Client-Controlled Slow TCP and Denial of Service

Songlin Cai, Yong Liu, Weibo Gong

***Abstract*— Denial of Service attacks are becoming an increasing threat to our information infrastructure. By exploiting vulnerability in existing protocols and infrastructures, malicious attackers consume resources in networks and servers to block or degrade the service to legitimate users. TCP is the dominant network transport protocol. It relies on the participating hosts' cooperation to make data transmission successful. This kind of trust has been exploited in some DoS attacks, such as SYN-flooding attack. In this paper, we investigate how a TCP client can extend the duration of its connection with a server only by setting the pace of sending back acknowledgement packets. Our study shows that the duration of a TCP connection could be extended tens of times without incurring timeout retransmission. This mechanism can potentially be used by attackers to launch DoS attacks by generating simultaneous prolonged TCP connections with the victim servers. Unlike SYN-flooding attacks, the low rate property of slow TCP connections makes the detection of this kind of attack difficult, which calls for a further study on this issue.**

I. Introduction

Denial of service attacks aim at consuming targets' resources, such as network bandwidth, server CPU time, TCP connections, etc., to prevent or degrade services to legitimate users. One way to occupy resource is to generate requests at very high rate [4][7][16]. For example, SYN-flooding attack relies on large volume of requesting packets to overwhelming the target. The request rate necessary to overwhelm a victim server increases with the server's capacity. It was shown in [3] that 500 SYN packets per second would overwhelm an unprotected server, while $14,000$ packets per second would be needed to disable a protected server. An attack with large volume of traffic is significant and easy to be detected by routers and traffic monitors. Once an attack is detected, many effective mechanisms [15][14][12] could be used to trace the source of attacking. Moreover, a feature called SYN-Cookies is implemented in Linux and FreeBSD to mitigate SYN-Flood effect [17].

Another way to occupy resource is to generate long-duration connections with servers. While a client can request large files available on a web server to establish long connections, the high data volume in transferring large files is again easy to be detected. In this paper, we study how a client can stretch its connection by slowing down TCP data transmission rate. By doing that, the client can occupy a large number of TCP connections available at the server with low traffic rate, which makes the detection difficult. This is in contrary to most research on TCP which has been focused on how to increase its throughput and decrease the download latency [11]. The purpose of our work is to point out the vulnerabilities in current TCP protocols and hopefully draw more attention on the study of TCP in the context of security.

S. Cai and W. Gong are with Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01003 (scai, gong@ecs.umass.edu)

Y. Liu is with the Department of Computer Science, University of Massachusetts, Amherst, MA, 01003 (yongliu@cs.umass.edu)

TCP is the dominant network transport protocol, and essentially a close-loop control protocol. Its operation largely depends on the collaboration and trust between the server and the client. The server relies on the client's feedback, in the form of acknowledgement, to adjust its sending rate. This provides the possibility for the client to control the server's sending rate by simply manipulating the pace of sending back acknowledgement packets.

We first demonstrate the feasibility of extending TCP connection duration by increasing its round trip time. This is achieved by introducing artificial delays through a network emulator between the client and the server. Then, we present a detailed study on how a client can deliberately delay acknowledgement packets to throttle down its connection's sending rate and extend the duration of the connection. We investigate to what extent a TCP connection can be stretched so that it still appears to the server a "normal" connection. We focus on the case where the client greedily stretches its connection, and in the same time tries to avoid TCP timeout at the server. While TCP timeout helps to prolong connections, frequent timeouts could alert the server's intrusion detection system. Therefore the server's TCP timeout value serves as an upper bound of the delay the client can put on acknowledgement packets. Fortunately (unfortunately), in order to deal with heterogeneity and randomness in networks, TCP reserves a large margin in its retransmission timeout value. We will present client-end algorithm to estimate and largely stretch the retransmission timeout value at the server end under different network situations. Our NS-2 simulations show that the durations of TCP connections can be stretched tens of times without triggering timeout retransmission. We also show through experiments that legitimate users would suffer if multiple slow TCP connections are initiated toward the server.

Slow TCP connections are "well-suited" in *low rate* DoS attack. To guarantee performance, web servers set up a limit on the number of concurrent connections. An attacker, as a TCP client, launches a large number of low rate TCP connections, either from one machine or a cluster of comprised machines, to use up connections available at the server. Unlike traditional high rate DoS attack, the low traffic rate property of this type of attacks makes them more difficult for routers and traffic monitors to detect. Although we present several possible solutions, we believe the trust built in TCP protocol is the source of this kind of attack. We believe a further study on how to construct a more secure network infrastructure and protocols is needed to finally solve this problem. This kind of study would help understand the vulnerability of the system and the possible attacking approaches, then preventive mechanism could be equipped before the vulnerability is exploited or detection methods could be deployed to detect the attack.

