A Way to Estimate TCP Throughput under Low-Rate DDoS Attacks: One TCP Flow

Minh Viet Kieu[†], Dai Tho Nguyen[‡] and Thanh Thuy Nguyen[*]
University of Engineering and Technology, Vietnam National University
Hanoi, Vietnam
Email: [†]15028023@vnu.edu.vn, [‡]nguyendaitho@vnu.edu.vn, [*]nguyenthanhthuy@vnu.edu.vn

***Abstract*—TCP-targeted low-rate distributed denial-of-service (LDDoS) attacks were first introduced by A. Kuzmanovic and E. Knightly in 2003. The authors also proposed a simple model to quantify TCP throughput under LDDoS attacks. Since then, there have been many researchers attemping to estimate the throughput, such as Luo et al. We agree with them upon the sketch of TCP congestion window under a successful LDDoS attack but we find out that there are more cases than what has been specified. Moreover, the relative error of Luo's estimation method is still high. Our goal in this paper is to propose a simple but more accurate method to estimate TCP throughput of a single TCP flow under such DDoS attacks. Our estimation values in various scenarios are compared with the results of simulations performed with NS-2 simulator, so that the effectiveness of our method is illustrated.**

***Index Terms*—Low-rate DDoS attack, estimating TCP throughput, TCP's congestion control algorithm, DropTail.**

I. Introduction

Estimation of TCP throughput under LDDoS attacks is important because it relates directly to the evaluation of the effectiveness of a LDDoS attack. One such estimation appeared at the same time as the introduction of LDDoS attacks. In the foundation paper of LDDoS attacks [1], the authors built a scenario in which a single bottleneck queue is shared by n long-lived TCP flows with heterogeneous round trip time (RTT) and a single DoS flow. Suppose the round trip times of the TCP flows are RTT_i, $i = 1, \ldots, n$ and the DoS flow is a square-wave DoS stream with period T. Then we have the following result given in [1]:

$$\rho(T) = \frac{\left\lceil \frac{minRTO}{T} \right\rceil T - minRTO}{\left\lceil \frac{minRTO}{T} \right\rceil T} \qquad (1)$$

if the two following conditions are met:

(C1) $l' \geq RTT_i$

(C2) $minRTO > SRTT_i + 4 \times RTTVAR_i$

where $\rho(T)$ is the normalized throughput of the aggregate TCP flows and l' is the outage[1] length. In addition, equation (1) is derived based on two assumption. The first one is that the TCP flows utilize full bandwidth of the bottleneck link after the end of each retransmission timeout. The second assumption is that $RTO = minRTO$ for $T > minRTO$ although this is not always true. So equation (1) is often used as an upper bound in practice.

[1]Outage is the time period where attack burst makes all TCP packets loss.

Luo et al. [2] proposed a mathematical model to quantify the TCP throughput, but the relative error of their estimation method is still high (e.g., the average relative error in the cases with a single TCP flow is 10.34%). Our goal in this paper is to propose a simple but more accurate method to estimate TCP throughput of a single TCP flow under LDDoS attacks. Simulation results show, at least for some discrete cases presented here, that our method outperforms Luo's method with the respect of the average relative error.

The rest of this paper is organized as follows. In Section II, we present network and attack traffic models and various assumptions about them and then briefly describe TCP's behavior under low-rate DDoS attacks. Section III presents our theoretical estimation of TCP throughput of a single TCP flow under low-rate DDoS attacks. Section IV presents simulation results used to verify our theoretical estimation in Section III. We coclude this paper in Section V.

II. Models and assumptions

A. Network model

The network model we will consider in this study is shown in Figure 1. In this network, there are N_c long-lived TCP flows[2] originating from TCP-1 to TCP-Nc and terminating at Receiver. By a *long-lived* TCP flow, we mean a TCP flow where the TCP sender always has data to send. We also assume that each TCP flow has a maximum window size that is large enough so that TCP's congestion window size, denoted by $cwnd$,[3] never exceeds that value, therefore the maximum window size is not an issue in our estimation. All TCP flows transmit data that is encapsulated within packets with a constant packet size of M_{TCP} bytes. The version of TCP is NewReno. Each time TCP receiver receives a TCP packet with new data, it immediately sends an ACK packet of size M_{ACK} bytes back to the sender with zero packet processing delay. All link from the network have bandwidth of nt_bw Mbps (**nt_bw:** network bandwidth) and one-way propagation delay of nt_dl ms (**nt_dl:** network delay), except the link between router R0 and router R1 that has bandwidth of bn_bw Mbps (**bn_bw:** bottleneck bandwidth) and one-way propagation delay of bn_dl ms (**bn_dl:** bottleneck delay).

