TCP ack storm DoS attacks

Conference Paper *in* Computers & Security · June 2011

DOI: 10.1007/978-3-642-21424-0_3

CITATIONS
19

READS
3,736

2 authors, including:

Amir Herzberg
University of Connecticut
278 PUBLICATIONS **7,767** CITATIONS

SEE PROFILE

TCP Ack Storm DoS Attacks

Raz Abramov, Amir Herzberg

Bar Ilan University

Abstract

We present *Ack-storm DoS attacks*, a new family of DoS attacks exploiting a subtle design flaw in the core TCP specifications. The attacks can be launched by a very weak MitM attacker, which can only eavesdrop occasionally and spoof packets (a *Weakling in the Middle (WitM)*). The attacks can reach theoretically unlimited amplification; we measured amplification of over 400,000 against popular websites before aborting our trial attack.

Ack storm DoS attacks are practical. In fact, they are easy to deploy in large scale, especially considering the widespread availability of open wireless networks, allowing an attacker easy WitM abilities to thousands of connections. Storm attacks can be launched against the access network, e.g. blocking address to proxy web server, against web sites, or against the Internet backbone. Storm attacks work against TLS/SSL connections just as well as against unprotected TCP connections, but fails against IPsec or link-layer encrypted connections.

We show that Ack-storm DoS attacks can be easily prevented, by a simple fix to TCP, in either client or server, or using a packet-filtering firewall.

Keywords: Denial of service, TCP, Secure network protocols.

1. Introduction

Most works in cryptography today adopt the all-powerful *Man In The Middle (MitM)* attacker model. The MitM attacker controls all of the traffic in the channels under him, with the ability to see, block and modify any package in the channel. In contrast, most works on Denial of Service (DoS) attacks, investigate the damage which much weaker attackers can cause, in order to focus on the most feasible and realistic attacks. Such weak attackers may only have the ability to send spoofed packets, or even weaker abilities -

sending raw packets, sending only well-formed packets, or even merely issuing HTTP requests (e.g., puppets, see [**?**]).

In this work, we present and investigate the *Weakling In The Middle (WitM)* attacker model. The WitM attacker can eavesdrop on communication, but with significant limitations, mainly: eavesdropping only to one side of the connection, and receiving only a small percentage of the packets sent. These limitations are inspired by real-world wireless eavesdropping abilities, especially to open wireless networks; the 'one-sided' limitation is due to the fact that often the attacker is only able to eavesdrop to communication from the access point, and the low percentage is due to the weak reception by a remote eavesdropper. Open wireless networks are becoming more and more common, whether its in restaurants, malls or even as a city-wide infrastructure [**?**]. Attackers today can eavesdrop on public networks from a distance without the need for special equipment, and with poor reception quality (capturing low percentage of packets). It is widely known how to make a directional('Yagi') antenna, that can reach up to 12 miles of range and cost no more than a few dollars (see [**?**] or numerous web pages).

In addition to their limited eavesdropping capabilities, WitM attackers can also send spoofed packets to the network. This ability is very common, since many ISPs fail to properly deploy Ingress filtering [**?**]. However, we restrict the number of packets that the attacker can send into the network per attack; real attackers will try to restrict the number of packets they send, in order to stay hidden and avoid capture. Note also that sending few packets per attack increases the number of attacks that the attacker can perform simultaneously. These aspects are similar to the stealth attacker model of [**?**].

We present several *'Ack-Storm DoS Attacks'* that, by injecting (two or more) packets into an existing TCP connection, cause a long exchange of TCP packets between a client and a server, terminated only by connection reset or packet losses. This way enables a WitM attacker to disrupt services to local and regional junctions in the Internet infrastructure, as well as well as to individual web sites and services.

The Ack-storm behavior of TCP has been mentioned before in [**?**] and [**?**] as a side effect of TCP hijacking attacks, and thus as something to be minimized and prevented.

We present a typical scenario in Fig. **??**, with a client Alice connected to an open wifi network AliceNet. Alice is connected to a remote web server Bob, over a standard TCP based connection, such as HTTP, SSL etc. The

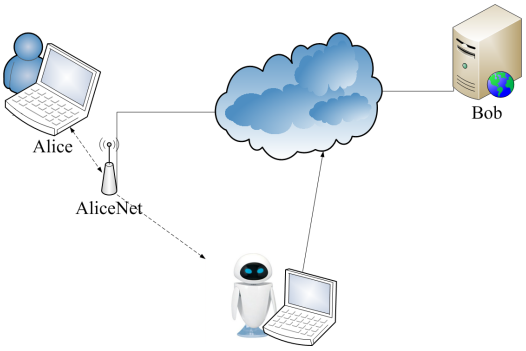

Figure 1: Example attacker model - Alice is connected through the wireless access point AliceNet, to a remote web server Bob. Eve is able to receive occasional traffic from Alice's network. In addition, Eve's ISP does not filter traffic, so Eve is able to send spoofed packets to the Internet.

Attack	Highest Measured Ampl.	Ampl. Attacker
Two-Pkt Ack Storm	261,000	WitM
N Ack Storm	261,000	WitM
Everlasting Ack Storm	400,000	WitM
Opt Ack [?]	251.6	Client
Smurf [?]	≤ 1000	Spoofer
DNS Amp. [?]	73	Spoofer

Figure 2: Comparing DoS Amplification Attacks

attacker (Eve) has two abilities: eavesdropping and spoofing. Eve has a receiver antenna, with which she able to eavesdrop on (a small percentage of) packets sent by the access point over AliceNet. Eve in not able to inject packets into AliceNet, because of the long distance between them. Eve is also able to send raw packets into the Internet via its ISP. We assume that Eve's latency to both Alice and Bob is higher than the latency between them. Also, Eve cannot delay, drop, or otherwise affect any traffic sent in the network.

The Ack-storm attacks are based on the fact that, upon receiving a packet with the acknowledge number field (the receiver's sequence number) larger than the one sent by the receiving client, the client must, according to the TCP standard [?], resend the last sent acknowledgment packet to the other side, and discard the received packet. A design flaw in TCP causes the client and the server to be trapped in an infinite loop of sending and receiving empty acknowledgment packets.

The basic attack - Two Packets Ack Storm, as performed by the attacker, consists of three main stages:

1. Pick up (at least) one packet from a TCP connection between a client and a server.
2. Generate two packets, each addressed to one party and with sender

address of the other party (i.e. spoofed). The packets must be inside the TCP windows of both sides. The packets should have content - at least one byte of data.

3. Send the packets to the client and the server at the same time. The connection will then enter an infinite loop of sending ack packets back and forth between both parties.

The *N-packets Ack Storm* attack and the *Everlasting Ack Storm* attack offer further amplification to the Two-packets Ack Storm attack, consuming more bandwidth and increasing the duration of the session. In our experiments, we measured an amplification factor of over 400,000, when performing the Everlasting Ack Storm attack - the highest amplification rate measured until today (see Fig. **??** for comparison with existing amplification attacks).

When utilized by a WitM attacker, Ack-storm DoS attacks target traffic in the reception range of the attacker's wireless card. As such, the attack's traffic bottleneck is the local Internet infrastructure. Most of the non-encrypted HTTP-based traffic in the local network is handled by the local proxy server. When the attack takes place, it overloads the proxy, making it unavailable to the general public.

Another possibility of the attack is to target a specific web site. In order to assure that the HTTP traffic to the site indeed reaches the web site's servers, and not handled by a proxy, the attacker only targets SSL-based traffic. SSL traffic is encrypted and is guaranteed to be handled by the web site's own servers' and not by any other entity. Among the web sites that always use an encrypted connection we can find banks, Email providers and other sites containing personal information of the connecting client.

In addition to the attacks we present Effective Amplification , an innovative way to measure the amplification of DoS attacks. Effective Amplification takes into account the individual charecaristics of the attacked protocol, and presents a more accurate way to measure the effectiveness of attacks against specific targets. We believe that by using Effective Amplification we can incorporate the attack scenario into the attack's analysis, creating a better image of the attack's effect in a specific environment.

1.1. Related Works

Denial of Service attacks (DoS), and especially Distributed Denial of Service attacks (DDoS), pose a serious threat to the Internet today. In this section we review some of the attacks that exist today, with emphasis on

amplification attacks, their relative strength and the potential damage that they can cause.