The rest of this paper is organized as follows. Section II demonstrates the feasibility of extending TCP duration by introducing additional delay in round trip time. In Section III, we focus on studying how to extend the TCP connection by controlling acknowledgement packets without incurring timeout retransmission. Section IV shows that a legitimate user could suffer from those slow TCP connections. In Section V we study the feasibility of slow TCP in denial of service attack and its possible solutions. Section VI concludes this paper.

II. Extended Round Trip Time and its Effect on Duration of Connection

TCP protocol provides reliable data transmission to application layer. The resources involved in data transmission include those in transport layer and application layer. In this section, we first describe the interaction between transport layer and application layer during data transmission. We emphasize the resources in application layer should be protected as well as those in transport layer. Because round trip time consists of different components, a delay component introduced between client and server to increase round trip time could extend the serving time at application layer as well as the duration of TCP connection at transport layer. Our experiment shows the resource at the server could be occupied longer if data transmission is extended.

A. Data Transmission: Resource Consumption at Transport and Application Layer

A TCP connection normally has three stages: connection setup, data transmission and connection tear-down. A typical scenario when a client requests data/service from a server consists of: (1) the client sends SYN request to the server, and a three-way handshake makes the connection established; (2) After the connection is established, the server passes data from application layer to transport layer, and then the data reaches the client; (3) After the server sends all the data to the client, it tears down the connection by sending FIN packet to the client. And a couple of packet exchanges finish the connection.

Each stage of TCP connection involves resource consumption. So does data transmission. After three-way handshake, the server starts to transmit data from application layer to transport layer. From this time till the connection is torn down, the operating system at the server has to allocate resource for application layer. This resource has a limitation. For example, the default value for maximum clients served simultaneously in Apache server is 150 [13], i.e, if more than 150 requests come for web content at the same time, some of them have to wait until those in serving finish.

If the data transmission is extended, the related resource at the server, in both transport and application layers, would be occupied longer. Before we proceed on, we need to answer this question first: is the server tolerant of extended data transmission to consume its resource longer? The following experiment shows that a delay component between the client and the server could result in longer TCP connection and serving time in web server.

B. Extended Round Trip Time in TCP Connection

In TCP protocol, the receiver is required to send acknowledgement packets to the sender, to ensure a reliable data transmission. Round trip time records time interval from when the data packet is sent to when an acknowledgement packet is received. Suppose Host A sends a packet to Host B, and receives an acknowledgement packet from Host B, the round trip time consists of three parts: the propagation time $rtt_{A\rightarrow B}$ from Host A to Host B, the processing time rtt_B in Host B, the propagation time $rtt_{B\rightarrow A}$ from Host B to Host A. Therefore, the round trip time rtt can be expressed as: $rtt = rtt_{A\rightarrow B} + rtt_B + rtt_{B\rightarrow A}$. No mechanism in TCP protocol is used to tell how round trip time is divided by these three components. Therefore, for Host A, increasing processing time rtt_B at Host B has the same effect as increasing propagation time between two hosts: both ways increase round trip time.

C. Experimental Study: Round Trip Time and TCP Connection Duration

The following experiment is to clear the doubt on whether real TCP implementation and the applications built on it would allow long round trip time, and help understand the effect of extended round trip time on the duration of TCP connection and serving time of the application.

The experiment setting in Figure 1 consists of one server and two clients. All computers run Linux operating system. The server is installed with Apache software [13] to provide web service. A software called NIST NET [8] is installed in $client1$. NIST NET is to introduce delay from $server$ to $client1$. In figure 1, each packet from $server$ to $client1$ would first feed into NIST NET and then NIST NET passes packet to $client1$ after the specific delay, while the packets from $client1$ to $server$ would directly propagate to $server$. The delay introduced by NIST NET increases round trip time between $server$ and $client1$.

In our experiment, we configure the delay introduced by NIST NET from 0 to 12sec, with 1sec as one step. For each delay, three files at the server whose sizes are 1KB, 100kB, 200KB respectively are retrieved. Each case is repeated 20 times to have the average of the result. We record the duration of TCP connection at $client1$ and get the serving

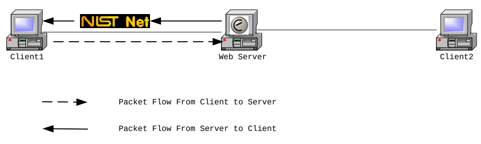

NIST Net
Client1
Web Server
Client2
Packet Flow From Client to Server
Packet Flow From Server to Client

Fig. 1. Experiment: RTT and TCP Duration

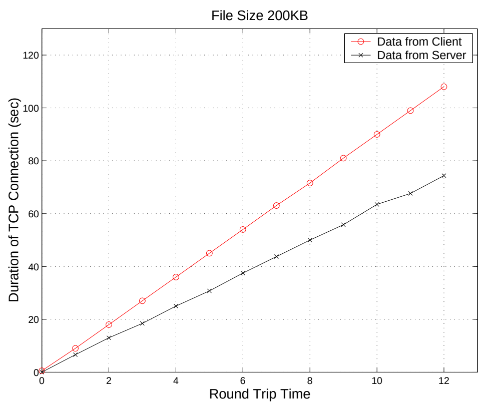


Fig. 2. Extended Round Trip Time and its Effect on the Duration of TCP Connection (File Size 200KB)

time at the server from the web log file. Figure 2 shows the experiment result for the case with file size of 200KB[1].

