Stealth DoS Attacks on Secure Channels

Amir Herzberg[*] and Haya Shulman[†]

Bar Ilan University
Department of Computer Science
Ramat Gan, 52900, Israel

Abstract

We initiate study of the use of 'secure tunnel' protocols, specifically IPsec, and its availability and performance guarantees to higher-layer protocols, in particular TCP, against Denial/Degradation of Service (DoS) attacks. IPsec is designed to provide privacy and authentication against MITM attackers, and employs an anti-replay mechanism to ensure performance. For our analysis, we define a new family of adversaries, the stealth denial and degradation of service (DoS) adversaries. These adversaries are weaker than the classical MITM adversary, and may be of interest in other works. We analyse their ability to launch (DoS) attacks on secure channels, and show realistic amplification attacks, disrupting TCP communication over secure VPNs using IPsec. In particular, we show that anti-replay mechanism is critical for performance by launching a DoS attack on communication over IPsec without anti-replay window. We present attacks exploiting insufficient IPsec anti-replay window size, and show how to calculate correct window size. Finally we present attacks on IPsec with correctly adjusted anti-replay window size thus showing that even large anti-replay window does not ensure performance to TCP flows. We then suggest a fix to TCP in IPsec gateway designed to prevent the above attacks, and to provide secure channel immune to degradation and other DoS attacks. Our solution involves changes (only) to the sending gateway machines running IPsec. In addition to their practical importance, our results also raise the challenge of formally defining secure channels immune to DoS and degradation attacks, and providing provably-secure implementations.

[*]Amir.Herzberg@gmail.com
[†]Haya.Shulman@gmail.com

1. Introduction

Denial/Degradation of service (DoS) attacks pose an ever growing threat to Internet services and applications. Secure channel protocols, with IPsec [27, 38] being the predominant one, are used to securely connect virtual private networks (VPN), i.e., authenticate data and origin, ensure confidentiality, and performance. IPsec is designed to protect against man-in-the-middle (MITM) adversaries that can eavesdrop on the communication and inject spoofed segments into the message stream. It is widely believed, and also specified e.g., in [27], that IPsec also defends higher-layer traffic from DoS attacks when attacker has limited resources (e.g., can only block, inject or reorder a limited number of packets). Defense against DoS attacks is often an important consideration in adopting IPsec for protecting a VPN (rather than say using SSL/TLS [22, 14]). We show that this belief is not precise and that IPsec does not deliver on its performance guarantees, by presenting several DoS attacks on TCP when used over IPsec.

TCP [35] is the transport layer communication protocol that underlies most Internet applications, e.g., web, mail, file transfer, remote access. TCP provides a reliable and connection oriented service to its users, allows fair sharing of network resources with mechanisms for flow and congestion control. However, TCP does not provide security guarantees against network adversaries.

Our attacks raise the following question: what are the properties that secure channel should satisfy to protect against performance degradation attacks? Existing works do not analyse the properties that secure channel protocols should possess to protect against denial of service attacks. There are works that attempt to define what secure channel is, e.g., [12], but they fail to capture performance analysis of secure channel, i.e., efficiency and resistance to denial of service attacks. Herzberg and Yoffe [21] present a framework that allows to define specifications that capture such properties, and suggest further research on defining secure channel protocols within that framework. However, they do not present such specifications for DoS-preventing secure channel protocols, or demonstrate that existing secure channel protocols fail to protect against DoS. Our work provides such demonstration; we hope that it will prompt research leading to such specifications and provably-secure DoS-preventing channels. Specifically, we show that although IPsec employs an anti-replay mechanism that is targeted at ensuring performance by detecting and discarding spoofed duplicate packets injected by a MITM adversary, it fails to counter denial/degradation of service (DoS) attacks. We show DoS attacks that exploit congestion control mechanism of TCP.

In each section we present different techniques for exploiting the vulnerabilities of TCP congestion control mechanism, which rely on slightly different adversarial model. The attacks we present rely on standard behaviour of correctly implemented TCP congestion control mechanism. We then analyse the impact that these attacks can have on TCP performance (when run over IPsec). In addition, we demonstrate the necessity for and motivate the anti-replay mechanism of IPsec, by presenting simple attacks on TCP congestion control mechanism when IPsec is used without the anti-replay window. We also investigate the correct *size* of IPsec's anti-replay window, and show attacks when incorrect window size is used. We also show how to compute correct anti-replay window size. Yet, we show degradation of service attacks by stealth adversary (defined in Section 2.2), even when sufficient anti-replay window size is used.

In Section 5.3 we discuss solutions to combat the reordering attacks (whether by malicious adversary, or due to benign network congestion), and present a fix in IPsec gateway, to address the reordering of packets. Our goal is not to require changes in the TCP protocol in every host separately, but to apply the modification to the firewall, and as a result to protect subnet of hosts. Many private networks connected to the Internet are protected by firewalls. Firewall protection is based on the idea that all packets destined to hosts behind a firewall have to be examined by the firewall. When applied to firewall, our mechanism requires minimal changes to existing implementations, to combat the attacks presented in the rest of this paper. Our solution is comprised of two phases: first detection and then prevention of an attack, and is based on delaying congestion notification, i.e., duplicate ACKs, and discarding if turned out to be false.

Our stealth attacks can be applied to other tunneling protocols, e.g., to the widely used tunneling Generic Routing Encapsulation (GRE) mechanism, see [15]. According to [15], GRE specifies a protocol for encapsulation of an arbitrary protocol over another arbitrary network layer protocol, and is a common way to achieve tunneling of IP encapsulated inside IP. GRE does not provide authentication, i.e., it is vulnerable to spoofing adversary; to perform denial of service against GRE, an attacker can simply send a segment with a higher sequence number. To prevent this type of attacks, it is suggested to run GRE over IPsec, however, as we show in this work, IPsec does not protect against this type of attacks.

In all our attacks we assume a stealth attacker model, presented in Section 2.2, that can with minimal effort significantly degrade the performance of communication over TCP. Our attacker may be restricted in its eavesdropping capability (may be able to eavesdrop on one network segment but not the other), as well as in the number of (spoofed) packets that it can inject. For instance, in wireless network attacker can only eavesdrop on wireless communication, and may be able to inject segments in the wired access network. Often attackers may be limited in their spoofing ability, e.g., attacker is able to disrupt communication by infiltrating a small device which has a limited power. In addition, attackers typically prefer to avoid detection, thus spoofing a limited number of segments. Note that our attacks exploit the congestion control of TCP, by injecting duplicate segments. This strategy allows attacker to evade DoS detection mechanisms, e.g., consider a sequence of routers on the path between source and destination, where the attacker controls one of the routers. The router simply duplicates some of the segments that traverse it, and reroute them via an alternative path. Thus the malicious router cannot be traced back. On the other hand, if the router simply dropped occasional segments, this could be detected, and the attack would be traced back to the malicious router. For more details on attacks on wireless networks by MITM adversary (and limitations) can be found in [33]. Similar attacker model was considered in [36], which investigated an Explicit Congestion Notification (ECN) with IPsec. We discuss this briefly in Related Works in Section 1.2.

1.1. Other DoS Attacks on IPsec

In this work we consider DoS attacks by stealth attackers, that can eavesdrop and spoof packets, yet even weaker, blind spoofing, attacker can mount a DoS on IPsec. For instance, it is known that fragmentation can expose IPsec to DoS attacks, e.g., IPsec cannot prevent attacks on fragments' buffer at the recipient if fragmentation is allowed. Specifically, since authentication is performed prior to fragmentation, spoofing attacker could launch a DoS attack by swamping the receiving gateway with (maliciously crafted) IP fragments, which could not be reassembled, thus legitimate packets could not be accepted, e.g., in [25]. This attack is made possible due to the fact that IPsec reassembles the fragments prior to authenticating them, and the attack can be prevented by defining minimal fragment size and not allowing fragmentation; another solution is to only allow prefragmentation, i.e., fragmentation by IPsec gateway prior to applying IPsec processing on the outgoing packet.

Another attack is returning an ICMP port unreachable error message which would force the sender to reduce fragments' size until no (or minimal size, e.g., Byte) packets can be exchanged, discussed in [16]. This attack is prevented trivially against a spoofing attacker; IPsec packets include the security parameters index (SPI), used to identify the security association (SA) used for the connection, in packets' headers. In addition, an ICMP port unreachable error message includes the 8 bytes of the original packet (which has the SPI). The SPI field is secret and random, therefore cannot be known to spoofing attacker, and if invalid SPI is received it is ignored by the receiving IPsec gateway. Yet this solution does not hold against our stealth attacker since it can observe the SPI value in packets' headers, and can thus forge a correct and valid SPI. However, this attack (as opposed to ours) can also be prevented, e.g., by defining minimal fragment size, and once reaching that size gateway would ignore further ICMP port unreachable messages. Another solution, against a stealth attacker (that cannot drop packets) can be to check if ACKs arrive, in response to transmitted messages, e.g., like in our our stealth attacks, then ignore the ICMP port unreachable messages.

DoS attacks can also be launched on IKE (key establishment protocol of IPsec), which was designed to run over UDP in order to avoid DoS attacks on TCP. In [25], the authors show an attack on IKE, by exploiting fragmentation.

A vulnerability of IPsec to DoS when using Explicit Congestion Notification (ECN) is investigated in [36]. If the IPsec gateway at the exit of the tunnel does not copy the ECN bit, then it ruins the ECN mechanism; on the other hand, if the gateway copies the ECN bit, then an attacker can degrade performance. The attack can be launched since the authentication that IPsec performs does not protect the ECN bit. However, there is noanalysis of this attack; such analysis is rather similar to the analysis we present, of similar attacks. In addition, our attacks work even if ECN bit is not used, as well as if the recommendation of the RFC not to copy the ECN bit from tunneled packets is followed. Note, that the authors of [36] consider similar adversarial model to ours, i.e., they consider a 'weaker MITM' attacker model like the one we present and define in Section 2.2, although we also consider duplications, and do not consider modifications to legitimate packets, e.g., turning on/off ECN bit.

1.2. Related Works

1.2.1. Denial/Degradation-of-Service (DoS) Attacks

Denial/Degradation of Service (DoS) attacks, and especially Distributed DoS (DDoS) attacks, pose a serious threat to Internet applications. In the last years, DoS attack methods and tools are becoming more sophisticated, effective, and also more difficult to trace to the real attackers. We briefly recap several types of DoS attacks, using different (roughly, diminishing) adversarial capabilities.

The basic distributed denial of service attack is brute force or flooding, see e.g. [23, 13], and SYN attack in [37]. Flooding DoS attacks typically utilise a large number of compromised nodes in order to consume network resources by flooding an Internet link, and thus shutting off TCP flows. The shortcomings of this attacks from the attacker perspective is that they are easy to detect due to high volume of uniform traffic, e.g., network administrators can identify performance degradation in infected machines and eliminate the vulnerabilities that allowed the attack. Alternately, an ISP can block the malicious traffic. In addition, attacker may also be blocked by rate controls or limited by bandwidth of a zombie. However, recently it has been shown that attackers can achieve similar outcomes without overloading the system in a persistent manner, using attacks such as described next (and such as the attacks investigated in this paper). TCP targeted (low-rate) *Shrew* attacks, [29] exploit the retransmission timeout (RTO) of TCP, by transmitting short traffic pulses of RTT scale length, of low average volume of RTO scale periods, causing TCP flows to continually timeout. The result is near zero TCP throughput. Due to the nature of the attack traffic it can be hard to distinguish it from other legitimate traffic, e.g., video. Low-rate TCP attacks are much harder to detect, and require much weaker attacker capabilities, i.e., the attacker can simply generate bursty UDP flows of low average rate.