The classical DDoS attacks are flooding attacks. These attacks overflow the victim with a large amount of traffic, causing loss of packets and termination of connections (see [**?**]). SYN Flooding [**?**] is another type of DDoS attack, overflowing the client with SYN packets and preventing it from servicing valid connections. The classic DDoS attack require a large number of compromised nodes in order to create the data volumes required, making them unrealistic to implement on strong enough targets (Content Delivery Networks, Internet Backbone etc.).

Puppetnets [**?**] consist of a large numbers of computers, which run - for limited time - malicious code, but only within a restricted environment ('sandbox'), typically, script or applets automatically executed from remote, untrusted websites. Puppetnets enable attackers to generate large amounts of traffic from random sources to the victim. In order to construct large scale puppetnets, the attacker has to control the content of a high traffic web site, making the attack set up challenging.

Amplification DDoS attacks increase the volume of traffic the attacker can generate, by utilizing some automated, benign mechansims. This enable the attacker to use a smaller number of compromised clients, yet generate significant bandwidth; this can make large scale attacks feasible to attacers with limited resources. Attacks such as the Smurf attack and DNS Amplification attack [**?**] can increase the attacker's strength by a factor of 100's, while only requiring spoofing abilities.

A much higher theoretical amplification factor is achieved in the Opt Ack attack [**?**]. The Opt Ack attack requires control on the attacking computers, and causes DoS to the attacking hosts as well as to the victims.

While DDoS attacks usually focus on a specific host, they can also target a network junction. The Coremelt attack [**?**] is an example of creating an overflow in the network core, and not in a specific end node. These attacks are far harder to block, since attacking traffic is similar to benign traffic (from each host).

DoS attacks rely on different attack models, giving the attacker different abilities and limitations. The MitM attacker is the most powerful attacker; there are trivial DoS attacks for MitM attackers, see for example the TCP RST attack [**?**]. But MitM ability is hard to come by, so MitM based DoS attacks are rare in practice. Spoofer attackers are far more common, due to the lack in Ingress filtering in ISPs [**?**], and the ability to spoof

on open wireless networks [**?**]. Eavesdropping ability was hard to achieve until the wireless age, but today open wireless networks are common, making eavesdropping possible for anyone with a laptop.

The WitM attacker model and the stealth attacker of [**?**]) are limited variants of MitM attackers, with the ability to eavesdrop on a small percentage of traffic, and spoof packets into the Internet. These models assume realistic abilities often available to real attackers.

The Acknowledge loop behavior of TCP has been mentioned before in [**?**] as a side effect of TCP hijacking attacks, and thus as something to be minimized and prevented. The Ack Storm attack, as a side effect of TCP session hijacking attacks, is also discussed briefly in [**?**], [**?**] and [**?**].

1.2. Contributions

This paper presents the following contributions:

- We present the WitM attack model and demonstrate how a WitM attacker can perform DoS attacks.
- We present the Ack-storm DoS attacks. These are powerful attacks, requiring low resources (low probability to intercept packets from the network, low bandwidth requirements) from the attacker and providing the highest amplification factor measured until today.
- Effective Amplification measures attacks' strength far more effectively than traditional amplification methods.
- The Ack-storm DoS attacks demonstrate an advantage of the use of IPSec over the use of TLS/SSL. SSL connections are vulnerable to the Storm attacks, and even help the attacker target the servers and not the web proxy. IPSec, on the other hand, is immune to the attack, as it does not reveal TCP connection details to an eavesdropper attacker.

Section **??** to **??** present the Ack Storm attacks and two variations that increase the attack's effectiveness. Section **??** presents analysis and experimental results of the attack to cause congestion in a network. Section **??** presents Effective Amplification and discusses its advantages in relation to the traditional amplification model. Section **??** discusses various network challenges an attacker can come across when trying to perform an attack. Section **??** discusses defenses that can prevent the attack.

2. Two-Packets Ack-storm Attack

In this section we present the flaw in the TCP standard that enables the Ack-storm DoS attacks, describe the Two-Packets Ack-storm attack and explain the strengths and weaknesses of this attack.

2.1. The TCP RFC Flaw

The TCP RFC [**?**] defines all states and actions in a TCP connection. According to the RFC, the way to handle false data is, usually, to drop the packet. This behavior is recommended as it does not allow an attacker to trigger a response from either party by injecting false packets into the stream. However, there is one exception: When a TCP connection in ESTABLISHED state, and a packet is received with an ACK field that acknowledges data not yet sent, the client must act as follows (described in page 71 of the RFC):

1. Send an ACK (the last sent).
2. Stop processing ('drop') the segment. In particular, *ignore* the payload in the segment.

We did not find any documentation motivating this exception. Maybe the thought was that the packet must have been corrupted, and hence responding with the previously sent ACK is similar to sending of duplicate ACK in case of packets received out of order (and may trigger retransmission). However, this event is extremely rare, and we do not anticipate any significant impact on TCP's performance by simply silently discarding such packets (not resending ACK). Indeed, sending an acknowledge packet in response to a malformed packet is not recommended. This behavior is what makes the Ack-storm DoS attacks possible.

2.2. Attack Description

To initiate the Two-Packets Ack-storm the attacker sends two packets containing data: one to either side of a TCP connection. The attack uses the RFC flaw described above in order to cause the client and server to send false acknowledgment packets back and forth. No additional data could be sent once the attack takes place: every packet sent from now on will contain ack number higher then the one the receiving party has, and hence will only cause generation of another (malformed) ACK. If either side trys to send additional data over the channel (assuming the TCP send window is not full), the packets will increase the strength of the attack by creating additional 'sub

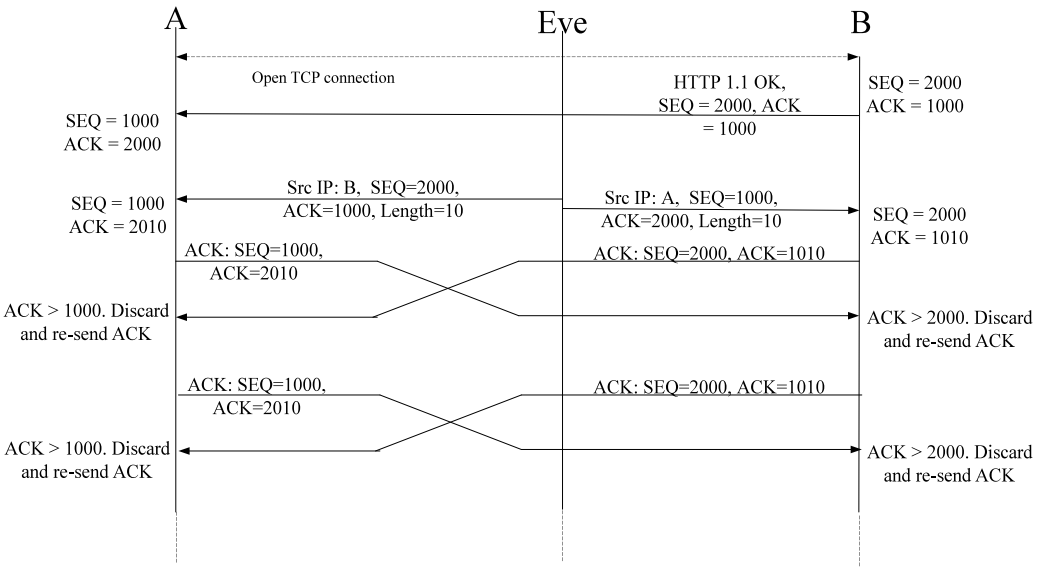

Figure 3: The Two-Packets Ack-storm attack. Numbers are for illustration only. The attacker sends both the client (impersonating as the server) and server (as the client) a message with length 10. Both sides send an ACK, while advancing their ACK number by 10. When the packets arrive at the other side, they contain an ACK field higher than the actual data sent. The client and server then send (according to the standard) the last ack sent by them, which triggers the loop all over again.

sessions' of acknowledge packets (additional explanation can be found in the next section). Since according to the standard such packet must be dropped, and its data discarded, neither side can increase its sequence number, making it impossible for the sequence numbers to re-synchronize.