In Figure 2, the duration of TCP connection increases as round trip time increases. So does the serving time at the server. Note a gap exists between the duration at the client end and the serving time at the server end. This is mostly due to the first several SYN packets and the last several FIN packets, since the serving time would only record the time the web server takes over and processes the data.

III. Data Transmission Extended by Manipulating Acknowledgement Packet

We present the objective and assumption of our study, and use TCP implementation in BSD UNIX [19] as an example to show how the client manipulates acknowledgement packets to extend TCP connection without incurring retransmission timeout at the server. Finally, we have a discussion on the variants of TCP implementation.

A. Objective and Assumption

Our study on extension of TCP connection is mainly from attacker's point of view. The objective is to extend the duration of TCP connection, and at the same, to make the TCP connection look normal to the server. Therefore, we bear several points in our mind during the study: (1) We want each TCP connection to experience its previously described three stages. (2) The round trip time should be adjusted gracefully. Though a couple of timeout retransmissions could increase round trip time, an attack initiated by this mechanism could alert intrusion detection system.

[1]We refer the readers to [2] for the other two cases.

Therefore, the client would adjust the pace of acknowledgement packets within the range of retransmission timeout at the server. In a word, the objective of this section is to study how long a TCP connection could be extended by the client controlling acknowledgement packet transmissions, given no timeout retransmission occurs at the server.

To simplify our study, we assume:

• The client acknowledges each packet from the server. No loss happenes to the acknowledgement packets.

• The propagation delay from the client to the server is the same as that from the server to the client.

•The TCP behavior is described in terms of "rounds". A round starts when the server begins the transmission of a window of packets and ends when the it receives an acknowledgement for one or more of these packets.

B. TCP Operations

In our study, the client will introduce an artificial delay to increase round trip time without incurring timeout retransmission in the server, so we give a brief description of the calculation of round trip time and retransmission timeout, the operation of RTT timer and RTO timer.

1) Round Trip Time, Retransmission Timeout: To ensure smooth operation of data transmission, TCP relies on the following three parameters: smoothed round trip time $srtt$, smoothed mean deviation $rttvar$ of $srtt$ and retransmission timeout RTO. The calculation of $srtt$ and RTO could be summarized by the following equations:

$$\begin{aligned} srtt &\leftarrow srtt + \frac{1}{4} \times (rtt - srtt) \\ rttvar &\leftarrow rttvar + \frac{1}{8}(|rtt - srtt| - rttvar) \\ RTO &= srtt + 4 \times rttvar \end{aligned} \tag{1}$$

where rtt is the measured round trip time.

The initial values of $srtt$, $rttvar$ and RTO are 0s, 3s, 6s. $srtt$, $rttvar$ and thus RTO are updated whenever the ACK packet for a timed data packet arrives.

2) Operation of RTT and RTO Timer: In BSD version [19], RTT and RTO timers work as follows[2]:

• When TCP transmits a data packet, if the RTT timer and RTO timer are not used, this data packet will be timed, otherwise, the data packet would not be timed. In Figure 3, at time t_1, one data packet is sent from the server. Since there is no data packet being timed at time t_1, RTT timer and RTO timer both start to time this packet.

• Since only one RTT timer and one RTO timer are available for a TCP connection, only one packet could be timed each time. In Figure 3, in addition to the first packet sent at time t_1, the other data packets sent between t_1 and $t_1^{'}$ are not measured for round trip time because RTT timer has been used to time the data packet sent at t_1.

• If there are data packets sent but not acknowledged, RTO timer will be set to time one data packet. In Figure 3,

[2]Since the case of timeout retransmission is not studied in this paper, we refer readers to [19] for details.

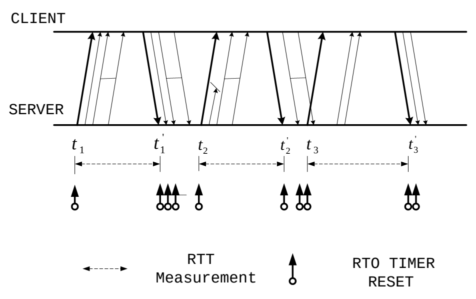


Fig. 3. Operation of RTT and RTO timer

RTO timer is first used to time data packet at time t_1. After the acknowledgement packet is received at $t_1^{'}$, RTO timer will reset to time those data packets not yet acknowledged.

• When TCP detects duplicate ACKs and a fast retransmission is initiated, RTT and RTO timers are reset to time this retransmitted data packet, even they are being used to time other data packets. In figure 3, between t_2 and $t_2^{'}$, there is a data packet loss. At time t_3, the server receives duplicate ACKs, and retransmits the lost packet. The RTT and RTO timers are reset to time this retransmitted packet.