Low-rate TCP targeted *Reduction of Quality (RoQ)* attacks are another type of low-rate TCP attack, introduced in [18, 19, 32], where attacker exploits the TCP AIMD mechanism causing TCP performance degradation. The main difference is that RoQ attacks do not require precise timing (to tune to the RTO frequency). The RoQ attacks are even more difficult to detect and block, since they do not operate at specific intervals. In [32] authors suggest a type of attacks similar to RoQ attacks, i.e., the pulsing attacks, which are targeted at TCP applications. The pulsing attacks can be categorised into two models: *timeout-based* attacks, and *AIMD-based* attacks, depending on the timing of the attack pulses w.r.t. congestion window of TCP. During the attack, pulses of malicious traffic are sent to a victim, resulting in packet losses. Authors of [32] show that even a small number of attack pulses can cause significant throughput degradation. Recently, in [1], a new denial of service attacks, dubbed *JellyFish*, were exhibited. JellyFish attacks target TCP congestion control mechanism of TCP flows, by having the relay nodes misorder, delay or drop packets which they are expected to forward.

Low rate TCP targeted attacks can be prevented by using secure channel protocol between the gateways, e.g., LOT in [16], and using mechanisms that provide quality of service by differentiating traffic, e.g., DiffServ [7]. Namely, when employing DiffServ, flows are given different priority, and flows over a secure channel can be given higher priority, and will be reserved space in routers buffers. Alternately, non-conforming packets can be dropped or given a lower priority and placed in different queues.

1.2.2. Making TCP Robust to Reordering

Some of our attacks, e.g., Section 5, are based on (malicious) reordering of network packets, and we propose a fix to TCP in IPsec gateways. Our solution may also be integrated in TCP in each host, to handle benign network reordering (yet further research is required to present experimental work and analysis).

A wide range of TCP modifications has been proposed to improve robustness to reordering, e.g., [8, 41, 10, 6, 40, 11]; see a survey in [30] and an analysis in [8]. Existing works focus on benign network reordering. In [4] authors describe a collection of techniques that provide one way reordering measurements in both directions between a client and most TCP based servers on the Internet, and propose a metric to summarise reordering activity. In [30] the authors consider the impact of packet reordering on TCP, and survey approaches to handle the issue. Authors identify two approaches: the ordinal approach and the temporal approach. Eifel and DSACK based algorithms are experimental RFCs.

1.2.3. Internet Protocol Security (IPsec)

Internet Protocol Security (IPsec), in [27], provides network layer security against MITM attackers, offering privacy and/or integrity to the exchanged communication, and authenticates source of IP packets, i.e., prevents spoofing of IP addresses. IPsec can be used in two modes: transport or tunnel, and has two security protocols ESP, providing encryption and optional authentication, and AH, providing authentication. IPsec employs an anti-replay window to ensure performance by preventing duplicates, i.e., replays of the communication exchanged by the legitimate parties, by discarding duplicate segments at the receiver. In a replay attack an adversary sends a copy of previously transmitted, legitimate message between a sender and a receiver, see [39] for more details. When the replayed packet reaches the destination, it will be passed to the transport layer buffer. Duplicate messages degrade performance and is an obvious motivation for anti-replay window. Maintaining and managing an anti-replay window can require significant memory resources,; optimisation of anti-replay mechanism, e.g., [24, 17, 43] try to come with more efficient implementations. Yet some solutions, e.g., [17], that attempt to save resources by decreasing window size are susceptible to attacks, which may result in more damage than not using an anti-replay window at all. According to [27, 24, 17, 43], the anti-replay mechanism of IPsec is used to secure IP against an adversary that can insert possibly replayed messages in the message stream, and as a result prevent denial of service attacks. In particular, the authors of [43] present *robustness* to DoS attacks as one of the requirements of anti-replay mechanism, and claim that the possibility that packets will be dropped is traded with the prevention of replay attack. We show that even large enough anti-replay window cannot prevent DoS attacks, and in Section 5 we present a new type of *low-rate* TCP attacks, the *reordering attacks*, which significantly degrade performance even when sufficiently large IPsec window is used.

1.3. Contributions

We show that IPsec alone cannot provide protection against DoS. The contributions of this work can be summarised as follows:

- We identify an important attack model, the stealth attack, which was not explicitly defined prior to this work.
- We justify and analyse IPsec's anti-replay mechanism, Section 3, and show how to compute optimal IPsec anti-replay window to prevent packets loss due to reordering attacks in Section 4.
- We present *degradation of service* attacks on TCP when running over IPsec, which work even when a large IPsec anti-replay window is employed, Section 5. We analyse our results with a simple analytical model of TCP performance degradation.
- We propose a fix to TCP in IPsec gateways, Section 5.3, to prevent the stealth reordering attacks.
- Conceptual contribution: we initiate investigation of the performance properties that secure channel protocols should provide.

2. Model

In this section we present the scenario which we consider in the paper, we model and motivate the attacker, and give assumptions on the communication.

2.1. Scenario and Attack Model

Consider the scenario presented in Figure 1, with a virtual private network between two branches, both connected via gateways $GW1, GW2$ to the Internet. All the communication between the branches is over IPsec, using IPsec's ESP mode with authentication. For simplicity, we assume

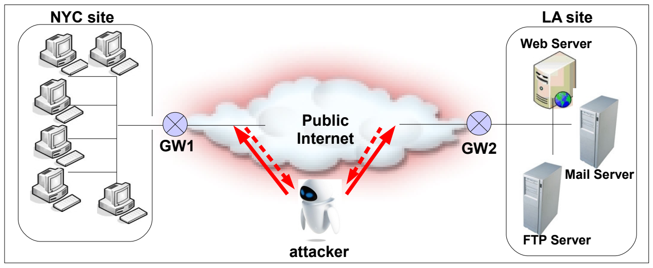

Figure 1: Virtual private network behind gateway $GW1$ with users accessing servers located behind $GW2$. IPsec is used for protection, and a stealth adversary is located on the Internet, and attacking a Virtual Private Network (VPN) between two sites.

that the clients are located behind $GW1$, and the servers are located behind $GW2$. The clients send requests to download files from servers, and servers send the requested files in response. We assume that all communication is over TCP, and upon each correctly received data segment, a client generates and sends an acknowledgment (ACK). An attacker located on the Internet between the two gateways, $GW1$ and $GW2$, in Figure 1, is able to eavesdrop on the communication and inject (a limited number of) packets into the message stream, but cannot drop legitimate packets. More details on the attacker are presented in the next subsection.

2.2. Stealth Adversary Model

In this work, we define and consider the stealth adversary model that can eavesdrop on communication, and spoof packets (based on packets it observed), but cannot delay or drop packets; the 'classical' man-in-the-middle (MITM) adversary can eavesdrop, intercept communication, drop and inject spoofed packets into the message stream. Attacker that drops packets, i.e., MITM, can disrupt communication and mount a denial of service attack, e.g., by blocking all communication, yet we are interested in sophisticated *amplification* attacks where attacker spends considerably less resources w.r.t. the resulting damage. In addition, in reality attackers often do not have MITM capabilities; and even when attackers can drop packets they often prefer to refrain when an alternative exists, in order to avoid detection.

Like in low rate attacks [29], we restrict the attacker's ability to send (inject) spoofed and/or duplicated packets. Specifically, we believe a realistic model would be to define a quantified, (ρ, σ)-limited stealth adversary following the 'leaky bucket' approach. Namely, an (ρ, σ)-limited stealth adversary is one who can send, during any interval of length T, at most $\rho \cdot T + \sigma$ spoofed and/or duplicated packets. These limitations are weaker compared to those of low-rate attacks, e.g., [29, 19, 32], since the attacker is not just limited in the amortised traffic, but also cannot create bursts of traffic. In particular, the bursts are limited by σ, i.e., an (ρ, σ)-limited stealth attacker can create a $\sigma-$burst which is a burst of σ segments. We show that even this weaker attacker can dramatically degrade performance, even when communication is protected by IPsec.

We consider stealth (i.e., weak MITM) attackers, and packets they inject can depend on the packets they eavesdrop. In fact, since the communication between the two gateways is authenticated (using IPsec), it follows that the adversary can effectively only duplicate packets, and possibly 'speed up' delivery of a duplicate so it will arrive before the regularly-sent packet, e.g., via an alternative path. Note that the attacker may be limited in the direction in which it can inject spoofed segments, e.g., can only duplicate segments sent from NYC site to LA site in Figure 1, but cannot duplicate segments in the other direction.

In each attack we present we use a slightly different variant of the (ρ, σ)-limited stealth attacker; the different variants of the attacker model are illustrated in Figures 2, and 3 and defined below; The weakest stealth adversary (Figure 2) can duplicate packets. A stealth attacker in Figure 3, can also reorder packets[1] by speeding them up, e.g., via a faster route, in addition to its ability to duplicate packets. We now model the attackers based on the definitions above:

σ-Duplicating stealth attacker (Figure 2): this is the weakest adversary model we consider. As the name implies, the duplicating attacker can duplicate packets, so they are received σ number of times instead of once. Specifically, let i be a sequence number of some packet. Packet i was duplicated if σ identical copies of packet i arrived (in addition to the original packet i) with sequence duplicate packets.

We use the duplicating attacker to motivate the use of anti-replay window mechanism in IPsec; for this attack, we only need to send $\sigma = 3$ duplicates of a few packets.

s-Reordering attacker (Figure 3): our next attacker can duplicate σ packets and cause the duplicate(s) to be delivered via a faster route to the destination, i.e. faster than the delay of other packets (including the original duplicated packet). Let $1, 2, 3, ...$ be a sequence of transmitted packets (bounded by the maximal number of packets in transit, see Claim 4). An s-reordering occurs if packet with sequence number i arrives before packet with sequence number $i - s$. We later show how such an attacker can disrupt communication over IPsec implementations which use an insufficiently-large anti-replay window. We believe that such reordering capability may often be available to attackers and is therefore a reasonable model, e.g., an at-

[1]Further research should be conducted to consider the damage that attackers without speed-up capabilities, i.e., with the same delay as the legitimate communicating parties, can inflict.

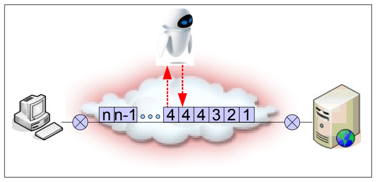

Figure 2: Duplicating stealth adversary.

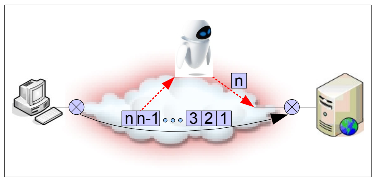

Figure 3: Reordering stealth adversary.

tacker may receive services from a better ISP that provides a faster communication channel than the channel used by the communicating parties, thus attacker can reorder packets by sending duplicates over a faster route; or attacker may control zombie computers that will send more traffic on the route between the two gateways, causing significant queuing delays there, while the attacker speeds-up the duplicate packet via a different path to the destination gateway. We assume that attacker has some non-zero delay, which is smaller than that of the legitimate parties. Specifically, adversary is said to be an s-reordering stealth attacker, if it can cause delivery of the duplicate packet with sequence number i, before packet with sequence number $i - s$ (before the delivery of the original packet i). The reordering parameter is a function of attacker's delay and the delay of the legitimate parties. The justification of our adversarial model, is that we focus on the use of IPsec, and IPsec is necessary only when there is concern about MITM. In particular, the anti-replay mechanism that IPsec employs is used to prevent injection of duplicate segments, by identifying and discarding replayed packets. This type of attack can be performed by attacker that can eavesdrop and inject spoofed packets, i.e., a MITM attacker.

As we mentioned before, we are not interested in trivial 'flooding' attacks where the attacker achieves degradation by spending resources proportional to the performance degradation achieved, e.g., attacker injected two packets thus the link carries additional load, and IPsec has to inspect two more packets, resulting in some degradation performance but also attacker's 'cost' is proportional. We are focus on *amplification* attacks where the attacker pays minimal resources with respect to the inflicted damage, e.g., injects three packets, but with a devastating result on the attacked flows.