Figure **??** demonstrates the message passing between the client, server and attacker. By sending packets to both sides simultaneously, the attacker raises both the client and server's reserved ack numbers. This will make it impossible for them to overcome the false data sent, as all packets sent from that point will be considered false due to the ACK field being too high.

By sending a minimum of two packets, the Two-Packets Ack-storm attack can cause hundreds of thousands of acknowledgment packets to be sent over a single TCP session. The figures, presented in the experiments section (Tab. **??**) show that by sending only two packets, each with the minimal length of an Ethernet packet size (64 bytes), we can cause an amplification factor of over 261,000 times the original sending size.

The attack scenario is illustrated below (and in Fig. **??**); to illustrate, we

assume initial values of $A.SEQ = 1000(= B.ACK)$, and $B.SEQ = 2000(= A.ACK)$.

1. Eve sends A and B packets of length 10, each on behalf of the opposite side.
2. Upon receiving the packet, A advances A.ACK to be 2010, and sends an ack to B. B advances B.ACK to be 1010 and sends an ack to A.
3. When B receives a packet with $A.ACK = 2010$, when $B.SEQ = 2000$, he acts according to the standard: discards the packet and re-sends A the ack (in which $B.ACK = 1010 > A.SEQ$). A does the same, as it received a packet from B with $B.ACK = 1010$.
4. Both A and B receive packets with the ACK number bigger than their SEQ. The behavior in step 3 is performed again.
5. The loop continues when both parties keep receiving packets with an ACK larger than their sequence numbers, stopping only when both packets are dropped, or when one side reaches a timeout and ends the connection by RST.

The state that both parties are in during the attack is described in Fig. **??**. Each side's ACK number is bigger than the actual sequence number of the other side. Since the TCP standard not only does not enable repairing an ACK backwards, but forces both parties to re-send the last sent packet - the connection enters an 'infinite' loop. This loop is only interrupted, by a RST packet, after the maximal number R of retransmissions; e.g. $R = 11$ for Apache servers we tested.

2.3. Analysis: Maximum Amplification Ratio

The attacker sends one packet to the client and one to the server, both with length of the minimum Ethernet packet size - 64 bytes. Each of the two packets sent by the attacker causes ACK packets (each of length 64 bytes) to be sent back and forth until the connection is terminated by the server. We are using the fact that the minimal length of an Ethernet packet is 64 bytes.

The two packets sent at the beginning of the attack are sent back and forth between the client and server, creating two 'sub sessions' of traveling packets. The attack continues until a RST packet terminates the attack after the maximal number R of retransmissions is sent; let $Time_r$ denote the time of the r^{th} retransmission (and $Time_R$ the time of the last retransmission before reset). During the total time of the attack, i.e., $Time_R$, these two

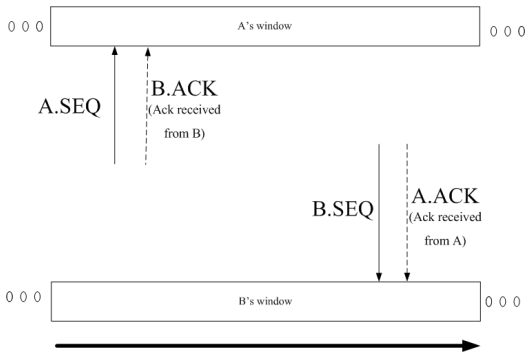

Figure 4: A and B's congestion window status when the attack had started. Both sides think the opposite side has a sequence number larger than it actually has. This leads to an infinite loop, where both sides send and re-send acks with wrong ack numbers, triggering the other side to send a wrong ack in return.

'sub sessions' of packets would have caused a total of $128 \times \frac{R}{\rho}$ bytes sent, where ρ denotes the round trip time (RTT). Notice that the shorter ρ (the RTT between the client and server), the more effective the attack.

Since the attack interrupts an active session, altering the sequence numbers as it does so, acknowledgments of already sent packets are dropped. Therefore, the unacknowledged packets will be retransmitted by the sender. Since no retransmission will succeed, the sender will eventually give up and abort the connection.

Table **??** shows the retransmission scheme of Apache web server - the most common web server in the Internet today. Transmission times are all based on our experiments. From the table we can see that after $Time_R$ =225 seconds the server resets the connection. The server would have retransmitted ten times, in each the time waited between retransmissions is roughly doubled[1].

Each retransmission packet that the server sends contains an acknowledgment number higher then the client's actual sequence number. Therefore, each retransmission attempt that the server sends starts a new 'sub session' of acknowledge packets sent between the client and server. Since

[1]The retransmission policy of Microsoft IIS server is different: an IIS server will attempt a retransmission once every ten seconds, and initiate a connection abortion after sixteen unsuccessful retransmission attempts. In the analysis, we use the Apache retransmission properties, since it is more common, but the calculation could easily be modified to fit IIS (and other servers).

Table 1: Apache Server Retransmissions During the Attack

Retr. Attempt	'sub sessions'	$Time_r$ $(sec.)$	$Time_r - Time_{r-1}(sec.)$
1	3	0	00.24
2	4	00.24	00.24
3	5	00.68	00.44
4	6	01.56	00.88
5	7	03.32	01.76
6	8	06.84	03.52
7	9	03.88	07.04
8	10	27.96	14.08
9	11	56.12	28.16
10	12	112.44	56.32
Reset	13	224.88	112.44

the server makes 10 retransmission attempts, by the end of the last time-out there are 13 'sub sessions' of packets traveling back and forth. We mark $T_r = Time_r - Time_{r-1}$ as the current retransmission duration, ρ being the RTT, R the maximum retransmissions before connection abortion (for Apache R=11). The amplification factor that the attacker achieves, with the first retransmission sent roughly at the beginning of the attack, is therefore:

$$Amp_{Two}\left(R, T, \rho\right) = \frac{1}{2} \times \sum_{r=1}^{R} \left(\frac{T_r}{\rho} \times (r+2) \right) \tag{1}$$

2.4. Experiments

In this section we present the results achieved both when we tested the Two-Packets Ack-storm attack in the lab, and when we tested the attack on popular sites in the Internet. We tested the attacks on both HTTP and HTTPS sites, using both Apache and IIS web servers. The results from the experiments can be seen in Table **??**. In the tests we measures the RTT to the site, the number of packets the attack generated and the time passed until the server reset the connection[2].

[2]We focused on Apache servers, because they are the most common. IIS and SSL(live.com) were tested for the attack but comparative research was not done on them.

Table 2: Comparison of 2-Packet Storm Attacks on Different Sites

Site	Total Packets	Dur. (sec) ($Time_R$)	RTT (sec) (ρ)	Server Type	Total Bytes	Ampl.	Ampl. By Analysis
live (SSL)	13,000	229	0.270	IIS	832,000	7,500	5,002
oranim	20,000	120	0.180	IIS	1,280,000	10,000	16,000
yahoo	50,000	225	0.170	Ap.	3,200,000	25,000	25,415.5
facebook	60,000	225	0.160	Ap.	3,840,000	30,000	27,004
google	140,000	225	0.110	Ap.	8,960,000	70,000	39,299
bbc (uk)	190,000	225	0.060	Ap.	12,160,000	95,000	72,000
il.msn (p)	234,000	225	0.030	Ap.	14,976,000	117,000	144,098
bing (p)	320,000	225	0.018	Ap.	20,480,000	160,000	240,163.5
Lab	522,000	225	0.001	Ap.	33,408,000	261,000	4,322,944

In Tab. **??** we can see that while attacking sites running Apache, the time until termination of the attack remains constant - 225 seconds. This value is the connection abortion after maximum failed retransmissions. Once the attack takes place, no data is acknowledged in the session. That causes the server (usually the one sending the data over HTTP sessions) to retransmit the data 10 times, when each time the retransmission timeout doubles. Table **??** shows us the times of the retransmissions in relation to the beginning of the attack, and the termination of the connection occurred after the 10th timeout expired.

The Two Packet Storm attack presents a substantial amplification factor to the data sent by the attacker. The attack, however, has two main limitations:

1. The attack causes a constant number of 'sub sessions' , consuming a limited amount of network resources. If an attacker wishes to attack a high bandwidth target, he would have to use a large number of connections.
2. This attack is time limited, since a server will terminate the connection after reaching the maximum failed retransmissions attempts. After $Time_R$ seconds, the server will abort the connection, terminating the attack in the process. In order to attack a target for a time larger than $Time_R$, he would have to start new attacks to replace the old ones.