C. Round Trip Time Extended through Acknowledgement Packet Controlling

From server point of view, the round trip time is the interval between when the server sends the data packet and when the server gets the ACK packet. It consists of three parts: the propagation time from the server to the client, the processing time at the client, and the propagation time from the client to the server. The calculation of round trip time depends on acknowledgement packets from the client. No mechanism in TCP protocol could identify each component of round trip time. The client could control the pace of acknowledgement packet, in particular, hold the acknowledgement packet to increase the round trip time.

Our study is focused on how the client could introduce artificial delay to increase round trip time gracefully and extend the TCP connection without incurring timeout retransmission. This is possible due to the following factors: (1) The client could measure the propagation delay from the time stamp of data packets; (2) The client knows whether data packet gets lost or not earlier than the server; (3) The client knows TCP algorithms in updating $srtt$, $rttvar$ and RTO and can predict those values at server end.

1) TCP with Extended Round Trip Time and No Packet Loss: To extend the duration of TCP connection without incurring timeout retransmission, the client needs to know how much artificial delay should be introduced for each acknowledgement packet. Before making the decision, the client needs to know the value of RTO at the server. The client will have to keep an estimate of all the related parameters used in the server.

TCP protocol transmits data packets in the unit of round. The related parameters rtt, $srtt$, $rttvar$ and RTO are updated round by round. To have an estimate of those parameters locally, the client needs to estimate those parameters each round. We use $c\_rtt[i]$, $c\_srtt[i]$, $c\_rttvar[i]$ and $c\_RTO[i]$ to represent the client's estimate on corresponding parameters that the server has at round i. When the client receives packets of the ith round, it introduces some artificial delay $delay[i]$ for their acknowledgements to extend their round trip time to $c\_rtt[i]$. It then updates $c\_srtt[i+1]$, $c\_rttvar[i+1]$ and $c\_RTO[i+1]$ for the next round. To avoid timeout retransmission, the client sets the target round trip time $c\_rtt[i]$ to be smaller than $c\_RTO[i]$, to make room for the variability of propagation delay between the server and the client. A small margin of τ is introduced such that

$$c\_rtt[i] = c\_RTO[i] - \tau \tag{2}$$

In the case when there is no packet loss, only the first data packet in each round is timed. The client would need the sequence number of the first data packet of each round, so that it could introduce new artificial delay, and update the corresponding parameters for next round. If we do not consider the limit of slow start threshold and advertised window size, the timed packets are those data packets with sequence number $2^n - 1$, while $n = 0, 1, 2....$ For the case with given slow start threshold and advertised window size, the sequence numbers of timed data packets can also be calculated accordingly. Another way of inferring the timed packets which doesn't depend on packet sequence number will be presented in Section III-C.2. Once the client identifies packets timed by the server, the procedure of updating the estimate of those $c\_$ parameters could be described as follows:

• When the first data packet arrives at the client, the client needs to decide how much artificial delay it will introduce before sending the acknowledgement packet. The client knows the initial value of RTO at the server is 6s, i.e, $c\_RTO(0) = 6$. Therefore, the client would introduce artificial delay time $dealy[0]$ for data packet of this round:

$$dealy[0] = 6.0 - 2 * rtt_0 - \tau, \tag{3}$$

where τ is the margin to make the round trip time of each packet smaller than RTO, rtt_0 is the propagation time from server to the client, which can be estimated by the client by taking the difference between a packet's arrival time at the client end and its time stamp put by the server. Then $rtt[0]$, the round trip time seen by the server, can be estimated as:

$$c\_rtt[0] = 6.0 - \tau. \tag{4}$$

• Once the server gets the acknowledgement for the first packet, it would initialize the values of $srtt$, $rttvar$ and update RTO. The client could update its estimate on these

TABLE I

RTO VS ROUND NUMBER

τ	Round 1	Round 2	Round 3	Round 4	Round 5
0.1	17.7000	27.9125	45.7813	77.0484	
0.3	17.1000	26.7375	43.5938	73.0828	
0.5	16.5000	25.5625	41.4063	69.1172	
0.7	15.9000	24.3875	39.2188	65.1516	
0.9	15.3000	23.2125	37.0313	61.1859	103.4285

parameters:

$$\begin{aligned} c\_srtt[1] &= c\_rtt[0] \\ c\_rttvar[1] &= \frac{1}{2} c\_rtt[0] \\ c\_RTO[1] &= c\_srtt[1] + 4 * c\_rttvar[1]. \end{aligned} \tag{5}$$

Then, when the client receives the second round of packets from the server, it could apply the artificial delay as

$$dealy[1] = c\_RTO[1] - 2 * rtt_0 - \tau. \tag{6}$$

This makes the round trip time of packets in the second round equal to $c\_rtt[1] = c\_RTO[1] - \tau$.

