storage channel that encodes data in the receive window size field, by setting the window size to a smaller (therefore safe) value, b) a defense against a timing channel that uses large/small IPDs by injecting random delays, and c) the receive window boosting and ACK boosting techniques. In the timing channel setup, the server is sending a file to the remote client; in the storage channel setup, the client is uploading a file to the server; in both cases, the server attempts to exfiltrate data via covert channels. NetWarden is directly connected to the server for defense.

Baseline systems. Our main comparison is between a) the simple defenses (e.g., shrinking receive window size, adding random delays), and b) the performance-boosting defenses, both in terms of defense effectiveness and performance impacts. We have additionally used a scenario without any defenses (i.e., native performance, but vulnerable to attacks) as the baseline. Since our initial prototype runs in software simulation, we note that the high-level trends are more meaningful than the actual (software-based) measurements.

Defense effectiveness. We found that both defenses are effective against the tested channels. When there is no defense deployed, using the storage channel, the attacker can successfully embed data in the receive window; using the timing channel, it can achieve a decoding error rate as low as 4%. When either defense is applied, the storage channel based on the receive window size is no longer usable; for the timing channel, the simple defense increases the decoding error rate from 4% to 50%, and the performance-boosting defense increases the error rate to 49.3%, both rendering the decoding close to a random guess.

Performance impacts. For the storage channel attack, the data transfer took 49.52s to finish under the simple defense.



(a) Storage channel (b) Timing channel

Figure 3: NetWarden mitigates network covert channels in a performance-preserving manner. N: no defense; S: simple defense; B: performance-boosting defense.

This is in comparison to the "no defense" baseline, where the transfer took 32.98s. Therefore, the simple defense inflates the transfer time by 50.1%, a significant performance degradation. Using the performance-boosting defense, however, NetWarden achieved a transfer time of 33.09s, which only represents a 0.3% increase. For the timing channel, "no-defense" achieved a 58.15s transfer time, the simple defense inflated this to 68.91s (a 18.5% increase), and the performance-boosting defense only took 59.76s (a 2.8% increase).

Summary. The preliminary results above suggest that NetWarden seems to be a promising defense against network covert channels with minimum performance loss.

5 Related Work

We have already discussed related work on timing and storage channels, so we focus on other related projects here.

Normalizers. Normalizers can eliminate ambiguities in protocol payloads, preventing attacks due to inconsistent TTL values [17] or TCP retransmissions [35–37]. The key approach is to normalize traffic payload to a deterministic byte stream. However, even deterministic payload streams can contain covert channels; NetWarden focuses on the latter.

Active wardens. Existing research has developed active wardens for covert channel defense [9, 12, 25], but most wardens are only proof-of-concept systems that are hard to deploy due to their inefficiency. In addition, none of the existing wardens has considered performance preservation as a key design goal.

Programmable data planes. Programmable data planes have been used for a wide variety of tasks [10,13,20,21,31,39]. NetWarden is most related to Dapper [13] on TCP performance diagnosis. However, the primary goal of NetWarden is to use programmable data planes for security.

6 Ongoing Work

NetWarden is ongoing work, and there are quite a few open issues to be explored as next steps. First, we are developing

principled approaches to identifying the boundary between the fastpath and the slowpath for an "optimal" division of labor. Second, estimating how much performance boosting NetWarden should perform for a connection requires a careful design. If we boost the performance of a connection too much, this would consume the bandwidth of contending flows. We would like to design defenses that not only preserve performance for a single connection, but also ensure fairness across TCP flows. Moreover, since many TCP variants exist, we would like to understand how NetWarden should perform the boosting differently, or whether we could design a universal strategy for preserving performance. Last but not least, our prototype and experimental results are in a very preliminary stage. We plan to continue our investigation on NetWarden by developing a full hardware-based prototype and evaluating it with realistic traffic speeds and deployment scenarios.

7 Acknowledgments

We thank Qiao Kang, Kuo-Feng Hsu, and the anonymous reviewers for their valuable feedback. This work was partially supported by an NSF grant CNS-1801884.

References

[1] Amazon EC2 instance types. `https://aws.amazon.com/ec2/instance-types/`.

[2] The P4 language repositories. `https://github.com/p4lang`.

[3] Tofino: World's fastest P4-programmable Ethernet switch ASICs. `https://www.barefootnetworks.com/products/brief-tofino/`.

[4] C. Abad. IP checksum covert channels and selected hash collision. *USA, University of California*, 2001.

[5] A. Belozubova, A. Epishkina, and K. Kogos. Random delays to limit timing covert channel. In *Proc. European Intelligence and Security Informatics Conference (EISIC)*, 2016.

[6] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN. *ACM SIGCOMM Computer Communication Review*, 43(4):99–110, 2013.

[7] S. Cabuk. *Network covert channels: Design, analysis, detection, and elimination*. PhD thesis, Purdue University, 2006.

[8] S. Cabuk, C. E. Brodley, and C. Shields. IP covert timing channels: Design and detection. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2004.

[9] D. M. Dakhane and P. R. Deshmukh. Active warden for TCP sequence number base covert channel. In *Proceedings of the International Conference on Pervasive Computing (ICPC)*, 2015.

[10] H. T. Dang, D. Sciascia, M. Canini, F. Pedone, and R. Soulé. NetPaxos: Consensus at network speed. In *Proceedings of the Symposium on SDN Research (SOSR)*, page 5. ACM, 2015.