In the following sections we present two variations of the Two-packets Ack-

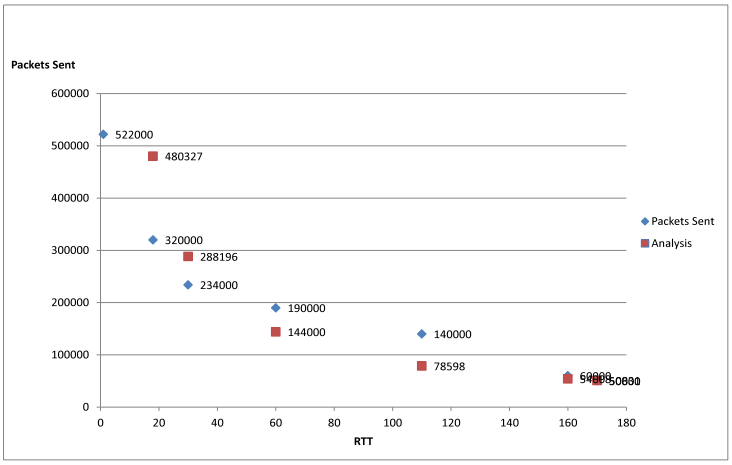

Figure 5: Attack results when attacking various Apache-based sites ($Time_R = 225$), in comparison to analysis. We measured the number of packets sent and compared to the analysis. The graph shows the correlation between the analysis results, and the ones achieved in real attacks. Differences can occur due to packet loss/duplication, retransmission time deviations etc.

storm DoS attack, which address the limitations described above. The N-packets Ack-storm DoS attack enable the attacker to control the bandwidth consumption per session, and increase it to consume the network resources of the client. The Everlasting Ack-storm DoS attack allows the attacker to control the duration of the attack, making it last as long as necessary.

3. The *N*-packet Ack-storm DoS Attack

The N-packet Ack-storm DoS attack enables the attacker to increase the number of packets sent over an attacked session. When using the N-packets Ack-storm DoS attack, the attacker can consume all of the bandwidth available for the session.

In the Two-Packets Ack-storm attack, the attacker sends two packets when triggering the attack. The two packets cause the client and server to send two acknowledgment packets, creating two 'sub sessions' of packets traveling back and forth. In the analysis of the Two-Packets Ack-storm attack, we showed that retransmissions done by the server create additional 'sub session' , increasing the amount of packets sent over the connection. Since the amount of packets sent in a given second is a function of the number of 'sub sessions' currently active, in order to increase the capacity

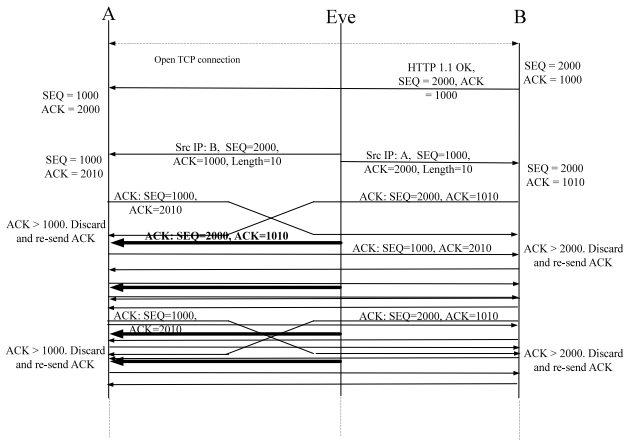

Figure 6: The N-packets Ack-storm DoS attack. The storm packets sent by the attacker are in bold. Every new packet triggers another 'sub session' of acknowledge packets sent back and forth.

of the attack the attacker will create additional 'sub session' artificially by injecting additional acknowledgment packets into the TCP stream, identical to the ones sent back and forth by the client and server.

By Injecting additional ack packets to an attack already in progress, the attacker increases the bandwidth consumed by the attack. For every additional acknowledgment packet the attacker injects into the stream, another 'sub session' is created, and the client and server start passing the packet between them. This method bypasses the latency limitations of the client and server, since more 'sub sessions' simulate a shorter distance between the client and server: for the affective RTT to decrease by 0.5, the attacker doubles the number of 'sub sessions' in the connection.

3.1. Analysis

In order to maximize the effectiveness of the additional 'sub session' he creates, the attacker sends the additional Storm packets at the beginning of the attack. The number of 'sub sessions' at the end of the attack, for 2 packets sent in order to trigger the attack, N additional Storm packets and retransmission generated packets is $2 + N + R$. The amplification achieved would be, for R maximum retransmissions and 64 bytes acknowledgment

packet length:

$$Amp_N\left(N,R,\rho\right)=\frac{1}{2+N}\times\sum_{r=1}^{R}\left(\frac{T_r}{\rho}\times\left(r+2+N\right)\right)$$

The attack minimizes the sessions needed to reach a high bandwidth consumption. Using storm packets, we were able to trigger *485,000* packets sent over a single session (with Yahoo.com), when the Two Packet Storm generated only 50,000.

When reaching the bandwidth limitations of a connection, packet losses will start to occur. Each packet loss will end the 'sub session' of that packet in the stream, while the rest of the 'sub sessions' will continue executing.

Table **??** presents the amplification results when attacking Yahoo.com. Number of packets increases as a function of packets sent, until reaches a point where packets losses start occurring (at about 42 Storm packets sent). Notice that packet losses do not stop the attack, but only limit the maximum bandwidth it can consume. We did not send over 42 packets in order to avoid causing real harm to users in the network, as would result from causing congestion on a live network).

Table 3: The N Packet Storm on Yahoo.com

Storm Packets Sent	Packets In Session	Attack Duration (T)	Bandwidth Consumption (Max.) $\left(\frac{Bytes}{Sec}\right)$	Amplification Factor
2 (Two Packet Storm)	50,000	225	4,517	25,000
6	103,000	225	6,023	17,166
10	153,000	225	7,579	15,300
22	283,000	225	12,047	12,863
42	485,000	225	19,576	11,547

While bandwidth consumption increases as a function of the Storm packets sent, the amplification factor decreases. The attacker has to send an additional packet for every 'sub session' he wants to create, but the number of retransmissions (and the number of 'sub sessions' they create) remains the same. The disadvantage of lower amplification factor is balanced by the lack of need to manage multiple attacks in order to achieve the same amount of bandwidth usage.

The N-packets Ack-storm DoS attack made it possible for the attacker to maximize the bandwidth consumption of the sessions he attacks, but the problem of a limited time attack still forces the attacker to find new sessions every T seconds. The Everlasting Ack Storm attack, described next, allows allow the attacker to preserve attacked connections for long periods of time, increasing the amplification factor as he does so.

4. Everlasting Ack Storm Attack

The main limitation of the two attacks presented so far is the the maximal connection duration $Time_R$ (Since the server dictates $Time_R$, the attack duration will be different per server). The attacker can avoid a connection abortion by artificially sending data over the channel, preventing the client or server from reaching a timeout. This method was tested on apache ad IIS and proven to work on both. The Everlasting Ack Storm data packets should contain at least one byte of data, which will not add to their size because it is below the Ethernet packet size minimum. The packets should be sent at least every $Time_R$ seconds, in order to avoid the timeout. When data is being sent over the connection (however considered retransmission by both the client and server), the connection timeout is not triggered, allowing us to continue the attack until reset by application layer or any other network entity.

4.1. Analysis

The number of initial packets sent by the attacker is the same as the Two Ack Storm attack. The additional packets are sent every $Time_R$ seconds in order to maintain the connection. The amount of total data sent over the channel is composed of the initial $Time_R$ seconds, in which the amount of packets is identical to those of the Two Packet Storm attack, and the following minutes, in which the attack continues with the number of 'sub sessions' increases for every storm packet sent. The amplification achieved when sending $Ever$ storm packets (for R maximum retransmissions of server):

$$Amp_{\infty}\left(R,T,\rho\right)=\frac{1}{Ever}\left(\times\sum_{r=1}^{R}\left(\frac{T_r}{\rho}\times r+2\right)+\sum_{r=R}^{Ever}\left(\frac{Time_R}{\rho}\times\left(r+2\right)\right)\right)$$

In Tab. **??** we show the different abilities the attacker can achieve by using the attacks described above. Notice that the amplification factor when

sending storm packets without data decreases, but the total bandwidth increases as a function of the packets sent. Also notice that when sending storm packets with data, the attacker increases both the duration of the attack (by avoiding timeout), and the bandwidth consumed by it (as it opens another 'sub session')).