• This procedure can be carried out by the client iteratively: During round i, the client calculates the artificial delay as $dealy[i] = c\_RTO[i] - 2 * rtt_0 - \tau$. By introducing this delay, the client extends the round trip time of packets in round i to $c\_rtt[i] = c\_RTO[i] - \tau$. Then the client could always predict the server's parameters of round $i + 1$ after it gets packets of round i:

$$\begin{aligned} c\_srtt[i+1] &= \frac{7}{8} * c\_srtt[i] + \frac{1}{8} * c\_rtt[i] \\ c\_rttvar[i+1] &= \frac{3}{4} * c\_rttvar[i] + \frac{1}{4} * |c\_rtt[i] - c\_srtt[i]| \\ c\_RTO[i+1] &= min(MAX\_RTO, c\_srtt[i] + 4 * c\_rttvar[i]) \end{aligned} \tag{7}$$

$c\_RTO[i + 1]$ will be used in the next round to determine the artificial delay as $dealy[i + 1]$.

Table I shows the trend of $c\_RTO[i]$ in the first 5 rounds of data transmission. In 5 rounds, c_RTO could reach 64sec, which is MAX_RTO in BSD TCP implementation [19]. Thus for large-size data transmission, the duration of the connection could be extended by MAX_RTO/rtt_0 times. For small-size data transmission, which finishes in a couple of rounds, Table I shows the total duration of the connection could be greatly extended as well.

2) TCP with Extended Round Trip Time and Moderate Data Packet Loss: In reality, packets are lost from time to time in the network. It is common for a TCP connection to experience some packet loss during its life-time. In this section, we present a client algorithm to stretch TCP connection when there is moderate packet loss. That is to say we only study the case where at most one packet will be dropped each round. This assumption can easily be justified in a well-engineered network and has been commonly adopted in TCP performance studies. Under this assumption, the server would receive duplicate ACKs for each lost packet. Duplicate ACKs would trigger fast retransmission and fast recovery algorithm [19]. From the duplicated ACKs, the server could infer which data packet is lost, and retransmit it. At the same time, the server will reset the RTT and RTO timers to time this retransmitted data packet. The client will have to detect this retransmission and synchronize its calculation with the server's option. The algorithm presented in Section III-C.1, which uses the sequence number to infer those timed packets, no longer works in this scenario. The client must have a new algorithm to infer timed packets, including both the first packet of each new round and the retransmitted packets.

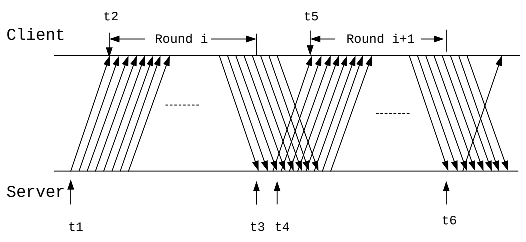
The key observation here is that there is only one RTT timer at the server end. When there is no packet loss, the server will time another packet only after it receives acknowledgement for the previous timed packet. On the other hand, if one packet is lost, server will immediately switch to time the retransmitted packet. Then the client's iterative algorithm to infer timed packets can be described as follows:

• When the client doesn't receive any duplicated ACKs, given that timed data packet in the current round has a time stamp t and the client's target round trip time is c_rtt, it can infer that the acknowledgement of the current timed packet will arrive at the server at time $t + c\_rtt$. Therefore, the next timed packet is the first packet sent out after $t + c\_rtt$. Then the client can update its estimates once he receives a packet with time stamp larger than $t + c\_rtt$.

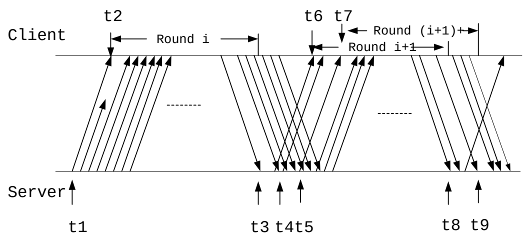
• If the client receives more than three packets with sequence number larger than k while it waits for the packet with sequence number k, it infers packet k is lost and sends duplicate ACKs back to the server to trigger the retransmission. The client also knows the server will reset its RTT and RTO timers to time the retransmitted packet. When the client receives the retransmitted packet, it again delays the acknowledgement and updates all its estimates according to the calculation in Section III-C.1.

Figure 4(a) shows how the client infers the timed data packets with no packet loss. At time t_2, the client gets the first data packet of certain round sent by the server at time t_1. The arrival time of its acknowledgement at the server end can be calculated as t_3. Therefore, data packets with time stamp between t_1 and t_3 would be considered in the same round as the current timed packet. The client introduces the same delay to acknowledgements for these data packets. At time t_5, the client gets the first data packet with a time stamp t_4 larger than t_3. The client knows another round of packets from the server arrives and the received packet is timed by the server. It then updates all its estimates.