2.3. Communication Model

We assume that packets arrival is organised in FIFO (first in first out) and that they are delivered with fixed latency which is known to the attacker. The delay of packets is between $delay_{MIN}$ and $delay_{MAX}$ (if a packet does not arrive within $delay_{MAX}$ seconds it is assumed to have been loast) which the attacker can choose. Delivery of attacker's packets may not be in FIFO but their but the delay is at least $delay_{MIN}$. In addition, we assume that the attacker (similarly to other network entities) is subject to some non-zero network delay, which may be smaller than that of the legitimate parties, and which the attacker cannot change. Throughout the paper we denote by RTT (Round Trip Time) the time it takes to transmit a segment from a client into the network and to receive an acknowledgment (ACK) for it in response. Namely, RTT is the sum of segment's transmission time, propagation delay, transmission of ACK and its propagation delay back to the sender, including any queuing and processing delays involved. The attacks we present apply to standard TCP implementations [35]; TCP state machine is in Figure 5 (from [28]). We assume that the connection is always open, and that the sender sends full sized segments as fast as its congestion window allows. For simplicity (only), assume that the recipient acknowledges every segment received from the sender, i.e., no delayed ACKs[2]. For ease of exposition, we work with segments instead of bytes (which is what TCP actually sends). We also assume that flow control does not restrict congestion window growth. Let $cwnd(t)$ be the congestion window size at time t. We analyse TCP throughput in terms of transmission rounds, each round starting with the sender transmitting the first segment in a window of size $cwnd(t)$ at time t. Each round ends when the sender receives an ACK for one of the segments in a window. In this model, the duration of a round is the round trip time (RTT), and is independent of the window size. Notice that at any time t holds that the number of 'pending' packets in transit at time t is smaller (or equal) to congestion window size at time t (unless the sender is in 'fast recovery' phase at time t).

3. Motivating Anti-Replay Window

In this section we consider a general question of anti-replay mechanism. More specifically, should a secure channel protocol that aims to protect against denial/degradation-of-service (DoS) attacks, employ an anti-replay mechanism. IPsec employs an anti-replay mechanism although replay sensitive protocols over IP are typically robust to replay, e.g., TCP. Our answer to this question is positive: we claim

[2]When receiver sends an ACK for every other segment, i.e., uses delayed ACK, the congestion window grows in less than one segment per RTT; this does not significantly change our results.

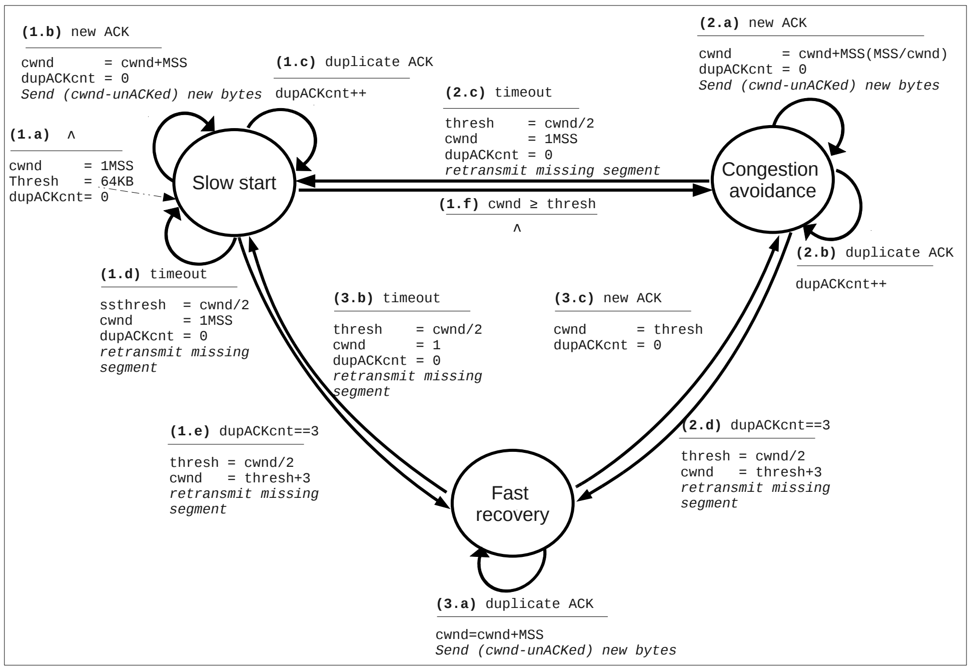

Figure 5: TCP congestion control state machine of the sender (based on [28]).

that anti-replay mechanism is essential to counter DoS attacks, and we show that protocols that provide only confidentiality and authentication are vulnerable to DoS attacks. In particular, we focus on IPsec, which is often used to provide solutions at the channel layer. IPsec standard, [26], requires anti-replay mechanism to identify and discard replayed packets in order to prevent DoS attacks, e.g., [43], claim that the reason for anti-replay mechanism is to save CPU cycles which will be wasted on replayed packets, as well as to prevent incorrect billing information. Yet to obtain access to a service or resource, attacker will have to obtain secret keys used to encrypt (and possibly authenticate) the communication, and it will not gain much by merely replaying already sent messages. In addition, typically, replay-sensitive applications check for freshness of messages and discard (or ignore) replayed messages. We present an additional motivation for IPsec anti-replay window; more specifically, we show that without the anti-replay window, (amplification) degradation of service attacks on congestion control of TCP can be launched with significant performance damages. In what follows we describe the attacks that could be launched if no anti-replay mechanism were used. These attacks require merely a duplicating stealth attacker (see Figure 2).

3.1. ACK Duplication Attack: Stealth DoS on Channel without Anti-Replay Mechanism

Client behind $GW1$ requests to download a file from server behind $GW2$, as in Figure 1. The attack is presented in Figure 6. The main idea of the attack is to duplicate a legitimate ACK sent by the client in response to some segment, and retransmit three duplicate copies of that ACK. TCP considers the receipt of three duplicate ACKs as an indication of lost segment, which in turn can be a sign of congestion (see Figure 5). As a result, TCP at the sender halves its congestion-control window. Furthermore, if prior to the attack TCP connection were in 'slow start' phase (where the congestion window grows exponentially), TCP also moves to the linearly-growing 'congestion avoidance' (CA) mode, thus prematurely aborting the slow-start phase. As a result, connection uses a small congestion window, which results in severe bandwidth underutilisation. By repeating this attack periodically, attacker can ensure that the connection continuously uses very small, suboptimal window. In Figure 6 we present an attack on a TCP connection in CA mode. For simplicity assume that the congestion window at the beginning of first attack epoch at time t_0 is $cwnd(t_0) > 4 * MSS$. Since this is the first attack, adversary did not inject any duplicate packets in the interval $(t_0 - T, t_0)$ (and hence can send three packets at any interval beginning from t_0). Assume that the server sends a window of $k + 1$ segments $i, ..., i + k$, and the client upon receipt, transmits $k + 1$ ACKs such that ACK on segment $i + k$ is

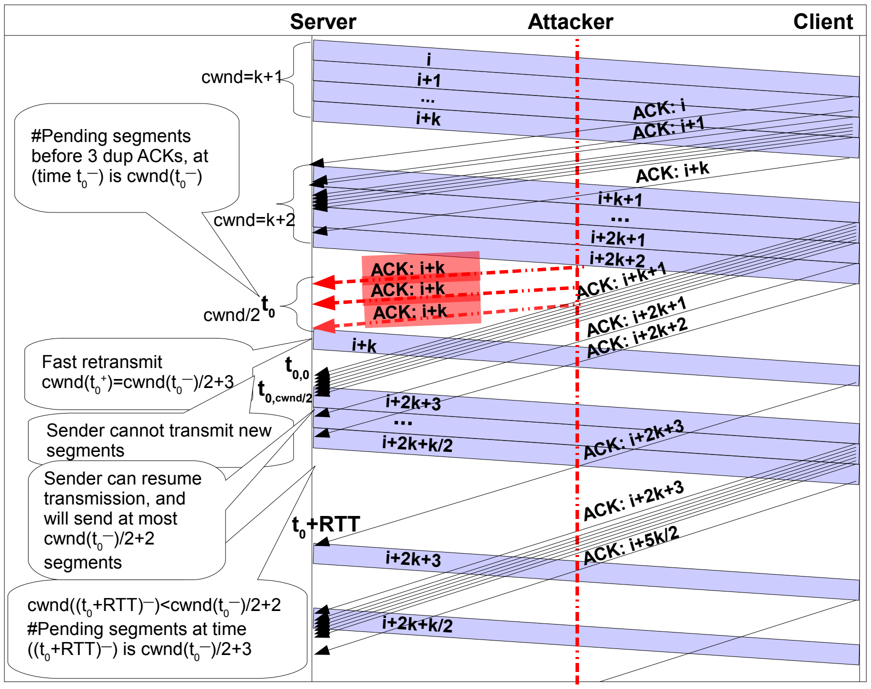
Server
Attacker
Client
cwnd=k+1
i
i+1
...
i+k
ACK: i
ACK: i+1
#Pending segments before 3 dup ACKs, at (time t0−) is cwnd(t0−)
ACK: i+k
i+k+1
...
i+2k+1
i+2k+2
cwnd=k+2
ACK: i+k
ACK: i+k
ACK: i+k
ACK: i+k+1
cwnd/2
t0
ACK: i+2k+1
ACK: i+2k+2
i+k
Fast retransmit cwnd(t0+)=cwnd(t0−)/2+3
t0,0
t0,cwnd/2
i+2k+3
...
i+2k+k/2
Sender cannot transmit new segments
ACK: i+2k+3
Sender can resume transmission, and will send at most cwnd(t0−)/2+2 segments
t0+RTT
ACK: i+2k+3
ACK: i+5k/2
i+2k+3
cwnd((t0+RTT)−)<cwnd(t0−)/2+2 #Pending segments at time ((t0+RTT)−) is cwnd(t0−)/2+3
i+2k+k/2

Figure 6: ACK duplication attack on TCP congestion control mechanism when no IPsec anti-replay window is employed (Section 3.1). The attacker duplicates a legitimate ACK segment (sent by the client to the server in response to receipt of a data segment) and sends three duplicate copies of that ACK to the server. The server takes the three duplicate ACKs as an indication of network congestion, retransmits 'missing segment' and reduces its sending rate.

last in the window. The attack begins when the attacker creates three duplicate copies of last ACK (for segment $i + k$) sent by the receiver in recently transmitted window of ACK segments. At time t_0 (in Figure 6) the server receives three duplicate ACK copies injected by the attacker. Since we ignore transmission delays, three duplicate ACKs arrive at the same time, with no other ACK segment arriving between the most recently transmitted legitimate ACK and the receipt of three duplicates of that ACK, that were injected by the attacker. This deceives the server into believing that the three duplicate ACKs are transmitted as a result of a lost segment in last transmitted window of segments. Receipt of three consecutive ACKs for segment $i + k$ is taken as an indication of congestion which resulted in loss of segment $i + k$. As a result, once the server receives three consecutive duplicate ACKs, (according to step (2.d) in Figure 5, if the TCP at the sender is in CA, or step (1.e) if the TCP is in slow-start) it performs fast retransmit of the segment which it believes to have been lost (step (2.d), Figure 5), i.e., transmits the 'lost' segment for which duplicate ACKs were generated, and sets the congestion window to $cwnd(t_0^+) = \frac{cwnd(t_0^-)}{2} + 3$ and $thresh = \frac{cwnd(t_0)}{2}$. The server then enters fast recovery mode (Figure 5) until receipt of an ACK for new data (i.e., on segment $i + k + 1$, in Figure 6). Since triple duplicate ACKs were not generated due to congestion, ACK segments for all pending segments at time t_0 eventually arrive at the server. Once ACK acknowledging new data arrives, the congestion window is deflated (step (3.c), Figure 5), i.e., set to half of its value before the receipt of three duplicate ACKs, i.e., $cwnd = thresh$, and the server enters congestion avoidance phase during which the sending rate grows linearly (step (2.a), Figure 5).