Table 4: Amplification Types - Assuming No Losses

Attack type	Data Sent	Attack Intervals	Attack Duration	Time Until Full Bandwidth Potential	Amp Fact
Two	128 B	Once	$Time_R$	Never	Amp_1
N	$128 \times Q$	Once	$Time_R$	Immediate	$Amp_N \leq$
Everl.	$128 \times Ever$	Every $Time_R$	$Time_R \times Ever$	$(Time_R) \times \left(\frac{BW}{64} - (2+R)\right)$	$Amp_\infty >>$

Using Storm packets with data, sent every 1 minute, we maintained an attack on Google.com for over 26 minutes, causing over 10,000,000 packets (over 640,000,000 bytes of data) to be sent before terminating the connection - an amplification factor of 400,000. We terminated the attack to avoid causing read damage.

4.2. Everlasting N Ack Storm Attack

In order to maximize the bandwidth consumed by the attack, the attacker can combine the N-packets Ack-storm DoS attack and the Everlasting Ack-storm DoS attack. The attacker will send amplification packets as fast as he can, combined with Everlasting packets to preserve the attack over time.

In the next section we present analysis and experimental results of an attack scenario on a network. We also measure the amplification factor the attack reaches using a new innovative method that offers a more accurate way to measure amplification, that takes into account the attacked protocol as well as the attacks bandwidth - Effective Amplification .

5. Attack Scenario - Analysis and Experiments

In this section we present an analysis of the attack's performance at different stages. We divide the attack into two stages - the growth phase and the preservation phase. We assume in this section that the attacker uses the most efficient form of the attack - Everlasting Storm for duration combined

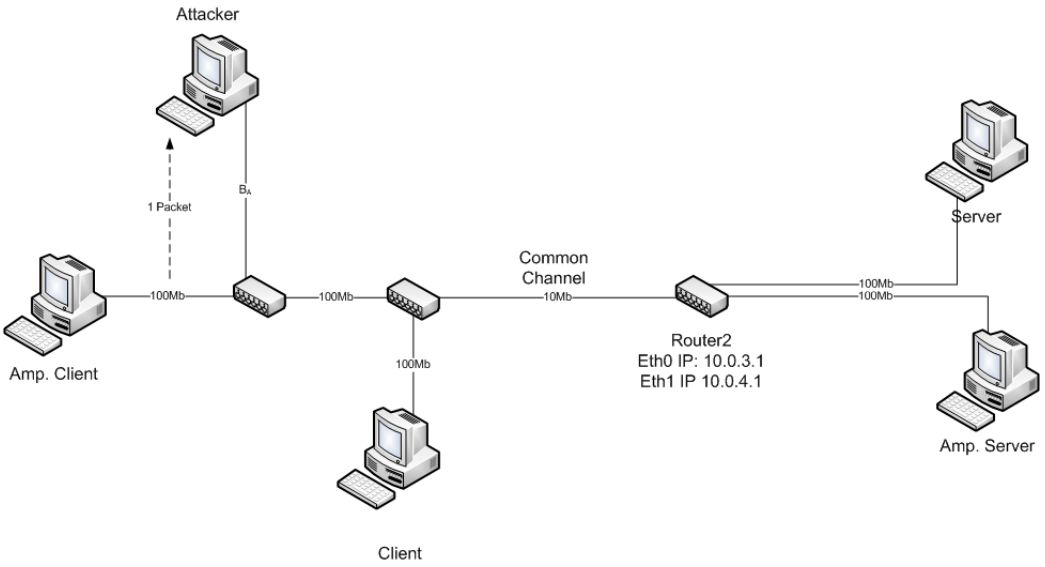

Figure 7: The experiment's network scheme. The innocent computer runs IPERF (a tool that takes up as much bandwidth as possible) to the server. The communication between the innocent and the server takes up almost the entire common channel. Once the attack starts - the innocent's bandwidth capacity decreases until losses in both the innocent's connection and the attack gets the traffic to a steady state.

with N Packet Storm for amplification. The attack's two stages are illustrated at Fig **??**. The attacker sends amplification packets as fast as he can (full bandwidth). We mark the attacker bandwidth as B_A, the attack bandwidth at time t as $B_{Attack}(t)$, the channel bandwidth as C, and the innocent bandwidth on the channel as $B_V(t)$.

5.1. Experiment Setup

When we approached our experiments we wanted to simulate an attack on the victim client's existing TCP connection. We wanted to check how much damage the attack can perform to other TCP connections occupying the same channel.

Our network consists of four nodes - the attacker's computer, the victim's computer, an innocent computer and a file server. The network will also include 3 routers to separate the different networks, creating a common channel between the different clients and the server. Fig **??** demonstrates the network's topography.

5.2. Analysis Assumptions

In order to simplify the analysis we assume in our analysis a single victim TCP session. The only difference between single and multiple innocent sessions is the affect packet loss has on the overall innocent bandwidth.

In the case of multiple innocent TCP sessions, each packet loss will cause only one of the sessions to lose throughput, which means that the larger the number of sessions, the smaller the drop in total innocent bandwidth for each innocent packet dropped. This means that the overall innocent will drop 'smoother' as the number of innocent TCP sessions increases.

5.3. Analysis and Experiments: The Growth Stage

When the attack starts in an uncongested channel, we assume that no losses occur. Therefore, every packet injected by the attacker creates a 'sub session' to be sent back and forth, consuming bandwidth from the channel. The attack's overall bandwidth will increase linearly with the rate of the attacker's bandwidth, as presented in Eq. **??**.

$$B_{Attack}(t) = B_A \times t \tag{2}$$

The linear growth will continue until packet losses start occurring on the channel. As long as the attackers bandwidth is higher than the loss rate, the attack's overall bandwidth will continue to grow. When the channel will become too congested, the loss rate will match the attacker's bandwidth, which will stop the attack's growth and stabilize the attack's overall bandwidth.

Figure **??** shows the analysis vs. experimental results. We see a strong correlation between the predicted results and the actual results achieved in the experiment environment.

5.4. Analysis and Experiments: The Preservation Stage

When attacking a congested channel, each packet passing in channel will have a chance of being dropped due to congestion. When an attack packet is dropped, this will terminate a 'sub session' , decreasing the attack's bandwidth by $\frac{PacketSize}{RTT}$. In order to compensate for the losses, the attacker should send a packet for every attack packet dropped. For example, if we assume a loss percentage of 5%, the attcker should replace 5% of the total attack packets in order to keep the channel congested. When the attack consumes 100% of a 10Mb channel, the attacker will have to re-send $5\% \times 10Mb = 500Kb$ of packets just in order to keep the channel congested.

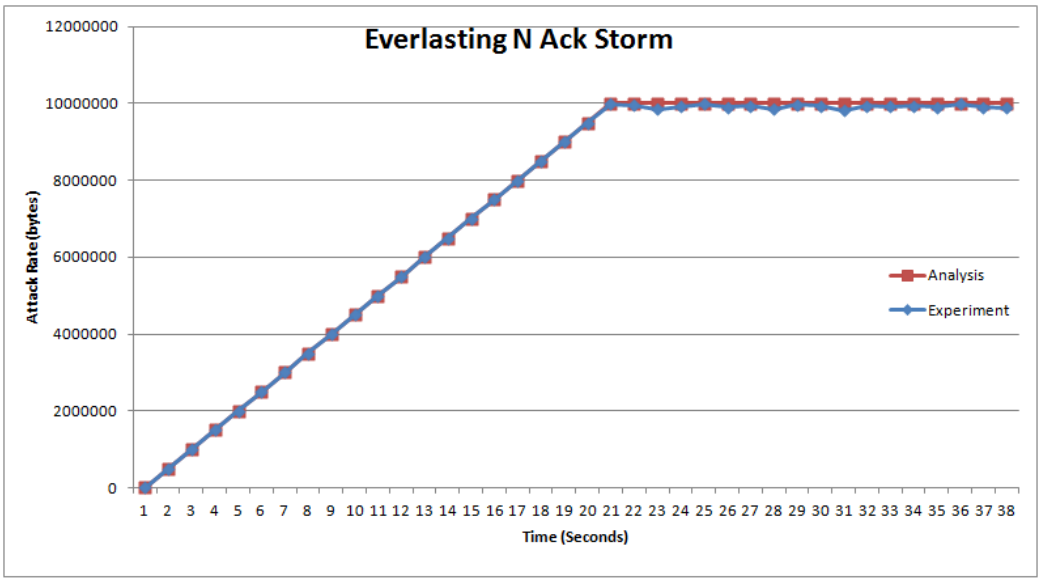

Figure 8: The Growth Stage analysis vs. experimental results on a 10Mb channel.