Figure 4(b) shows how the client infers the timed packet, and shifts the effective period of the estimated parameters when some data packet gets lost. Round i starts with a packet sent out at time $t1$. The first packet of this round arrives at client end at time $t2$ and its acknowledgement received by the server at time $t3$. Then the server begins



(a) Round Adjustment: No Packet Loss



(b) Round Adjustment: With Sparse Packet Loss

Fig. 4. Round Adjustment at the Client End

to send another round of packets at time $t4$ and the timers are used to time the first packet of this new around. The client detects the start of round $i+1$ at time $t6$ and updates its estimates as in the previous example. However some packet sent in round i is lost, duplicate ACKs are received by the server. The lost packet is retransmitted at time $t5$ and RTT and RTO timers are reset to time this packet. When the client receives the retransmitted packet at time $t7$, it updates its estimates again to synchronize with the server's operation. The new parameters are effective until the acknowledgement of the retransmitted packet arrives at the server at time $t9$.

3) Simulation: We implement the mechanism above to control acknowledgement packet in TCP/Sink in NS simulator [9]. In TCP/Sink, we use the mechanism mentioned above to keep in TCP/Sink an estimate of the parameters in the server, and to introduce artificial delay time for each packet. A detailed algorithm in tracking the timed packets, calculating artificial delay time and updating the estimate of the parameters is presented in Technical Report [2].

A two-node simulation is setup to study how much the duration of TCP connection could be extended by our mechanism. One node uses TCP/Reno, and serves as a server, while the other node uses TCP/Sink, and serves as a client. A link connecting these two nodes has 5Mb bandwidth and 400ms delay. The advertised window size is the default value in NS. A ftp flow is setup between the server and the client. The amount of data from the server to the client is changed from 1 to 5000. The experiments are carried out for the two cases: one has no data packet loss; the other has data packet loss from the server to the client, with packet loss rate as 0.002.

The experiments are used to compare normal TCP connection with slow TCP connection. Figure 5 shows the simulation result. x-axis represents the number of data packet the ftp flow wants to send from the server to the client. y-axis represents the duration of TCP connection in Log scale. The durations of both normal TCP connection and slow TCP connection are shown in the figure. Figure 5 shows the case when the link allows data packet loss[3]. The

[3]We refer readers to our technical report [2] for the case when there is no loss in the link.

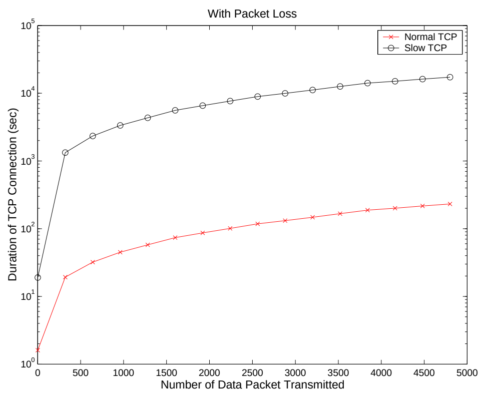


Fig. 5. Duration of Normal TCP vs Slow TCP (Packet Loss Rate 0.002)

figure shows there is a gap between two curves. This gap increases at the beginning and converges to certain value, i.e, the ratio of the duration of slow TCP connection over that of normal TCP connection increases and converges to a constant when the number of data packets transmitted increases. As Table I shows, in a couple of rounds, the extended round trip time could become very close to MAX_RTO, and thus the ratio of the duration between slow TCP connection and normal TCP connection would be approximated by $MAX\_RTO/rtt_0 \cong 64/0.8 = 80$. This trend has been reflected in Figure 5.

In summary, our simulation result shows the duration of slow TCP connection is tens of times that of normal TCP connection. In our technical report [2], we also list some other techniques which could be combined with our mechanism to extend the duration of TCP connections.

D. Discussion on Diverse TCP Implementation

Since RFC does not specify the detailed implementation of TCP, the implementation in different operating systems varies. However, this does not change the fact that the client could control the pacing of acknowledgement packets. For example, "TimeStamp" option [5] is introduced to make the measurement of RTT easier, and could be encrypted by the server. Nevertheless, the client could still hold the data packet for the delay before it begins to process the data

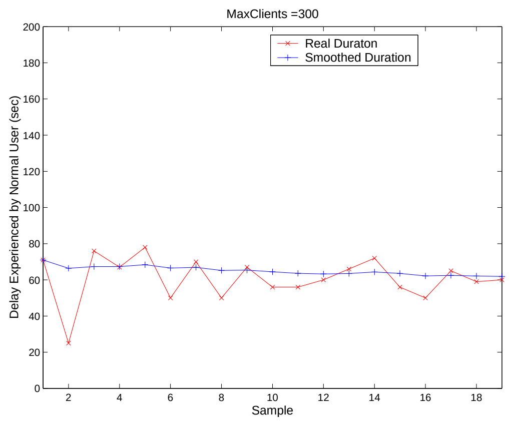


Fig. 6. Delay Experienced by Normal User with Multiple Slow TCP Connections ($MaxClient = 300$)

packet and generates the acknowledgement packet.