3.2. ACK Duplication Attack: Analysis

We consider a $(\rho, 3)$-limited duplicating stealth adversary (see Figure 2), with constant delays; notice we require $\sigma = 3$ since the attack requires the adversary to duplicate three[3] ACK segments. Attacker can repeat the attack once every $T = 3/\rho$ seconds; we refer to T as the length of each *attack epoch* (or the frequency of attack epochs), i.e., the time elapsed between two consecutive duplications of three ACK segments by the attacker. We analyse the operation of TCP in the T seconds from one duplication to the next.

In Claim 1, we show that when connection is under packet duplication attack, average steady state congestion window $cwnd_{MAX}^{ATK}$ is bounded by $\frac{6}{\rho \cdot RTT}$ (average congestion window growth between attack epochs is in Figure 7).

[3]Injecting more than 3 duplicate segments will not have the maximal amplification effect.

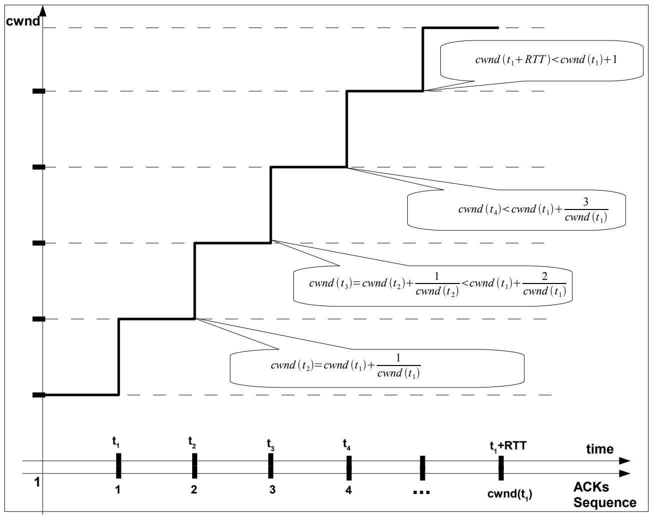
cwnd
$cwnd(t_1+RTT)<cwnd(t_1)+1$
$cwnd(t_4)<cwnd(t_1)+\frac{3}{cwnd(t_1)}$
$cwnd(t_3)=cwnd(t_2)+\frac{1}{cwnd(t_2)}<cwnd(t_1)+\frac{2}{cwnd(t_1)}$
$cwnd(t_2)=cwnd(t_1)+\frac{1}{cwnd(t_1)}$
t_1
t_2
t_3
t_4
t_1+RTT
time
1
1
2
3
4
...
cwnd(t_1)
ACKs Sequence

Figure 4: TCP congestion window growth in congestion avoidance phase upon receipt of each new ACK. Let t_1 be the time after a window of packets was sent and before first ACK was received, and let $cwnd(t_1)$ be congestion window size at time t_1. Then at each time t_i, for $i \in \{2, ..., RTT\}$, when each ACK arrives, the congestion window set to $cwnd(t_i) = cwnd(t_{i-1}) + \frac{1}{cwnd(t_{i-1})}$. After a sequence of $cwnd(t_1)$ ACKs, congestion window $cwnd(t_1) \leq cwnd(t_1) + 1$.

This window size results in data transfer rate of at most $\frac{cwnd^{ATK}_{MAX}}{RTT} = \frac{6}{\rho \cdot RTT^2}$ (as we show in Claim 2), which can be very small - in fact, negligible compared to the expected throughput without attack, which is the average TCP congestion window[4] divided by the round trip time. The number of attack epochs i to reach steady state congestion window is $i = cwnd(t_0^-) - \frac{2T}{RTT})$, which we derive in Claim 3.

Claim 1 *Steady state congestion window $cwnd^{ATK}_{MAX}$ of TCP sender when under packet duplication attack is $cwnd^{ATK}_{MAX} \leq \frac{6}{\rho \cdot RTT}$*

Proof Let $t_{i,j}$ be the j^{th} ACK segment received by server, at time t_i after attack epoch i (described in Figure 6), i.e., after the receipt of three duplicate ACKs at time t_i. At time $t_{0,1}$, ($t_{0,1} > t_0$) first legitimate ACK segment sent by the receiver after first attack epoch, arrives. Since we assume constant delays, the first ACK arriving at time $t_{0,1}$ is essentially a legitimate ACK generated by the receiver with a higher sequence number than the ACK duplicated by the attacker. As a result the server will exit fast recovery, will set the congestion window to $cwnd(t_{0,1}) = \frac{cwnd(t_0^-)}{2}$ thus deflating the congestion window, and will enter a CA phase during which the congestion window increases linearly, approximately by one segment during each RTT; analysis of linear congestion window growth is presented in Figure 4, and holds: $cwnd(t + RTT) < cwnd(t) + 1$. At time $t_{0,1}$ (when first legitimate ACK arrives) the server cannot transmit new segments into the network since $cwnd(t_{0,1}) < cwnd(t_0^-)$, where $cwnd(t_0^-)$ is the number of pending segments and $cwnd(t_{0,1}) = \frac{cwnd(t_0^-)}{2}$, i.e., the congestion window does not allow transmission of new segments. Let n be the number of pending segments at time t_0, then n ACKs should arrive at the sender at times $t_{0,1}, ..., t_n$ at constant intervals (since we assume constant delays). After arrival of $\frac{n}{2} = \frac{cwnd(t_0^-)}{2}$ ACK segments, $t_{0,1}, ..., t_{0,\frac{n}{2}}$, the server can resume transmitting segments with each new ACK arrival, and upon receipt of $cwnd(t_0^-)$ ACK segments, a total of $\frac{cwnd(t_0^-)}{2}$ segments will be transmitted. After RTT seconds, at time $t_0 + RTT$ the ACK on the retransmitted segment at time $cwnd(t_0^+)$ arrives, and the congestion window size is

$$cwnd((t_0 + RTT)^+) = \frac{cwnd(t_0^-)}{2} + \frac{cwnd(t_0^-)}{\frac{cwnd(t_0^-)}{2}} = $$
$$= \frac{cwnd(t_0^-)}{2} + 2$$

Next attack epoch is initiated $T = 3/\rho$ seconds later, i.e., it takes $T = 3/\rho$ seconds until attacker can launch the attack again, by sending three subsequent duplicate ACK segments. Since connection at this time is in congestion avoidance (CA) phase, i.e., congestion window grows by one segment in every round-trip time (RTT), the congestion window at second attack epoch, at time $t_1 = t_0 + \frac{T}{RTT} RTT$

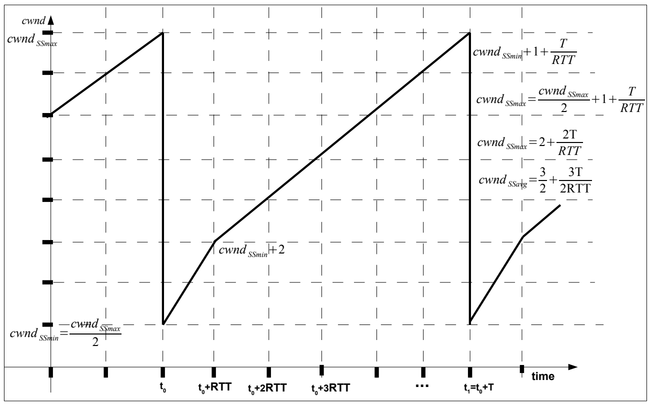
cwnd
$cwnd_{SSmax}$
$cwnd_{SSmin}+1+\frac{T}{RTT}$
$cwnd_{SSmax}=\frac{cwnd_{SSmax}}{2}+1+\frac{T}{RTT}$
$cwnd_{SSmax}=2+\frac{2T}{RTT}$
$cwnd_{SSavg}=\frac{3}{2}+\frac{3T}{2RTT}$
$cwnd_{SSmin}+2$
$cwnd_{SSmin}=\frac{cwnd_{SSmax}}{2}$
t_0
t_0+RTT
t_0+2RTT
t_0+3RTT
...
t_1=t_0+T
time

Figure 7: Average congestion window size $cwnd$ at steady state, if TCP connection is under packet duplication attack when no anti-replay window is employed (see Figure 6), resulting in data transfer rate of at most $\frac{cwnd^{ATK}_{MAX}}{RTT} = \frac{6}{\rho}$. Note that the growth of window size is discrete, i.e., upon each packet arrival. When $T = RTT$, $cwnd = 3$ and throughput is at most $\frac{3}{RTT}$.

[4]Average TCP window depends on the operating system and configuration. Typical bound on window size, e.g., on windows 7, is 256KB.

will have reached $cwnd(t_1) = \frac{cwnd(t_0^-)}{2} + \frac{T}{RTT}$, where $\frac{T}{RTT}$ is the number of RTTs between each attack epoch. Let $t_i = t_0 + i\frac{T}{RTT}RTT$ for $i \in \mathbb{N}$, be the time at i^{th} attack epoch. In Equation 1, we derive congestion window size $cwnd$ at time t_i as a function of the attack frequency and of the congestion window size $cwnd(t_0)$, prior to first attack epoch.

$$\begin{aligned} cwnd(t_i) &\leq \frac{cwnd(t_{i-1})}{2} + \frac{T}{RTT} = \\ &= \frac{cwnd(t_0^-)}{2^i} + \frac{(2^i-1)}{2^{i-1}}\frac{T}{RTT} = \\ &= \frac{2T}{RTT} + \frac{1}{2^i}\Big(cwnd(t_0^-) - \frac{2T}{RTT}\Big) \quad (1) \end{aligned}$$

Since $\frac{1}{2^i}$ is negligible, the whole expression approximates $\frac{2T}{RTT}$, i.e., $cwnd(t_i) \leq \frac{2T}{RTT}$. For $T = 3/\rho$, $cwnd(t_i) \leq \frac{6}{\rho \cdot RTT}$. In addition, since for every t_i, $cwnd_{MAX}^{ATK} \leq cwnd(t_i)$, the bound on the steady state congestion window size when under attack is given by: $cwnd_{MAX}^{ATK} \leq \frac{6}{\rho \cdot RTT}$.$\square$

Claim 2 *Throughput of average steady state congestion window is at most $\frac{6}{\rho \cdot RTT^2}$.*

Proof To compute throughput we take the steady state congestion window long term data size derived in Claim 1, and divide it by the RTT:

$$\frac{\frac{cwnd(t_0^-)}{2^i} + \frac{(2^i-1)T}{2^{i-1}RTT}}{RTT} \leq \frac{2T}{RTT^2} = \frac{6}{\rho \cdot RTT^2}$$

$\square$

Claim 3 *Let i be the number of attack epochs required to reach steady state congestion window $cwnd(t_i)$ at time t_i. Then $i < log_2(cwnd(t_0^-) - \frac{2T}{RTT} - 2) - 1$.*

Proof Assume at time t_i, steady state is reached, and according to Lemma 1, $cwnd(t_i) - 1 < cwnd(t_{i+1})$; then $cwnd(t_i) < \frac{2T}{RTT} + 1$. In Claim 1 we derived the expression for steady state congestion window to be $cwnd(t_i) = \frac{2T}{RTT} + \frac{1}{2^i}\Big(cwnd(t_0^-) - \frac{2T}{RTT}\Big)$; by substituting $cwnd(t_i)$ with $\frac{2T}{RTT} + 1$ we obtain $\frac{2T}{RTT} + \frac{1}{2^i}\Big(cwnd(t_0^-) - \frac{2T}{RTT}\Big) < \frac{2T}{RTT} + 1$; by solving the equation we obtain that upper bound on the number of attack epochs to reach steady state is $i < log_2(cwnd(t_0^-) - \frac{2T}{RTT})$. $\square$

Lemma 1 *Steady state congestion window $cwnd(t_i)$ is reached when $cwnd(t_i) < cwnd(t_{i+1}) + 1$, during i^{th} attack epoch at time t_i.*

Proof We consider the following two cases related to congestion window growth between each subsequent attack epoch:

$\frac{cwnd(t_i)}{2} > \frac{T}{RTT}$ The window size decreases between each subsequent attack epoch; this is due to the fact that the duration between attack epochs does not suffice for congestion window to restore its size.