If the attack is held on an already busy channel, the attacker will be able to reach higher loss rates. If the channel is relatively unused - the attacker will have grater difficulty making significant impact.

Although it effects the amount of bandwidth decreased by each 'sub session' terminated, RTT does not affect the maximum loss percentage since each packet sent by the attacker will start a new 'sub session' that will consume $\frac{packetsize}{RTT}$ bytes - the same amount that the dropped packet subtracted from the overall attack bandwidth.

We present an analysis of how a TCP connection is effected by an ongoing attack. In order to do so, we must first understand the performance of TCP under congestion.

5.4.1. TCP Under Congestion

The TCP protocol is considered sensitive to packet loss. When a channel is congested and packet losses, even of 1-2% of packets, start to occur - a TCP connection's throughput will decrease significantly. For Example, according to TCP's theoretical maximum throughput (with MSS=1460 and RTT=80ms) when loss percentage is 10^{-06}% is 1392.36 Mbit/sec. If the channel has 1% loss - TCP's maximum throughput drops to 1.39 Mbit/sec.

TCP's throughput is approximated by the formula described in [**?**].

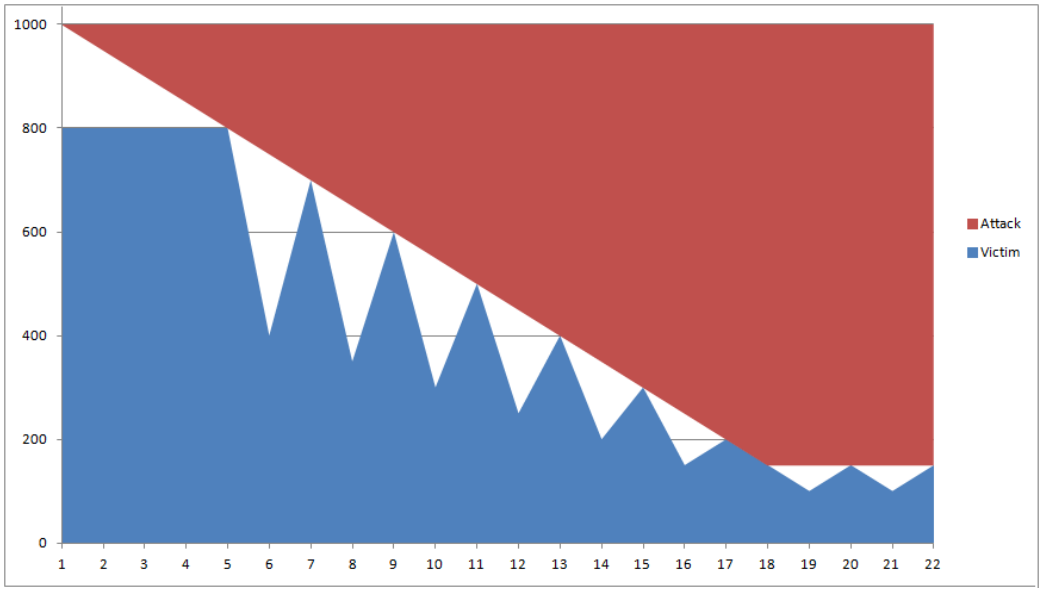

Figure 9: When the attack starts (growing stage) the attack's bandwidth will consume larger parts of the channel over time. A TCP connection that shares the channel will encounter packet losses whenever it's throughput will be high enough to fill the channel. Eventually the attack will consume nos of the channel's bandwidth, and the TCP connection's throughput will decrease to a minimum.

While [**?**] presents a more accurate formula in loss rates higher than 2%, we focus on relatively weak attackers that can preserve lower loss rates. The formula for TCP's throughput upper bound is

$$Rate(p, RTT, MSS) = \frac{MSS \times \left(\frac{1-p}{p} + w\left(p\right) + \frac{Q(p,w(p))}{1-p}\right)}{RTT \times \ \left(w\left(p\right)\right) + \frac{Q(p,w(p)) \times G(p) \times T_0}{1-p}} \tag{3}$$

Where W, Q anf G are defined in **??**, **??**, **??** respectively:

$$w\left(p\right) = \frac{2}{3} \times \left(1 + \sqrt{3 \times \sqrt{1-p}p + 1}\right) \tag{4}$$

$$Q\left(p, w\right) = \min\left(1, \frac{\left(1-\left(1-p\right)^3\right) \times \left(1-\left(1+p\right)^3\right) \times \left(1-\left(1-p\right)^{w-3}\right)}{1-\left(1-p\right)^w}\right) \tag{5}$$

$$G\left(p\right) = 1 + p + 2 \times p^2 + 4 \times p^3 + 8 \times p^4 + 16 \times p^5 + 32 \times p^6 \tag{6}$$

Where p is the loss rate, MSS is the maximum segment size (typically 1470) and RTT is the round trip time between client and server. Since during an attack MSS and RTT are constants, in Fig. **??** we see that linear increase in loss rate creates a exponential decrease in TCP throughput.

Figure **??** demonstrates the expected bandwidth in the preservation stage vs. the experiment results. We see that the experiment results offer a lower TCP rate then the analysis. This can be caused by a router queuing issue: since the attack packets are without data - they are smaller then the data packets sent in a standard TCP session. therefore, they have a better change to enter the router's queue then the full sized TCP packets.

We can see that the attack causes other TCP connections on the common channel to lose significant amount of throughoput. We want to better understand the relation between the attacker's bandwidth and the amount of throughput lost while under attack.

6. Re-Inventing Amplification

When discussing the attack we presented an amplification model that works well when the network is not congested. When packet losses start to occur, the attacker needs to send additional packets in order to compensate the dropped packets, causing the attacker's amplification to decrease.

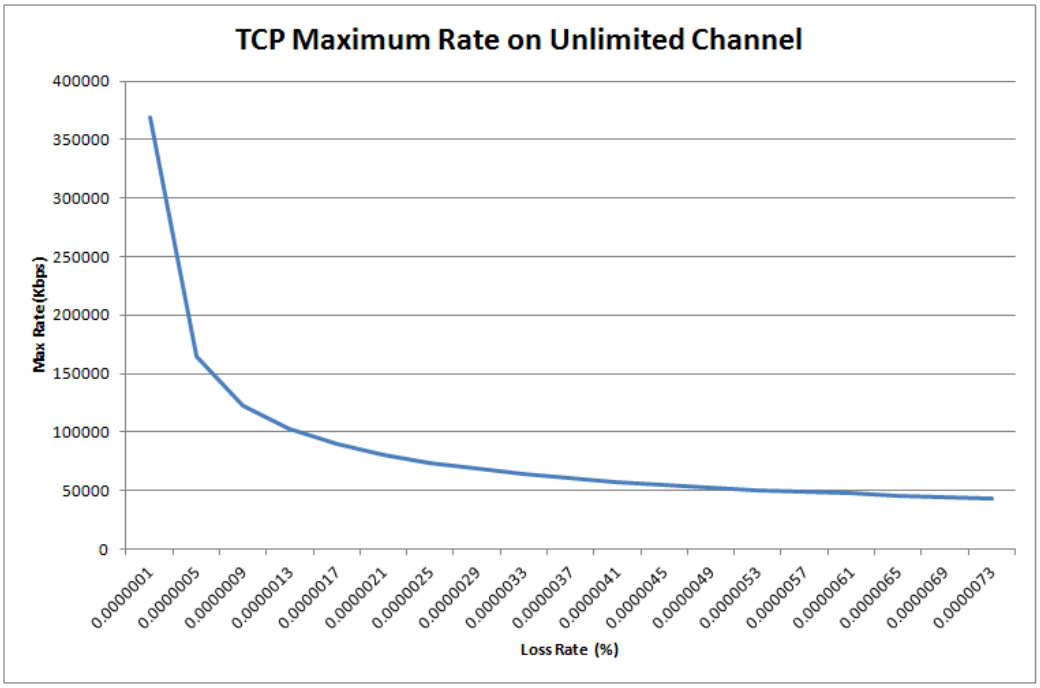

Figure 10: Maximum TCP Throughput under losses. Calculated with MSS=1460 bytes and RTT=0.1 sec.