IV. Experimental Study: The Impact of Slow TCP Connections on Normal Users

In this section, we present a study on how a normal user could be affected by slow TCP connections. The web server has a limit on the number of clients it can serve simultaneously. In Apache web server, this number is reflected in the directive $MaxClients$ in the configuration file [13]. The default value of $MaxClients$ is 150. When a web file request arrives, if the number of clients served by web server is less than $MaxClients$, the request is served right away; if the number of clients served by the web server is $MaxClients$, the request is put into the waiting queue, only after some clients being served finish could the request in the queue be put into service.

We use the computer setting in figure 1. $Client1$ will initiate multiple threads to simultaneously retrieve the 200KB file from the web server. The delay introduced by NIST NET is 12sec. $Client2$ would always have one normal TCP connection with the server, also to retrieve the 200KB file.

In our experiment, we present a simple scenario to show a normal user could be affected by slow TCP connections. We make $Client1$ initiate twice the number of $MaxClients$. Therefore, the server would always have $MaxClients$ requests being served, and $MaxClients+1$ requests in the queue waiting for service. Therefore, the single request from client2 will have to experience the average delay introduced by the long service time of slow TCP connections.

Figure 6 shows the delay experienced by the normal user, when there are multiple slow TCP connections requesting web files at the same time. The delay is monitored at the client end, since this is more likely to reflect the web surfer's experienced time. Figure 6 shows 19 samples, and the real delays experienced by the normal user are shown. This result shows the normal user experiences about 80-100sec delay, which is closed to the delay introduced by $Client1$ in Figure 2.

In summary, with multiple slow TCP connections to request web file from web server, the normal user would experience similar delay like slow TCP connections. Moreover, these slow TCP connections involves small TCP traffic. Therefore slow TCP connections could be used to extend occupance of resource at application layer and result in denial of service.

V. DoS Attack with Slow TCP Connections and its Possible Solutions

In previous sections, we have shown TCP client can easily slow down the sending rate of the server and thus stretch its connection. In this section, we investigate the possible denial of service attacks using multiple *slow* TCP connections. In view of the vulnerability exploited by slow TCP connections, we present a couple of possible counter measures to such attacks and discuss the difficulties in completely solving this problem.

A. Denial of Service Attack Using Slow TCP Connections

Slow TCP connections can be used by malicious users to launch denial of service attacks to web servers. For a server, the number of the connections for specific application is limited. For example, in Apache Web server, there is a parameter called $MaxClients$ which specifies how many clients the server could serve simultaneously(the default value is 150) [13]. If malicious users establish many slow TCP connections with the server, and once the number of connections reaches the maximum value, legitimate users will be totally blocked out.

From an attacker's point of view, slow TCP connection has the following properties suitable to denial of service attack: (1) A slow TCP connection behaves like a "normal" TCP connection: three-stage connection, no frequent timeout retransmissions, low traffic rate. (2) Since each slow TCP connection can last for a long time, a moderate request rate is sufficient to overwhelm a server. This reduces the cost of DoS attacks in comprising a large number of machines.

The only abnormal symptom of slow TCP is its long round trip time. However, the server has to be robust to allow a large range of round trip time. Due to dramatic variation of traffic conditions and the heterogeneity in networks, round trip time varies a lot. The past and recent studies [10] [1] show TCP Round-trip Time has great variability. Recent study observes RTTs range from *1ms* to more than *200s*. Without additional mechanism, the sever would not be able to tell whether the receiver manipulates acknowledgement packet to control Round-trip time or not.

Recently, in order to deal with DDoS flooding attack, there are some mechanisms which suggest controlling traffic at the client end [6]. When the edge router detects there is a DDoS attack, it will control the rate of flows. Therefore, some artificial delay is introduced into the flow. This has made it valid to delay traffic deliberately. In our study, we emphasize that the duration of TCP connection could be extended tens of times by slow TCP connections, which involves with low volume of traffic. To make the situation more complicated, varied durations of slow TCP connections could be introduced, and make the detection of this kind of attack more difficult.

B. Possible Solutions and Their Limitation

There are two major factors contributing to the application of slow TCP in DoS attacks: one is its low data transmission rate, the other is the trust inherent in TCP protocol. In view of these, people could propose the possible solutions as follows.

• Increase server's capacity. It can be achieved by employing the high end servers or server clusters. It has proven to be an efficient way to deal with DoS attacks. Despite the cost of increasing capacity, it doesn't solve the problem completely. Attackers can always compromise more computers to launch larger scale distributed DoS attack.

• Disconnect slow TCP connection. This method is to disconnect those TCP connections whose data transmission is slower than certain threshold. However, this method could deny certain portion of legitimate users. Moreover, the attackers could extend the duration to the threshold allows.