$\frac{cwnd(t_i)}{2} \leq \frac{T}{RTT}$ The congestion window size is restored between each attack epoch; this occurs when steady state is reached.

Conside the case where $cwnd(t_i) > \frac{2T}{RTT}$; congestion window size decreases between each two subsequent attack epochs t_i and t_{i+1}: at each attack epoch holds: $cwnd(t_{i+1}) + 1 \leq cwnd(t_i)$. Once $cwnd(t_i) \leq \frac{2T}{RTT}$, congestion window between attack epochs t_i and t_{i+1} is: $cwnd(t_i) < cwnd(t_{i+1}) + 1$, and the steady state window size is reached; then $cwnd(t_{i+1}) \leq cwnd(t_i) < cwnd(t_{i-1})$, where $cwnd(t_{i-1})$ is congestion window right before steady state, and $cwnd(t_i) < cwnd(t_{i+1}) + 1$. $\square$

3.3. Data Packets Duplication Attack: Stealth DoS on Channel without Anti-Replay Mechanism

Packets duplication attack is symmetric to ACKs duplication attack presented in Section 3.1. In order to achieve performance reduction, attacker tricks the receiver into generating duplicate ACKs by duplicating packets. Attacker injects three duplicate copies of a previously sent packet. As defined in [20], TCP receiver acknowledges every packet received, thus three duplicate packets trigger three duplicate ACKs at the receiver. Upon receipt of three consecutive duplicate ACKs sender performs fast retransmit and slows down its sending rate.

Attack on data packets duplication, albeit similar in nature to attack on ACKs, serves several purposes: it may be the case that the attacker can only duplicate and inject packets in one direction, in which packets flow and not in the direction in which ACKs flow, i.e., can inject segments sent from server to client but cannot inject segments from client to server, e.g., if the attacker is located in one network and can only duplicate and inject packets in that same network, or if ingress filtering mechanism is employed. Another purpose is related to assumptions on the power of the attacker: duplicating packets attack can be carried out by a much weaker (slower), attacker, than the attacker that mounts duplicating ACKs attack, since it does not require attacker to inject the duplicates in limited time frame. In packets duplication attack attacker can inject segments that are much older than the legitimate segments. This attack will not succeed with duplicating ACK segments, since the sender will ignore outdated duplicate ACKs, i.e., if a new ACK has already arrived the sender ignores the old ACK that arrives.

4. Determining Anti-Replay Window Size

Attacks presented in Section 3 motivate the necessity for anti-replay mechanism. Anti-replay mechanism would discard duplicate packets, thus preventing amplification DoS attack. IPsec standard [27], recommends using anti-replay mechanism as a protection against denial of service (DoS) attacks by a MITM adversary and to prevent replay of packets, however current specifications do not provide recommendation on how to calculate proper window size. If anti-replay window is incorrectly adjusted, i.e., too small, re-ordered packets can result in packets' loss, due to discarded legitimate packets by IPsec implementation.

Packet reordering occurs when packets are received in a different order from the one in which they were sent. Packet reordering can significantly degrade TCP performance. Upon duplicate ACKs the sender triggers *fast retransmit* and *fast recovery*. As a result the congestion window remains small relatively to the available bandwidth. As specified in [2], out of order data segments should be acknowledged immediately in order to accelerate loss recovery. To trigger the fast retransmit algorithm the receiver should send an immediate duplicate ACK when it receives a data segment above a gap in the sequence space. If messages are reordered on transit, e.g., due to benign network reordering or by malicious attacker, there will be a gap in sequence numbers of the arrived packets, which may result in anti-replay mechanism discarding valid messages if the anti-replay window is incorrectly adjusted, i.e., too small.

Existing works propose more efficient implementations of anti-replay window. In [17] the authors analyse correctness of anti-replay window in malicious setting, where they assume a MITM adversary that is located on the Internet and can inject duplicate segments. Authors conclude that anti-replay window is designed to prevent such attacks. Subsequent works, [24, 43], show that severe reordering of messages, possibly maliciously by a MITM (e.g., controlling a router), can result in discarded legitimate packets by IPsec implementation (due to reordering) and suggest alternative mechanisms that should reduce the number of discarded packets, w.r.t. to IPsec anti-replay mechanisms.

In this section we show that if anti-replay window is not correctly adjusted, a more severe performance degradation could be induced to TCP flows, than not using anti-replay window at all. We then show how to calculate correct window size in Section 4.3, given the relevant network parameters.

4.1. Packets' Reordering Attack: Stealth DoS on Channel with Small Anti-Replay Window

An adversary can cause an IPsec implementation to discard valid packets by injecting replayed packets with higher sequence number into the message stream thus advancing the anti-replay window, and as a result legitimate packets with low sequence numbers, i.e., to the left of the anti-replay window, will be discarded by the IPsec. Discarded packets result in three duplicate ACKs at the sender, which then reduces the TCP congestion window. The throughput of TCP connections under attack is significantly degraded. The damage is a function of the frequency at which the adversary can launch the attack. The attacker model we consider in this section is presented in Figure 3.

Since we analyse worst case scenario, we assume throughout the attack a single TCP flow which the attacker attempts to attack, with no other communication (this is equivalent to using a distinct SA per connection). The course of the attack is presented in Figure 8. Assume that IPsec anti-replay window consists of $W = n$ packets and TCP window is comprised of $cwnd = k+1$ segments, such that $n < k + 1$; attack can be launched when IPsec anti-replay window is smaller than TCP congestion window[5]. We assume that the attacker knows when IPsec window is smaller than TCP window, since IPsec anti-replay window size is part of the design (thus is known to the attacker), and due to the fact that attacker is eavesdropping on the communication it can observe the TCP window size at any given time.

In Figure 8 sender receives k ACKs and transmits a window of $k+1$ segments, with i^{th} segment being the first segment in the TCP window and $i + k$ being the last segment, i.e., with highest sequence number. Attacker reorders, i.e., duplicates and injects (speeding it up), a segment[6] with sequence number $i + n$ (segment $n + 1$ in the TCP window), which is the first segment to the right of the anti-replay window, such that it arrives before the first segment i in the window. Thus upon receipt of the segment, IPsec implementation at the receiving gateway $GW1$ advances the anti-replay window with segment $i + n$ being the right edge of the window, and passes this segment to the receiver. When TCP at the client receives segment with sequence number $i+n$ it generates a duplicate ACK with sequence number i, i.e., sequence number of the expected segment, indicating a gap in sequence numbers of the received segments. The rest k segments arrive intact, and are passed to the client. When the original segment with sequence number $k + 1$ arrives, IPsec detects a replay and discards it. For each subsequently received segment, with sequence numbers in the range between $i + 1$ and $i + n - 1$, the TCP at the client

[5]This is a reasonable assumption. In particular, provided no network congestion, TCP window is limited only by threshold, which is typically set to 65KB, according to RFC 2988 [34].

[6]In this attack we assumed for simplicity that the attacker speeds-up a single segment, since this (simpler) case is bad enough to motivate correct window size. If attacker speeds up l segments, then the 'recovery' is much slower, since each of the l segments will have to be retransmitted, and thus recovery will require additional l transmission rounds.

generates a duplicate ACK, indicating that it is missing a segment with sequence number i. A total of k duplicate ACKs are returned to the sender. Packet loss is taken as an indication of congestion and that the TCP window has grown larger than the network can handle, hence TCP at the sender takes corrective steps by decreasing its sending rate, i.e., decreases the number of segments in a window. Denote by t_0 the time at which the sender receives three duplicate ACKs. According to the specification in RFC 2581 [3], upon receipt of 3 duplicate ACKs, fast retransmit algorithm at the sender retransmits lost segment, i.e., segment with sequence number i; sender halves the congestion window adding three duplicate ACKs, yielding congestion window of size: $cwnd(t_0) = \frac{cwnd(t_0^-)}{2} + 3$. Then the sender enters a fast-recovery phase until a non-duplicate ACK arrives. At this stage the congestion window does not allow transmission of new segments into the network, since the number of pending segments is larger than the congestion window size, i.e., $cwnd(t_0^-) + 1 > \frac{cwnd(t_0^-)}{2} + 3$. For each subsequent duplicate ACK the sender increments the congestion window by 1 MSS. The addition of duplicate ACKs to the congestion window artificially inflates the window by the number of segments that have left the network and arrived at the receiver. After receipt of $\frac{cwnd(t_0^-)}{2} + 1$ duplicate ACKs (since t_0) the sender can resume transmission of new segments; and $\frac{1}{2}cwnd(t_0^-) - 2$ remaining duplicate ACKs will arrive. For each duplicate ACK the sender transmits a new segment into the network. In this transmission round a total of $\frac{1}{2}cwnd(t_0^-)$ segments will be sent.

At time $t_0 + RTT$ the sender should receive an ACK for the retransmitted segment. Once the sender receives an ACK for new data[7] congestion window is deflated, i.e., set to $\frac{cwnd(t_0^-)}{2}$, and the sender enters congestion avoidance (CA) phase, during which the congestion window is incremented linearly, i.e., roughly by one segment in every RTT.

4.2. Packet Reordering Attack: Analysis

The throughput of the connection is kept low, since the adversary can resume the attack (if the attack frequency parameter allows it) every time the congestion window is larger than the anti-replay window. The ratio between TCP window $cwnd$ and IPsec anti-replay window W before the first attack epoch, as well as the frequency at which the attacks can be launched, dictates the performance degradation inflicted by the attack, and the impact can range between degradation of service and complete denial of service. If TCP congestion window is larger than IPSec anti-replay by 1 segment, then attack achieves a result similar to reduction of quality (RoQ) attacks, in [18, 19]. In this case, it will take $\frac{k+3}{2}$ RTTs to restore the congestion window from $\frac{k-1}{2}$ back to its original value, before the first attack, i.e., $k + 1$, since in every RTT the congestion window grows by one segment. But attacker cannot keep the congestion window at steady state (like in Section 3.2), since next attack can be launched when TCP window grows larger than IPsec anti-replay window. Congestion window growth between each attack epoch (which is launched when TCP window is larger than IPsec window is presented in Figure 9.

Alternately, if $cwnd \geq 2 * W + 4$ attacker can disrupt the connection by causing the retransmission timeout (RTO) to expire, thus performance degradation induced by the attack is similar in its result to the low rate attacks presented in [29]. In order to cause connection to timeout, attacker will 'speed-up' (reorder) segment which will result IPsec anti-replay window to move forward a window number of segments, thus discarding segment(s) to the left of the window. When sender re-transmits this segment in next transmission round, attacker reorders segments again, such that the retransmitted segment is again discarded. At this time the sender is in fast recovery, and will only change state when it receives an ACK for a re-transmitted segment. However the sender keeps receiving duplicate ACKs, therefore connection will eventually timeout, and move to slow start phase, and retransmit the segment again. But in response it will receive all the duplicate ACKs that were previously transmitted by receiver. If the IPsec and TCP windows ratio is sufficiently large, attacker can cause timeout again, which will reset connection. The minimal $cwnd$ and W ratio which would result in timeout of the retransmitted (due to receipt of three duplicate ACKs) segment i, should be computed as follows: let $cwnd$ be the amount of transmitted segments in the window prior to first attack epoch. Denote by P the number of segments in transit (for which the sender has not yet received acknowledgments), and denote by W the anti-replay window size. In order for the attack to result in a timeout the following inequality has to hold: $\lfloor \frac{cwnd}{2} \rfloor + (cwnd - 1) - P > W$. Since the number of pending segments P is equal to the number of transmitted segments, i.e., $P = cwnd$, holds: $\lfloor \frac{cwnd}{2} \rfloor - 1 > W$.