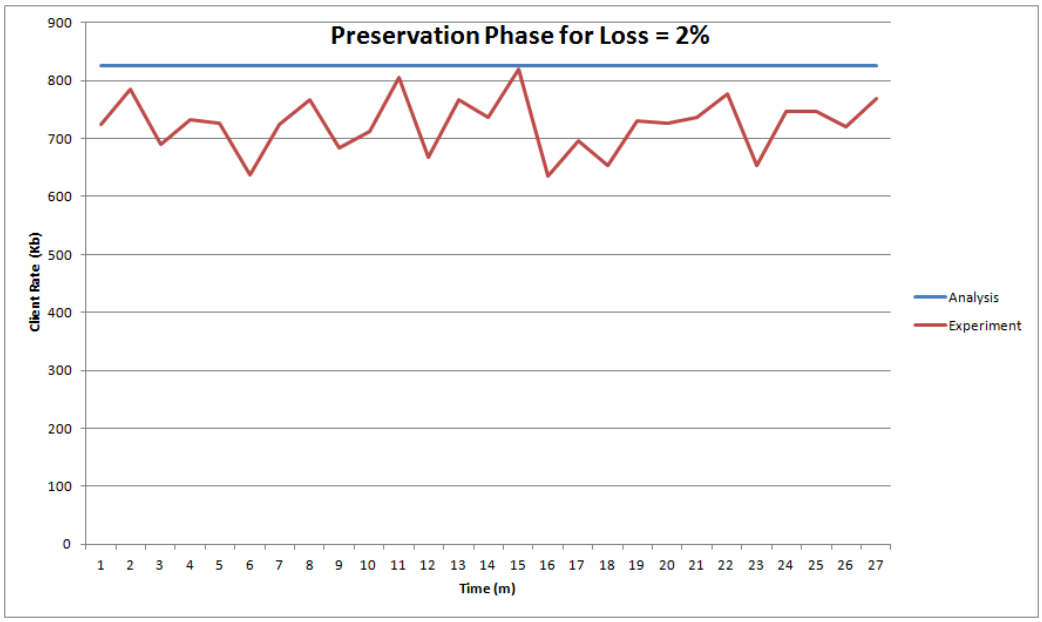

Figure 11: Experimental results for preservation stage we used an attacker with 20Kb bandwidth on a channel of 10Mb.

Each attack packet dropped will stop a 'sub session' . In bandwidth terms, for each packet dropped the attack's consumed bandwidth will decrease by $\frac{PacketSize}{RTT}$ bytes. If the attacker will not inject a packet to replace the dropped one - the attack's bandwidth will slowly decrease until the network reaches a steady state (almost no losses).

For the attack to maintain the same bandwidth volume, the attacker must replace the dropped packets with new ones. When the attacker has to dedicate bandwidth to preserve the attack's volume, the amplification factor of the attack decreases. When we look at how much the attacker needs to invest in preserving the attack, we see that the higher the loss rate, the more the attacker needs to retransmit packets.

For example, if we assume a loss rate of 5%, the attacker has to regenerate 5% of the attack bandwidth - send $\frac{5}{100} \times \frac{C}{RTT}$ bytes to the channel. Notice that the attacker does not need to re-send every lost packet - but only one packet for every 'sub session' lost - in order to maintain the attack under losses.

When attacking TCP based connections, the amplification factor discussed previously does not incorporate TCP's high sensitivity to packet

losses. Since TCP's throughput can decrease significantly with as low as 1-2 percent packet loss, we can modify the amplification calculation to include the potential bandwidth the TCP connection loses when under an attack causing losses.

6.1. Why a new amplification model?

When analyzing an attack's damage potential, we must consider the attacked protocol's behavior under congestion in order to reflect the real damage our attack can cause. When measuring Effective Amplification we shift the focus from the amount of bandwidth generated by the attacker, to the amount of bandwidth taken away from the victim(s). When we measure the bandwidth lost by the victim(s) we get a much more accurate measurement of the damage an attack can cause.

Effective Amplification takes into account the individual charecaristics (and weaknesses) of the attacked protocol, enabling us to more accurately asses the damage caused to a victim while under attack. While Effective Amplification and regular amplification will give the same results when attacking stateless protocols, such as UDP or RTP, the Effective Amplification can give a far better estimate to an attack's effectiveness when measuring against stateful protocols such as TCP, because of its sensitivity to packet loss.

6.2. Effective Amplification

When showing the amplification the attack causes we want to reflect the damage congesting the channel can do to TCP connections. In order to do so, we define the *Effective Amplification* of an attack as the ratio between the bandwidth dedicated by the attacker, to the difference between the full potential of a connection and the bandwidth achieved when under attack. For example, when an attacker transmits attack packets at 0.01Mbps, and the TCP's connection throughput drops from 10Mbps to 0.1Mbps, the Effective Amplification of the attack will be $\frac{10-0.1}{0.01} = 990$. In general, the Effective Amplification will be calculated by equation (**??**):

$$EffectiveAmplification_{B_A,C,B_V} = \frac{C - B_V}{B_A} \tag{7}$$

6.3. Amplification in Steady State

We want to find the relation between $B_A(t)$ to the maximum amplification achieved by the attack. When the attack is in place we assume the it takes up most of the channel bandwidth. Since in the attack steady state the attacker's bandwidth is consumed with restoring lost packets, we can mark $B_A(t) = \frac{p \times AttackBW}{RTT} < \frac{p \times C}{RTT}$. We isolate p and get $p = \frac{B_A(t) \times RTT}{AttackBW} < \frac{B_A(t) \times RTT}{C}$.

If we replace p in the TCP rate formula we can get an upper bound to the innocent bandwidth in the channel. Now we can calculate the Effective Amplification according to the formula above and get $\frac{10-0.1}{0.01} = 990$. In general, the Effective Amplification will be calculated like so

$$\text{Effective Amplification}_{B_A(t),C,B_V(t)} = \frac{C - Rate(p, MSS, RTT)}{B_A(t)}$$

6.3.1. The Ideal Attacker

We want to find the 'ideal' attacker. An ideal attacker is considered one that will provide the best ratio between the amount of damage caused, and the amount of bandwidth invested. As seen earlier in the section, an attacker's ability to cause losses increases linearly in relation to his bandwidth.

Figure **??** shows the Effective Amplification of an attacker using attacking a 10Mb channel. We can see that the maximum is at 0.00033% loss rate - which requires only 3.3Kbps connection from our attacker. The Effective Amplification in that case is 1321 times the potential.

Since the Effective Amplification depends on the maximum channel bandwidth, the amplification calculation must be done per channel bandwidth. The Effective Amplification factor can reach over 100,000 under large enough channels, without the need for a stronger attacker. The Ack Storm attacks Effective Amplification is so strong because its accumulation stage can take up an infinite channel with sufficient amount of time, causing congestion regardless of the channel overall size.

7. Network Challenges

In this section we discuss the difficulties a WitM attacker can encounter while trying to interfere in an active session, and offer ways to bypass such difficulties.