• Track the round trip time between the server and the neighboring hosts of the client. This method has a hope that the neighboring hosts of the attacker have normal behavior as protocol specifies. Therefore, monitoring the round trip time between the server and the neighboring hosts of the client would help the server to detect the abnormal behavior of the attacker.

• Recently [18] proposed an adaptive overload control for busy internet servers (called Haboob). Haboob divides the data service into several stages. Each stage maintains relevant queues. The server would monitor its performance such as the response time, and adjust the admission control policy accordingly. Therefore, Haboob could adaptively change the number of connections allowed dynamically. However, Haboob has the following vulnerabilities: (1) Haboob uses the response time as performance criterion. The response time of the extended TCP connections provides the server the wrong information. (2) The admission control used in Haboob is achieved by dropping the requests from both legitimate and illegitimate users. How to differ legitimate users from illegitimate users remains an unsolved problem.

In summary, the above analysis points out the difficulties in the possible solutions. This suggests a further study is needed on how to solve the denial of service by client-controlled slow TCP.

VI. Conclusion and Discussions

In this paper, we present a mechanism for web clients to control the sending rate of web servers just by manipulating the pace of acknowledgement packets. We show that the duration of TCP connection can be slowed down and its duration can be stretched by tens of times. Those stretched slow TCP connections can possibly be used in denial of service attacks to web servers. Slow TCP connections behave like normal TCP connections. This makes the detection of such attacks difficult. Our study calls for more attention on the study of TCP in the context of security.

We hope this work could help demonstrate once again that the Internet security needs to be reviewed as a whole, and starts from the basic assumptions and architecture principles. For example, to counteract the scheme described in this paper, one could try to shorten the TCP timeout limit. However doing so would immediately reduce the reliability of TCP connections in its ability to fight against congestion bursts on the way. To balance the risks brought in via trust, we need to consider scalable and easy-to-manage monitoring and defense system for trust-based attacks. Current flat routing structure of the Internet makes this task very difficult and it seems substantial research effort is needed to resolve the entanglement of the trust, convenience, security and reliability, and the overall architecture.

References

[1] J. Aikat, J. kaur, F. Smith, and K. Jeffay. Variability in TCP Round-trip Times. *Proceedings of the Internet Measurement Conference*, pages 279–284, 2003.

[2] S. Cai, Y. Liu, and W. Gong. Client-Controlled Slow TCP and Denial of Service. Technical Report TR-04-CSE-04, Department of Electrical And Computer Engineering, University of Massachusetts, Amherst, 2004.

[3] T. Darmohray and R. Oliver. Hot Spares for DoS Attacks. *login*, 25(7), July 2000.

[4] A. Hussian, J. Heidemann, and C. Papadopoulos. A Framework for Classifying Denial of service Attacks. *Proceedings of ACM SIGCOMM*, pages 99–110, 2003.

[5] V. Jacobson, R. Braden, and D. Borman. RFC 1323: TCP Extensions for High Performance. May 1992.

[6] J. Mirkovic, P. Reiher, and G. Prier. Attacking DDoS at the Source. *Proceedings of the IEEE International Conference on Network Protocols*, 2002.

[7] D. Moore, G. Voelker, and S. Savage. Inferring Internet Denial of Service Activity. *Proceedings of USENIX Security Symposium*, 2001.

[8] NIST NET. "http://snad.ncsl.nist.gov/itg/nistnet/".

[9] The Network Simulator: ns 2. "http://www.isi.edu/nsnam/ns/".

[10] V. Paxson. *Measurements and Analysis of End-to-End Internet Dynamics*. PhD thesis, University of California, Berkeley, 1997.

[11] S. Savage, N. Cardwell, D. Wetherall, and T. Anderson. TCP Congestion Control with a Misbehaving Receiver. *ACM Computer Communications Review*, 29(5):71–78, October 1999.

[12] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical Network Support for IP Traceback. *Proceedings of ACM SIGCOMM*, pages 295–306, 2002.

[13] Apache2 Web Server. "http://httpd.apache.org/".

[14] A. Snoeren, C. Partridge, L. Sanchez, C. Jones, F. Tchakountio, S. Kent, and W. Strayer. Hash-based IP Traceback. *Proceedings of ACM SIGCOMM*, pages 3–14, 2001.

[15] D. Song and A. Perrig. Advanced and Authenticated Marking Schemes for IP Traceback. *Proceedings of Infocom*, pages 878–886, 2001.

[16] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the Internet in Your Spare Time. *Proceedings of the 11th USENIX Security Symposium*, 2002.

[17] SynCookie. "http://cr.yp.to/syncookies.html/".

[18] M. Welsh and D. Culler. Adaptive Overload Control for Busy Internet Servers. *Proceedings of the 4th USENIX Conference on Internet Technologies and Systems*, March 2003.

[19] G. Wright and R. Stevens. *TCP/IP Illustrated*, volume 2. Addison-Wesley Pub Co.