[11] Department of Defense. Trusted computer system evaluation criteria (TCSEC). (DoD 5200.28-STD), 1985.

[12] G. Fisk, M. Fisk, C. Papadopoulos, and J. Neil. Eliminating steganography in Internet traffic with active wardens. In *Proceedings of the International Workshop on Information Hiding (IH)*, 2002.

[13] M. Ghasemi, T. Benson, and J. Rexford. Dapper: Data plane performance diagnosis of TCP. In *Proceedings of the Symposium on SDN Research (SOSR)*, pages 61–74. ACM, 2017.

[14] S. Gianvecchio and H. Wang. Detecting covert timing channels: An entropy-based approach. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2007.

[15] S. Gianvecchio, H. Wang, D. Wijesekera, and S. Jajodia. Model-based covert timing channels: Automated modeling and evasion. In *Proceedings of the International Workshop on Recent Advances in Intrusion Detection (RAID)*, pages 211–230. Springer, 2008.

[16] J. Giffin, R. Greenstadt, P. Litwack, and R. Tibbetts. Covert messaging through TCP timestamps. In *Proceedings of the International Workshop on Privacy Enhancing Technologies (PETS)*, pages 194–208. Springer, 2002.

[17] M. Handley, C. Kreibich, and V. Paxson. Network intrusion detection: Evasion, traffic normalization and end-to-end protocol semantics. In *Proceedings of the USENIX Security Symposium*, 2001.

[18] A. Hintz. Covert channels in TCP and IP headers. *Presentation at DEFCON*, 10:16, 2002.

[19] S. Jero, E. Hoque, D. Choffnes, A. Mislove, and C. Nita-Rotaru. Automated attack discovery in TCP congestion control using a model-guided approach. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2018.

[20] X. Jin, X. Li, H. Zhang, N. Foster, J. Lee, R. Soulé, C. Kim, and I. Stoica. NetChain: Scale-free sub-RTT coordination. In *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 35–49, 2018.

[21] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica. NetCache: Balancing key-value stores with fast in-network caching. In *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP)*, pages 121–136. ACM, 2017.

[22] E. Jones, O. Le Moigne, and J.-M. Robert. IP traceback solutions based on time to live covert channel. In *Proceedings of 12th IEEE International Conference on Networks (ICON)*, volume 2, pages 451–457. IEEE, 2004.

[23] B. Lampson. A note on the confinement problem. *Communications of the ACM*, 16, 1973.

[24] K. S. Lee, H. Wang, and H. Weatherspoon. PHY covert channels: Can you see the idles? In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 173–185, 2014.

[25] G. Lewandowski, N. B. Lucena, and S. J. Chapin. Analyzing network-aware active wardens in IPv6. In *Proceedings of the International Workshop on Information Hiding (IH)*, pages 58–77. Springer, 2006.

[26] M. Liu, L. Luo, J. Nelson, L. Ceze, A. Krishnamurthy, and K. Atreya. IncBricks: Toward in-network computation with an in-network cache. *ACM SIGOPS Operating Systems Review*, 51(2):795–809, 2017.

[27] X. Luo, E. W. W. Chan, and R. K. C. Chang. TCP covert timing channels: Design and detection. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2008.

[28] X. Luo, E. W. W. Chan, and R. K. C. Chang. CLACK: A network covert channel based on partial acknowledgment encoding. In *Proceedings of the IEEE International Conference on Communications (ICC)*, 2009.

[29] X. Luo, E. W. W. Chan, R. K. C. Chang, and W. Lee. A combinatorial approach to network covert communications with applications in web leaks. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2011.

[30] G. R. Malan, D. Watson, F. Jahanian, and P. Howell. Transport and application protocol scrubbing. In *Proceedings of the IEEE International Conference on Computer Communications (INFOCOM)*, 2000.

[31] S. Narayana, A. Sivaraman, V. Nathan, P. Goyal, V. Arun, M. Alizadeh, V. Jeyakumar, and C. Kim. Language-directed hardware design for network performance monitoring. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 85–98. ACM, 2017.

[32] P. Peng, P. Ning, and D. S. Reeves. On the secrecy of timing-based active watermarking trace-back techniques. In *Proceedings of IEEE Symposium of Security and Privacy*, 2006.

[33] C. H. Rowland. Covert channels in the TCP/IP protocol suite. *First Monday*, 2(5), 1997.

[34] G. Shah, A. Molina, M. Blaze, et al. Keyboards and covert channels. In *Proceedings of the USENIX Security Symposium*, 2006.

[35] U. Shankar and V. Paxson. Active mapping: Resisting NIDS evasion without altering traffic. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2003.

[36] G. Varghese, J. A. Fingerhut, and F. Bonomi. Detecting evasion attacks at high speeds without reassembly. In *ACM SIGCOMM Computer Communication Review*, volume 36, pages 327–338. ACM, 2006.

[37] M. Vutukuru, H. Balakrishnan, and V. Paxson. Efficient and robust TCP stream normalization. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2008.

[38] X. Wang and D. S. Reeves. Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 20–29. ACM, 2003.

[39] N. Yaseen, J. Sonchack, and V. Liu. Synchronized network snapshots. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pages 402–416. ACM, 2018.