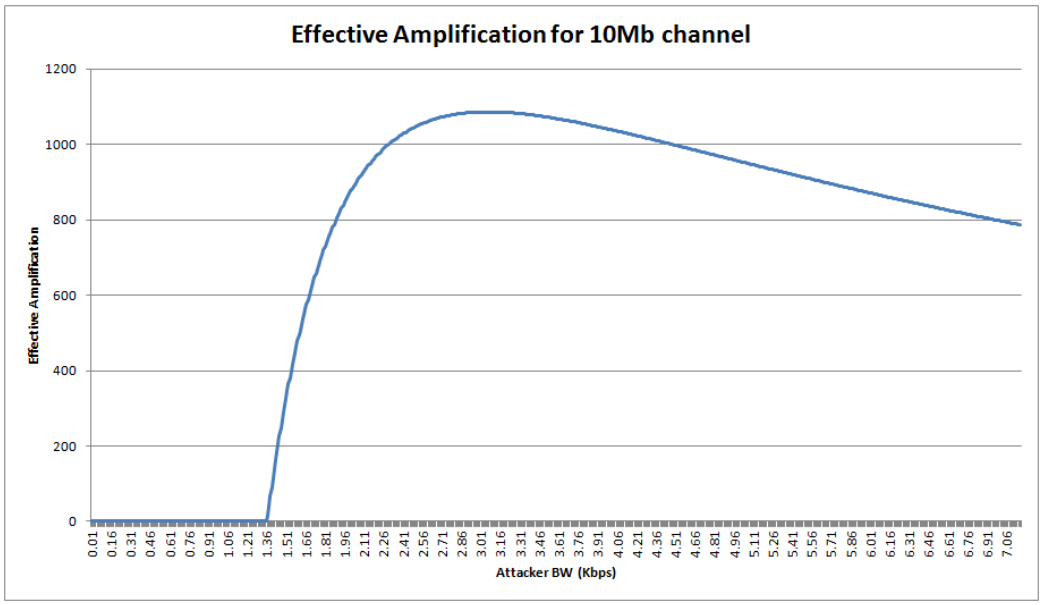

Figure 12: Effective Amplification on 10Mb channel. We can see that the maximum amplification is at 0.00033% loss rate - which requires only 3.3Kbps bandwidth from the attacker.

7.1. Injecting Packets Into an Active Session

For the attack to work, the attacker must succeed in injecting a packet into in active TCP session. To do so, the spoofed packets sent must be received by both the client and server, falling inside the TCP congestion window of both. When the connection between the client and server is idle, sequence numbers remain the same from the point the attacker received the packet, to the point where the client and server receive the spoofed packets. That allows the attacker to know the exact sequence number of both the client and server, guaranteeing that his packet will be received.

When the connection is an HTTP request, a file transfer or some other type of TCP based one sided traffic, once the request is made, the client's sequence number remains the same, while the server's advances as the data is being sent. When data is sent by both sides consistently, both the client and server's sequence numbers advance and injecting a packet is more difficult, especially is the attacker's latency is high.

When transferring large files over TCP, the TCP congestion window becomes large. When the window is large, it's easier for the attacker to send a

packet inside the window. In our experiments, we found that when injecting into an ongoing HTTP request, the attacker needed 10 packets, with each containing different server sequence number for the injection to succeed.

If the attacker wishes to improve the chances to inject a packet to a HTTP session, it can wait for the HTTP data is done transferring. The most common web browsers, such as Internet Explorer and Firefox, keep TCP sessions open after the web page is finished downloading, allowing the attacker to sent the packet when the connection is idle.

7.2. Bypassing NAT

Our attack scenario uses wi-fi networks in order to eavesdrop packets from various locations. But wireless networks present some difficulties the attacker must overcome in order to conduct a successful attack.

The attacker can pick up packets sent over a remote wi-fi network using very inexpensive equipment; however, much better equipment is necessary to allow the attacker to inject packets back into the network. Moreover, if the attacker wishes to eavesdrop on multiple networks simultaneously, the chance of being able to inject packets to all of them is remote. To overcome this, the attacker must find a way to inject a packet into a wireless network from the Internet. This requires two abilities: sending spoofed packets to the Internet, and sending packets to nodes inside the network's LAN from outside.

The ability to send spoofed packets to the Internet should be filtered by the ISP using ingress filtering [**?**]. Unfortunately, most ISPs today do not filter outgoing or ingoing traffic, leaving the network completely exposed to spoofing threats.

The attacker must also be able to send a packet to a specific computer inside a network with NAT (as most internal networks implement today), when the attacker itself resides outside the network (also called NAT hole-punching [**?**]). To do that, the attacker has to know the external IP and port the NAT allocates for the internal computer when it's accessing the Internet. Since small/medium networks have usually one to few external IP addresses, the attacker can preform initial mapping of the wi-fi networks in his area and map external IP addresses to networks. Since the connections of public networks is usually long lasted and consecutive, the IP is likely to remain the same for long periods of time. To discover the port, the attacker uses the packet it picked up from inside the network, from which it can learn the internal port the node uses. The attacker then relies on the port-preserving

quality that over 70% of networks in the world today implement, according to [**?**]. Networks implementing port forwarding keep, whenever possible, the internal port number when mapping outgoing packets to the Internet. Since web browsers use random ports each time they initiate a connection to a web server, chances are that the port will be available. Using the IP and port the attacker can address packets from the Internet to the client inside the NAT.

8. Defenses

The attack described above is based on injecting packets into an existing TCP session. While this ability in non trivial, once achieved it enables an attacker to cause great damage to both client and server. In this section we describe protective measures which can be applied by either the client or the server, in order to block or reduce the attack's damage.

8.1. Fixing TCP Implementation/Standard

The TCP standard describes the desired behavior in every state in the connection. As such, TCP implementations try to stick to the standard completely, avoiding diversions as much as possible. However, the direct approach in order to prevent Ack storm attacks, is to preform a small change in the TCP implementation. When receiving a packet containing Ack field higher then the receiver's sequence number - the packet must be discarded, and no response should be sent. That will guarantee that upon such an attack, no traffic will be generated. A more strict approach is to reset the connection, but such an action is not recommended, since it opens a possibility of DoS attack based on sending the client a packet with the maximum ack number available, causing him to terminate the connection.

8.2. Use encrypted wireless network and/or IPsec-protected connections

Eavesdropping a non encrypted wireless network is fairly simple, even from a great distance. Non encrypted networks are still common, especially in public places such as coffee houses, parks etc. If Bobnet was an encrypted network, the attacker would be unable to eavesdrop on it, making its attack impossible. Similarly, the attacks would fail if communication is done over IPsec (encrypted).

8.3. Filtering TCP duplicate Acks

Filtering TCP duplicate acks can neutralize the Two-Packets Ack-storm attack's ability to cause large amount of traffic (even if done at the cost of terminating the connection). The filtering can occur in any firewall along the packets route.

We offer a way to filter the attack according to the packets' length and quantity. When over 1000 packets are sent in less than 1 minute without any content (can be checked using TCP sequence numbers), the connection will be assumed as bad and dropped. The timeframe is meant to assure that the packets are not sent as keepalive, which will generate up to 20 packets per minute. The length limitation is meant so we will not mistake a large file transfer as an attack, as in the above the sequence numbers of at least one side in the connetion will increase.

Using a traffic filter will ensure that such an attack will not be possible, and with minimal filtering costs.

9. Conclusion and Future Work

In the article we presented the the WitM attacker model. We showed the Storm attacks, which can cause DoS to network infrastructure, as well as individual web sites and services. We showed both in analysis and in experiment results the high amplification factors the Storm attacks achieve.

We also presented an analysis and experimental results of a full attack scenario showing the degradation in TCP throughput in an attacked channel.

In addition, we showed an innovative way to measure an attack's amplification factor - a method that considers the attacked protocol individuals weaknesses as it calculates the amplification not according the attack's bandwidth, but according to the actual damage that the attack causes to a congested channel.

The Storm attacks are just the beginning of a wide rage of attacks possible for the WitM attacker. Zombie-based DoS attacks can be accomplished by a WitM without the need to control the node itself, but only by using its connection details. Injection attacks are also possible to the WitM attacker, however the ability to inject the data in the correct timing requires further work, due to the high latency that the attacker has. Finding additional DoS attacks and uses for a WitM attacker will add additional impact for the WitM attacker model.

Acknowledgments

Many thanks to Charlie Kaufman, Amit Klien and Ben Laurie for their important feedback and encouragement. Amit also introduced us to the earlier work discussing Ack storms (as an undesirable side-effect of TCP hijacking attacks), e.g.[**?**].