4.3. Adjusting IPsec Anti-Replay Window

In order to prevent denial/degradation of service (DoS) attacks we presented, a larger anti-replay window should be used, and the question is how much larger. The largest pos-

[7] We stress, that for a larger ratio of TCP congestion window and IPsec anti-replay window, a more devastating attack is possible. More specifically, the attacker will again speed-up the retransmitted segment i, which will again be discarded, and thus the sender will continue receiving duplicate ACKs till it encounters a timeout event. After a timeout the sender again retransmits the 'lost segment i' and enters a slow start. However, then it receives duplicate ACKs for i^{th} segment from previous transmission round, and enters congestion avoidance. If the $cwnd$ vs. W (IPsec anti-replay) is sufficiently large, and enough duplicate ACKs return, the connection will eventually be reset.

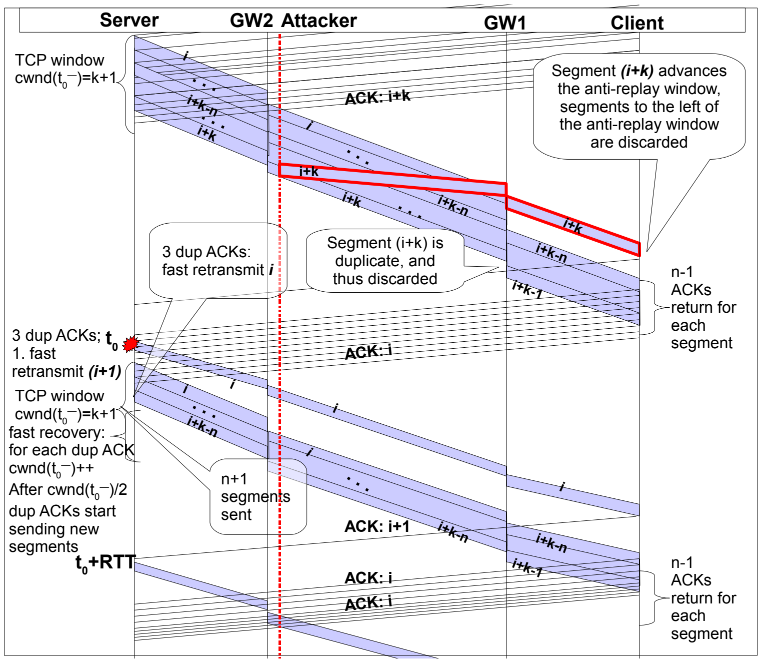
Server
GW2 Attacker
GW1
Client
TCP window cwnd(t0−)=k+1
ACK: i+k
Segment (i+k) advances the anti-replay window, segments to the left of the anti-replay window are discarded
3 dup ACKs: fast retransmit i
Segment (i+k) is duplicate, and thus discarded
n-1 ACKs return for each segment
3 dup ACKs; t0 1. fast retransmit (i+1)
ACK: i
TCP window cwnd(t0−)=k+1
fast recovery: for each dup ACK cwnd(t0−)++
After cwnd(t0−)/2 dup ACKs start sending new segments
n+1 segments sent
ACK: i+1
t0+RTT
ACK: i
ACK: i
n-1 ACKs return for each segment

Figure 8: Packets' reordering attack (Section 4.1) on TCP exploiting an insufficient size of IPsec anti-replay window (single attack epoch). In this attack we assume first attack epoch is launched when congestion window is of size $W + 1$ (where W is IPsec anti-replay window).

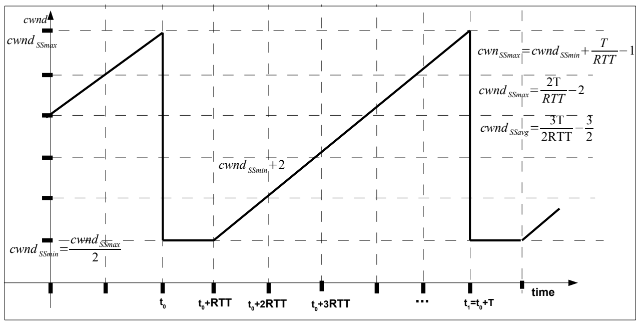

Figure 9: TCP congestion window $cwnd$ analysis, when connection is under packets' reordering attack, as in Figure 8, and TCP congestion window is by one segment larger than IPsec anti-replay window.

sible IPsec anti-replay window is one that can contain all the packets within a specific SA, i.e., window containing 2^{32} packets. Such anti-replay window of maximal size prevents the attacks presented in Section 4.1, which were made possible due to insufficient anti-replay window size. Namely, even severe (whether malicious or benign) reordering will not result in dropped packets when anti-replay window size is increased to maximal size. However, a naive implementation of anti-replay window containing 2^{32} packets requires 2^{32} bits (an average of 536 Mega-Bytes) is inefficient and is rendered impractical due to high memory cost and maintenance.

We differentiate between the size of the data structure N required to store and maintain an anti-replay window of size W, i.e., number of packets that anti-replay window of size N can reflect (or represent).

There are works that attempt to achieve a more efficient anti-replay window implementation requiring less storage size, e.g., [24, 43, 17]. However, there are no works that analyse anti-replay window size W in an adversarial setting, where attacker can maliciously adjust its strategy. More specifically, there are two issues that should be addressed w.r.t. anti-replay mechanism: the number of packets that anti-replay window should reflect, in order to prevent packets' loss due to reordering, and the data structure to store and manage this information efficiently. In this work we focus on window size (and not the size of its representation), and compute an upper bound on the number of packets W that the anti-replay window should reflect, based on the rates of the given network, in order to avoid discarding segments by IPsec implementation (when small anti-replay window is used) due to reordering, based on the rates of the given network.

Claim 4 *Let R be transmission rate and L_{MIN} be minimal packet size. Let $delay_{MIN}$ be the minimal delay and let $delay_{MAX}$ be the maximal delay. Attacker can set delays to all packets to any value in the interval $[delay_{MIN}, delay_{MAX}]$ as long as legitimate packets arrive in FIFO (first in first out). Then IPsec anti-replay will not discard reordered legitimate packets (due to packets'* *reordering attack) that are not duplicates of a previously received packets, if IPsec anti-replay window size* W *is:*

$$W \geq \frac{R \times (delay_{MAX} - delay_{MIN})}{L_{MIN}}$$

Proof Assume attacker's delay $delay_{MIN}$ is 0, then anti-replay window is $W \geq \frac{R \times delay_{MAX}}{L_{MIN}}$, i.e., at least the size of maximal number of packets in transit. In this case even maximal reordering of packets, i.e., last packet arriving before first in a window of transmitted packets, will not result in discarded packets by IPsec, since anti-replay is at least the size of packets in transit. Alternately, if attacker's delay $delay_{MIN} > 0$, then anti-replay window can be less than maximal number of packets in transit, since attacker's packets are also experiencing delay, and as a result it will be limited in its ability to reorder packets. Thus the upper bound on IPsec anti-replay window size is a function of maximal packets in transit and delay of the attacker. □

Since attacker's delay is typically not known, in order to compute upper bound on anti-replay window size, we assume worst case, i.e., attacker with zero delay. Note that this is a rather conservative computation, since typically, the attacker's speed will also be a function of the delay. Therefore, IPsec's anti-replay window W should be at least the size of the maximal number of packets in transit at any given time, to prevent the attacker from discarding out of order packets, by advancing IPsec anti-replay window.

For network with propagation delay $delay_{MAX}$ of one second, transmission rate R of 10 Mega-Bytes per second, and maximal packet size $L = 1000$ Byte, the maximal number of packets in transit is $10,000$.

Existing works [43, 17, 24], investigate constructions of optimal anti-replay mechanism, to reduce the resources required to maintain anti-replay window. A naive implementation would be to set $N = W$, i.e., the representation of the data structure of anti-replay window is the same as the number of packets that the window represents. While this lower bound on anti-replay window will prevent (both benign and malicious) reordering which advance the anti-replay window and result in discarded segments, it can be too large and inefficient for practical purposes, i.e., maintaining such a large window can be a challenge, w.r.t. processing requirements and storage resources (especially if a distinct SA pair is established per each TCP flow). Thus the goal is to maintain a window of size $N < W$ where W is the number of packets in the anti-replay window. Typically, anti-replay window is a sparse vector with occasional out of order packets, and allocating large buffers may be alleviated with a more efficient data structure. Naive solutions that attempt to save resources by decreasing window size are susceptible to attacks, and may result in more damage than when implementing IPsec with no anti-replay mechanism.

4.4. Single vs. Multiple Flow Security Association

The IPsec standard [27] specifies establishing a unidirectional security association (SA) between the communicating parties, but it does not specify whether to establish a single SA in each direction or if to establish a distinct SA for each traffic flow. A single SA for all flows is more efficient since the gateway is required to keep less state. In addition, single SA prevents distinguishing between different communication flows. On the other hand it increases the risk for a chosen plaintext attack (CPA) [5], since the attacker can 'use' other flows to attack a target flow (thus for cryptographic purposes it is recommended to use a distinct SA per each flow). Furthermore, when a single SA is used for all flows our attack on small anti-replay window is more devastating, and the attack can be launched more frequently. An SA per each flow allows distinguishing between different flows, albeit when one flow is attacked, others are not effected. On the flip side, a distinct SA for each connection imposes a significant overhead, and requires larger memory resources. However, even when a single SA pair per connection is established reduction of performance attack is still possible, albeit not as effective as compared to a single SA pair. If a single security association (SA) pair is established for all communication, then the attacker can mount the attack more frequently.

5. IPsec with Large Anti-Replay Window

If IPsec anti-replay window is not properly adjusted, i.e., too small, reordering of packets (e.g., by malicious attacker) can degrade performance of TCP connections. However, as we show in this section, even sufficiently large IPsec anti-replay window, i.e., computed in Claim 4, does not prevent throughput degradation attacks. In this section we assume that there are available resources to maintain an anti-replay window of the required size to prevent attacks that advance the anti-replay window resulting in packets' loss. We consider a $(\rho, 3)$-limited stealth attacker, following the model in Figure 3, where attacker reorders segments, i.e., injects duplicate packets and speeds them up, e.g., via a faster route than the one available to the legitimate parties. This is a slightly stronger attacker than the one assumed in Sections 3.1 and 4.1. We show that the throughput degradation is identical to the one in Section 3, which shows that IPsec anti-replay mechanism does not prevent degradation-of-service attacks, even when the anti-replay mechanism is of sufficient size.

5.1. Packets' Reordering Attack: Stealth DoS on Channel with Large Anti-Replay Window

Consider scenario in Figure 1, in which client behind gateway $GW1$ requests to download a file from server behind gateway $GW2$, and assume IPsec implementation is using sufficiently large anti-replay window. We present a detailed attack on this connection in Figure 10. Server transmits a window of $k+1$ segments to the client, and assume $k \geq 3$. Attacker speeds up three segments[8] with higher sequence numbers, i.e., $(i+k-2), (i+k-1), (i+k)$. These segments arrive before i^{th} segment to the gateway. IPsec authenticates the segments, and verifies that they are not a replay of previously sent segments and passes them to the client. When client receives these out of order segments, it detects a gap in sequence numbers of received packets, i.e., the sequence number is higher than the one expected. As a result, client generates a duplicate acknowledgment for next segment it expects to receive, i.e., segment with sequence number i. Once the sender receives three duplicate ACKs for the same segment, it retransmits the 'lost' segment, halves the congestion window, and enters fast recovery (step (2.d) if TCP is in CA or step (1.e) if TCP is in slow start, see Figure 5). Upon arrival of an ACK acknowledging new data, i.e., ACK for segment $(i+1)$, (step (3.c) in Figure 5) congestion window is deflated, i.e., set to half of its value before the receipt of three duplicate ACKs, and the server enters congestion avoidance phase, i.e., increases the congestion window by roughly 1 segment in every RTT (step (2.a) in Figure 5). Note: no packets are discarded by IPsec implementation, since the anti-replay window is assumed to be sufficiently large.

5.2. Packets' Reordering Attack: Analysis

In Claim 5 we show that an upper bound on congestion window at steady state when under attack is $cwnd_{AVG}^{ATK} \leq \frac{6}{\rho \cdot RTT}$ (resulting in average congestion window size at steady state of $\frac{3T}{RTT}$). We then derive, in Claim 6, the long-term throughput of the connection when under packet reordering attack: $\frac{6}{\rho \cdot RTT^2}$ (we ignore the part of the connection when the window keeps decreasing since it is negligible w.r.t. the overall connection throughput). Note that this throughput is much lower than the average throughput of TCP connection that is not under attack, i.e., throughput is the average TCP congestion window (which can be up to $65KB$ divided by the round trip time (RTT)). Let T be the attack epoch frequency, and $T = \frac{3}{\rho}$, then for $T = RTT$ (T defined in Section 2.2), the resulting average congestion window is 3 MSS.

Claim 5 *Steady state congestion window $cwnd_{MAX}^{ATK}$ of the TCP sender when under packet reordering attack, with network and communication model as in Section 2.3, and attacker model in Section 2.2, Figure 3, is $cwnd_{MAX}^{ATK} \leq \frac{3}{2RTT} + 2$*

Proof Assume that at time t_0 the sender is in slow start or in congestion avoidance state (in Figure 5) and it receives three duplicate ACKs. Let t_0^- be the time right before arrival of three duplicate ACKs, and t_0^+ be the time right after the arrival of three duplicate ACKs, and denote by $cwnd(t_0^-)$ the congestion window size at time t_0^- (this is also the number of ACKs that will arrive following time t_0^-). For each duplicate ACK the $dupACKcnt$ variable is incremented, and once $dupACKcnt = 3$, the TCP at the sender transitions to fast recovery (steps (1.e) or (2.d) in Figure 5): sets the $ssthresh$ to $\frac{1}{2}cwnd(t_0^-)$, and sets the congestion window to $cwnd(t_0^+) = ssthresh + 3$, fast retransmits the 'missing segment', and switches to fast recovery. When an ACK (for new data) at time $t_{0,1}$ arrives (phase (3.c) in Figure 5), the sender sets the congestion window to $cwnd((t_{0,1})^+) = ssthresh$, sets $dupACKcnt = 0$, and transforms to congestion avoidance. The number of pending (transmitted but not yet ACKed) segments at time $(t_{0,1})^+$ is: $cwnd(t_0^-) + 1$, which is greater than the congestion window size: $\frac{1}{2}cwnd(t_0^-)$, therefore, according to Figure 5, the sender cannot transmit new segments into the network (the sender can transmit segments when the amount of transmitted but yet to be ACKed (a.k.a. pending) segments is less than the size of the congestion window). Below is an expression to calculate the number k of ACKs required for the sender to resume transmission of new data segments: $\frac{1}{2}cwnd(t_0^-) + k = cwnd(t_0^-) + 1 \Longrightarrow k = \frac{1}{2}cwnd(t_0^-) + 1$. Namely, the sender can resume transmission after receipt of $\frac{1}{2}cwnd(t_0^-)+1$ ACKs, i.e., ACKs arriving at: $t_{0,1}, ..., t_{0,k}$, for $k = \frac{1}{2}cwnd(t_0^-)+1$. The number of remaining ACKs to arrive is $\frac{1}{2}cwnd(t_0^-)-5$, i.e., ACKs that will arrive at time: $t_{0,k+1}, ..., t_{0,2k-1}$. Since in congestion avoidance the congestion window grows linearly, roughly by $MSS\frac{MSS}{cwnd}$ for each ACK (according to step (2.a) in Figure 5), the amount of segments that the sender will transmit in $[t_0^+, (t_0 + RTT)^-]$ (and the congestion window size at time $(t_0 + RTT)^-$) is given by:

$$\begin{aligned} cwnd((t_0 + RTT)^-) &< \frac{cwnd(t_0^-)}{2} + \frac{cwnd(t_0^-) - 4}{\frac{1}{2}cwnd(t_0^-)} = \\ &= \frac{1}{2}cwnd(t_0^-) + 2 - \frac{8}{cwnd(t_0^-)} \end{aligned}$$

[8]Similarly to attack in Section 3, optimal amplification impact is achieved when attacker speeds up exactly three segments. If less than three packets arrive out of order, the sender will not reduce the congestion window but instead will wait for a timeout. However, since the segments transmitted by the sender eventually arrive, and ACKs for them are returned, no timeout will occur. Note that some TCP implementations may expect for more than three duplicate ACKs before reducing transmission rate; adopting the attack to these cases is trivial.

Since $\frac{8}{cwnd(t_0^-)} > 0$ the number of transmitted segments in interval $[t_0^+, (t_0+RTT)^-]$ is at most $\frac{1}{2}cwnd(t_0^-)+2$; and $cwnd((t_0+RTT)^-) < \frac{1}{2}cwnd(t_0^-)+2$. At time $t_0 + RTT$ an ACK for the segment (retransmitted at time t_0^+) should arrive (since we assume no loss and constant delays), followed by at most $\frac{1}{2}cwnd(t_0^-)+2$ ACKs in response to earlier transmitted segments, i.e., the segments transmitted in interval $[t_{0,k+1}, (t_0+RTT)^-]$. The congestion window size by the end of transmission round (t_0+RTT) (at time $(t_0+2RTT)^-$), is

$$\begin{aligned} cwnd((t_0+2RTT)^-) &< cwnd((t_0+RTT)^-) + \\ &+ \Big(\frac{\frac{cwnd(t_0^-)}{2}}{\frac{cwnd(t_0^-)}{2}}\Big) = \frac{cwnd(t_0^-)}{2} + 3 \end{aligned}$$

Namely, congestion window, $cwnd$, increases by at most one MSS in each transmission round (RTT). By next attack epoch, at time t_1, the congestion window $cwnd(t_1^-)$ will have grown by at most $\frac{T}{RTT}$, and holds $cwnd(t_1^-) < \frac{1}{2}cwnd(t_0^-) + \frac{T}{RTT} + 1$. More generally, by i^{th} attack epoch the congestion window $cwnd(t_i^-)$ will be:

$$\begin{aligned} cwnd(t_i^-) &< \frac{cwnd(t_{i-1}^-)}{2} + \frac{T}{RTT} + 1 = \\ &= \frac{cwnd(t_0^-)}{2^{i+1}} + \sum_{j=0}^{i}\Big(\frac{T}{(j+1)RTT} + \frac{1}{2^j}\Big) = \\ &= 2\Big(\frac{T}{RTT}+1\Big) + \\ &+ \frac{1}{2^{i+1}}\Big(cwnd(t_0^-) - 2(\frac{T}{RTT}+1)\Big) \end{aligned}$$

The whole expression approximates $\frac{2T}{RTT}$, thus the bound on congestion window size at time t_i^- is $cwnd(t_i^-) \le \frac{2T}{RTT}$. For $T = 3/\rho$, $cwnd(t_i^-) \le \frac{6}{\rho \cdot RTT}$. In addition, since for every t_i holds: $cwin_{ATK}^{MAX} \le cwnd(t_i)$, the bound on the steady state congestion window size when under attack is $cwin_{ATK}^{MAX} \le \frac{6}{\rho \cdot RTT}$. □

Claim 6 *The throughput of steady state congestion window is at most* $\frac{6}{\rho \cdot RTT^2}$.

Proof To compute throughput we take the average steady state window size derived in Claim 5, and divide it by the RTT:

$$\frac{2\Big(\frac{T}{RTT}+1\Big) + \frac{1}{2^{i+1}}\Big(cwnd(t_0^-) - 2(\frac{T}{RTT}+1)\Big)}{RTT} \le$$

$$\le \frac{6}{\rho \cdot RTT^2}$$

□

5.3. Protecting TCP from Stealth DoS Attacks

Packets reordering (whether by malicious attacker or due to benign network conditions) has a negative effect on TCP throughput. Upon receipt of three duplicate ACKs TCP sender will trigger fast retransmit and will resend a 'believed to be lost' segment, resulting in wasted bandwidth, and reduction of sending rate at the sender. Two independent directions can be pursued in order to address the reordering and specifically the attack in Section 5.1. One direction is to adjust TCP to diverse network conditions, i.e., to immune TCP to packet reordering. A wide range of TCP modifications has been proposed to improve robustness to reordering, e.g., [8, 31, 41, 42, 10, 6, 40, 11, 9]; see a survey in [30] and an analysis in [8]. The main idea of all those solutions is to detect and ignore false duplicate ACKs. Sender halves the congestion window upon duplicate ACK, but then restores it back when receiver signals receipt of 'supposedly lost' segment, thus resulting in an insignificant slowdown. Yet none of the proposed solutions is widely adopted, mainly since changing TCP requires a change in every end host, and may take considerable time to adopt.

Algorithm 1: Implementation of the fix to TCP in the sending gateway $GW2$. Gateway will inspect incoming ACK segments and will delay response to congestion, i.e., duplicate ACKs. If new ACK is received all duplicate ACKs are discarded, otherwise, when typical delay is reached all ACK segments are forwarded to sending host behind the gateway.

```
Incoming Segment $ack$ from Internet
    if $ack.SN == SN$ then
        //it's a duplicate ACK
        if $dupACKctr == 0$ then
            $delay \leftarrow set\_delay()$;
            $timer \leftarrow set\_timer(delay)$;
            $dupACKctr \leftarrow 1$;
        if $dupACKctr < 2$ then
            $forward(ack)$;
        $dupACKctr++$;
    if $ack.SN > SN$ then
        $SN \leftarrow ack.SN$;
        if $dupACKctr > 0$ then
            $dupACKctr \leftarrow 0$;
            $stop\_timer()$;
    if $timer == timeout$ then
        for $i = dupACKctr$ to $2$ do
            $forward(ack)$;
        end
        $dupACKctr \leftarrow 0$;
        $stop\_timer()$;
end
```

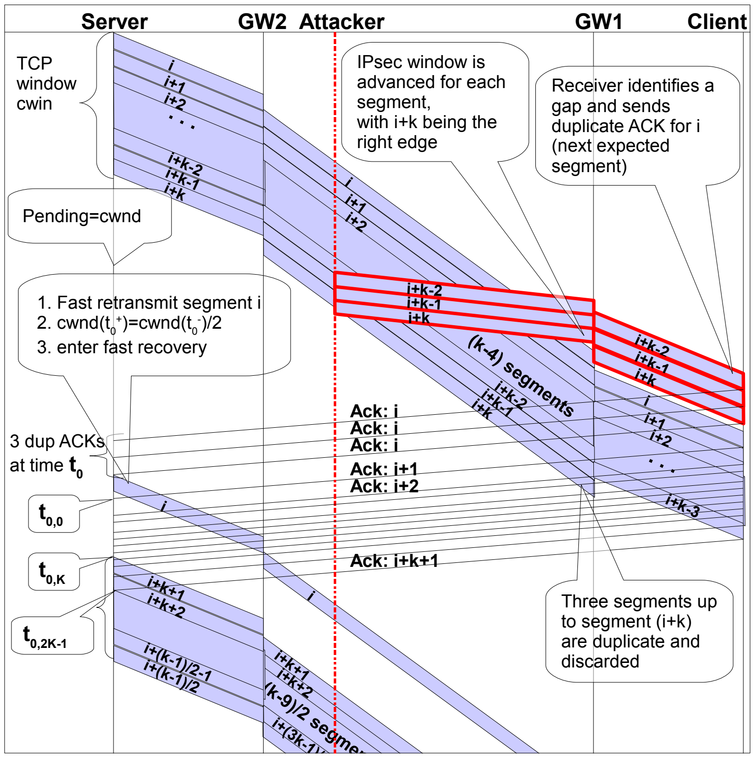
Server
GW2
Attacker
GW1
Client
TCP window cwin
Pending=cwnd
IPsec window is advanced for each segment, with i+k being the right edge
Receiver identifies a gap and sends duplicate ACK for i (next expected segment)
1. Fast retransmit segment i
2. cwnd(t0+)=cwnd(t0-)/2
3. enter fast recovery
3 dup ACKs at time t0
(k-4) segments
Ack: i
Ack: i
Ack: i
Ack: i+1
Ack: i+2
Ack: i+k+1
t0,0
t0,K
t0,2K-1
Three segments up to segment (i+k) are duplicate and discarded
i+(k-1)/2-1
i+(k-1)/2
(k-9)/2 segments

Figure 10: Throughput degradation attack on communication over TCP with infinitely large IPsec anti-replay window, i.e., packets are not discarded by IPsec implementation at the receiving gateway. Attacker reorders segments, thus generating a gap in sequence numbers at the receiver, which responds with duplicate ACKs. Upon receipt of three duplicate ACKs, the sender reduces its sending rate. Specifically, TCP at the sender transmits a window of $k+1$ segments, and the attacker speeds up last three segments, i.e., segments with sequence numbers $i+k-2, i+k-1, i+k$, such that they arrive before 1^{st} segment in the window, i.e., segment with sequence number i. These segments are passed by the gateway to the client (since they are authentic and not a replay), and trigger three duplicate ACKs for i^{th} segment at the receiver. Upon receipt of three duplicate ACKs the sender retransmits the missing segment i, and reduces its transmission rate. At time t_0 the sender receives three duplicate ACKs (this initiates first attack epoch); also note that $cwnd = pending$. The sender fast retransmits lost segment, and reduces its transmission rate. The sender cannot resume transmission of further segments since the number of pending segments is larger than the congestion window size. After receipt of sufficient number of ACKs the sender can resume transmission of new segments.

We focus on a solution that requires a modification to IPsec gateways. Solution in firewalls does not require changing each host separately, but only to apply the modification to the firewall, and as a result to protect subnet of hosts. Many private networks connected to the Internet are protected by firewalls. Firewall protection is based on the idea that all packets destined to hosts behind a firewall have to be examined by the firewall. The drawback of this approach is the additional processing delay on every packet, and having the firewall maintain state. On the other hand, many firewalls examine TCP connections for security reasons, e.g., solution to SYN attack, thus firewalls keep state. Therefore, we believe that our addition is minimal. Our solution is applied to the sending IPsec gateway (and no change to receiving gateway $GW1$) and is comprised of two phases: first detection, then prevention of an attack. Note that our suggestion can also be applied to TCP in every host, yet we leave it as a further research.

The main idea of our proposition is to delay response to congestion, i.e., duplicate ACK, in the sending gateway (w.l.o.g. $GW2$ in Figure 1) and not deliver to the sending host behind $GW2$ until maximal delay is reached. The sending gateway $GW2$ will measure delay of outgoing segments[9] between itself and the receiving gateway $GW1$ (for every VPN), and will use these measurements to estimate typical delay for outgoing packets. In addition, gateway $GW2$ will store time, and sequence number of outgoing segments. These can be maintained in an array, and upon arrival of an ACK, the corresponding sample is released. If a duplicate ACK for some segment, e.g., segment with sequence number i, arrives, gateway $GW2$ will approximate

[9]The gateway will estimate a typical delay between itself and the receiving gateway, similarly to existing approaches that estimate TCP timeout.

the sending time, based on the stored samples, of segment i. More specifically, the gateway will locate in the array the interval of the sending time of missing segment (for which a duplicate ACK was received); and (based on the delay measurements it recorded) will delay the duplicate ACK (and all subsequent duplicate ACKs for that segment) till typical (plus possibly some safety margin) delay is reached. If ACK with higher sequence number arrives before maximal delay, all duplicate ACKs are discarded, and the new ACK is forwarded to the sending host. However if no new ACK arrives, once maximal delay is reached, all duplicate ACKs are released and forwarded to the sending host (see pseudocode in Algorithm 1).

Note that storing and maintaining duplicate ACKs may impose an overhead on the gateway, therefore we propose the following implementation: sending gateway $GW2$ will store the sequence number SN of the most recently received ACK, and will maintain duplicate ACKs counter $dupACKctr$. For each subsequent duplicate ACK for the same SN, the counter will be incremented. This addition requires the gateway only to store the sequence number (32 bits) of last correctly received packet, and counter with number of duplicate ACKs for last correctly received segment. We defer detailed analysis of the fix to full version of the paper.

6. Further Research Directions

Below we present several future research directions:

- An important further research is defining secure channel protocol with performance guarantees, that would ensure security and efficiency to traffic above it, or to prove that no such channel exists.
- It is interesting to consider a weaker attacker without speed-up advantage, i.e., attacker may be using the same channel as the legitimate parties, or another with the same delay, and to perform analysis of performance degradation (and probabilities given distribution on traffic).
- Further work is required to analyse our suggested fix to TCP in IPsec gateway as a solution to benign (as well as malicious) packet reordering on Internet. In particular, to perform simulations, and experiments showing the impact on efficiency when under attack, as well as under bening network reordering.

References

[1] I. Aad, J. Hubaux, and E. Knightly. Denial of service resilience in ad hoc networks. In *Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 202–215. ACM New York, NY, USA, 2004.

[2] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581 (Proposed Standard), Apr. 1999. Updated by RFC 3390.

[3] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control, RFC 2581. *Internet request for comments*, 1999.

[4] J. Bellardo and S. Savage. Measuring packet reordering. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 97–105. ACM New York, NY, USA, 2002.

[5] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. *Lecture notes in computer science*, pages 26–45, 1998.

[6] S. Bhandarkar and A. Reddy. TCP-DCR: Making TCP robust to non-congestion events. *Lecture Notes in Computer Science*, pages 712–724, 2004.

[7] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. RFC2475: An Architecture for Differentiated Services. *RFC Editor United States*, 1998.

[8] E. Blanton and M. Allman. On making TCP more robust to packet reordering. *ACM SIGCOMM Computer Communication Review*, 32(1):20–30, 2002.

[9] E. Blanton and M. Allman. Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions. RFC 3708 (Experimental), Feb. 2004.

[10] S. Bohacek, J. Hespanha, J. Lee, C. Lim, and K. Obraczka. TCP-PR: TCP for persistent packet reordering. In *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*, pages 222–231, 2003.

[11] S. Bohacek, J. Hespanha, J. Lee, C. Lim, and K. Obraczka. A New TCP for Persistent Packet Reordering. *IEEE/ACM TRANSACTIONS ON NETWORKING*, 14(2):369, 2006.

[12] R. Canetti and H. Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology*, pages 337–351. Springer-Verlag London, UK, 2002.

[13] R. Chang. Defending against flooding-based distributed denial-of-service attacks: A tutorial. *IEEE Communications Magazine*, 40(10):42–51, 2002.

[14] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), Aug. 2008.

[15] G. Dommety. Key and Sequence Number Extensions to GRE. RFC 2890 (Proposed Standard), Sept. 2000.

[16] Y. Gilad and A. Herzberg. Lightweight opportunistic tunneling (lot). In *ESORICS*, pages 104–119, 2009.

[17] M. Gouda, C. Huang, and E. Li. Anti-replay window protocols for secure IP. In *Computer Communications and Networks, 2000. Proceedings. Ninth International Conference on*, pages 310–315, 2000.

[18] M. Guirguis, A. Bestavros, and I. Matta. Exploiting the transients of adaptation for RoQ attacks on Internet resources. In *Network Protocols, 2004. ICNP 2004. Proceedings of the 12th IEEE International Conference on*, pages 184–195, 2004.

[19] M. Guirguis, A. Bestavros, I. Matta, and Y. Zhang. Reduction of quality (RoQ) attacks on Internet end-systems. In *Proceedings IEEE INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, 2005.

[20] G. Hellstrom. RTP Payload for Text Conversation. RFC 2793 (Proposed Standard), May 2000. Obsoleted by RFC 4103.

[21] A. Herzberg and I. Yoffe. The layered games framework for specifications and analysis of security protocols. *International Journal of Applied Cryptography*, 1(2):144–159, 2008.

[22] K. Hickman and T. Elgamal. The SSL protocol. *Netscape Communications Corp*, 1995.

[23] K. Houle, G. Weaver, N. Long, and R. Thomas. Trends in denial of service attack technology. *CERT Coordination Center*, 2001.

[24] C. Huang and M. Gouda. An anti-replay window protocol with controlled shift. In *Proceedings of the 10th IEEE International Conference on Computer Communications and Networks*, 2001.

[25] C. Kaufman, R. Perlman, and B. Sommerfeld. DoS protection for UDP-based protocols. In *Proceedings of the 10th ACM conference on Computer and communications security*, page 7. ACM, 2003.

[26] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401 (Proposed Standard), Nov. 1998. Obsoleted by RFC 4301, updated by RFC 3168.

[27] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), Dec. 2005.

[28] J. Kurose, K. Ross, and K. Ross. *Computer networking: a top-down approach featuring the Internet*. Addison-Wesley Reading, MA, 2003.

[29] A. Kuzmanovic and E. Knightly. Low-rate TCP-targeted denial of service attacks: the shrew vs. the mice and elephants. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 75–86. ACM New York, NY, USA, 2003.

[30] K. Leung, V. Li, and D. Yang. An overview of packet reordering in transmission control protocol (TCP): problems, solutions, and challenges. *IEEE Transactions on Parallel and Distributed Systems*, 18(4):522–535, 2007.

[31] R. Ludwig and R. Katz. The Eifel algorithm: making TCP robust against spurious retransmissions. *ACM SIGCOMM Computer Communication Review*, 30(1):30–36, 2000.

[32] X. Luo and R. Chang. On a new class of pulsing denial-of-service attacks and the defense. In *Proceedings of the ISOC Symposium on Network and Distributed Systems Security (SNDSS)*, pages 61–79, 2005.

[33] M. Maxim and D. Pollino. *Wireless security*. McGraw-Hill Osborne Media, 2002.

[34] V. Paxson and M. Allman. RFC2988: Computing TCP's Retransmission Timer. *RFC Editor United States*, 2000.

[35] J. Postel. Transmission Control Protocol. RFC 793 (Standard), Sept. 1981. Updated by RFCs 1122, 3168.

[36] K. Ramakrishnan, S. Floyd, and D. Black. The addition of explicit congestion notification (ECN) to IP, 2001.

[37] C. Schuba, I. Krsul, M. Kuhn, E. Spafford, A. Sundaram, and D. Zamboni. Analysis of a Denial of Service Attack on TCP. In *1997 IEEE Symposium on Security and Privacy, 1997. Proceedings.*, pages 208–223, 1997.

[38] W. Stevens. *TCP/IP illustrated (vol. 1): the protocols*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1993.

[39] P. Syverson. A taxonomy of replay attacks [cryptographic protocols]. In *Computer Security Foundations Workshop VII, 1994. CSFW 7. Proceedings*, pages 187–191, 1994.

[40] F. Wang and Y. Zhang. Improving TCP performance over mobile ad-hoc networks with out-of-order detection and response. In *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 217–225. ACM New York, NY, USA, 2002.

[41] M. Zhang, B. Karp, S. Floyd, and L. Peterson. Improving TCP's Performance under Reordering with DSACK. *International Computer Science Institute, Berkeley, Tech. Rep. TR-02-006*, 2002.

[42] M. Zhang, B. Karp, S. Floyd, and L. Peterson. RR-TCP: A reordering-robust tcp with dsack. In *11th IEEE International Conference on Network Protocols, 2003. Proceedings*, pages 95–106, 2003.

[43] F. Zhao and S. Wu. Analysis and improvement on IPSec anti-replay window protocol. In *Computer Communications and Networks, 2003. ICCCN 2003. Proceedings. The 12th International Conference on*, pages 553–558, 2003.