[2]Although we only consider the case with $N_c = 1$ in this study, but we still examine the network with N_c long-lived TCP flows for future use.

[3]$cwnd$ is measured in units of packets rather than bytes for simplicity.

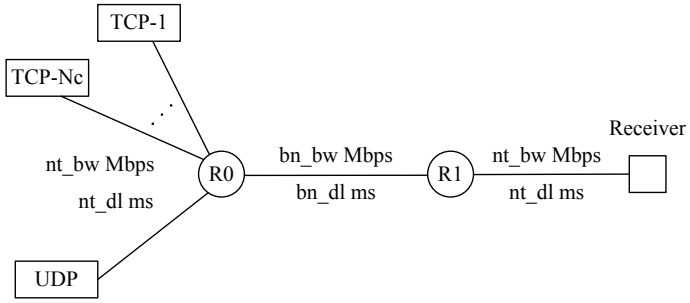

Fig. 1: Network topology.

Because bn_bw is set to be extremely smaller than nt_bw, the link between router R0 and router R1 become the congestion point of the network. The queue of the congested link can accomodate at most B packets. All links use DropTail queue discipline.

B. Attack model

In our network, there is also one attack machine named UDP (see Figure 1). The attack machine sends packets with a constant packet size of M_{UDP} bytes. In general, the attack traffic pattern is as in Figure 2, where s is the starting time of the attack, T_a is the attack cycle, T_b is the burst length and R_b is the rate at which the attack sends packets into the network.

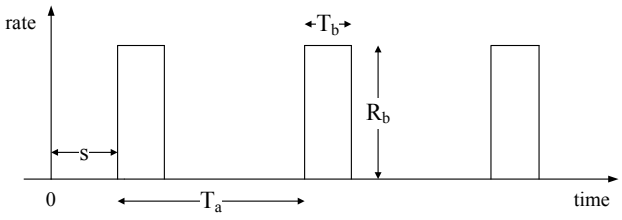

Fig. 2: A low-rate DDoS flow.

The attack flow can be thought of as a representative of a LDDoS attack in practice with a large number of attack machines as you will see later when we conduct simulations to validate our theoretical estimation. To be effective, the attack flow should have burst length T_b large enough to cause a TCP window's worth of packets lost (T_b must be larger than round trip time of every TCP packet but it needs to be as small as possible to keep average attack rate small, thereby the attack can bypass different statistical-measurement-based DDoS attack detection mechanisms), rate R_b large enough to keep the queue at the bottleneck link full to cause all packets lost (R_b is usually greater than or equal to the bottleneck link's bandwidth), and the attack cycle T_a is at the scale of $minRTO$. Without loss of generality, we assume that R_b equals to nt_bw, the bandwidth of the links connecting left-most computers to the router R0 in Figure 1.

In our study, we only consider about LDDoS attacks with an attack cycle $T_a \geq 1$ second because performing the attack with $T_a < 1$ second is an unnecessarily useless waste of attack resources due to the existence of an attack with an attack cycle not less than 1 second having the same attack efficiency (according to [1] and using a simple derivation). An important thing is that, in this study, we focus particularly on a subset of the attack, successful low-rate DDoS attacks. By the name describing and as defined in [2], a successful low-rate DDoS attack must force every TCP flow fall into timeout state after each outage. TCP flows then go to slow start phase RTO seconds later (usually $minRTO$ seconds) rather than going to congestion avoidance phase.

Now let us define three parameters that is used throughout this work. Suppose T is the two-way propagation delay of TCP packets.[4]

$$T = (nt\_dl + bn\_dl + nt\_dl) \times 2 \quad \text{(s)} \tag{2}$$

So that the round trip time[5] of a data packet when the queue at the bottleneck link is empty is:

$$T_1 = T + \frac{(M_{TCP} + M_{ACK}) \times 8}{bn\_bw} + 2 \times \frac{(M_{TCP} + M_{ACK}) \times 8}{nt\_bw} \tag{3}$$

Finally, suppose C is the bottleneck link's bandwidth in units of TCP packets per second. We then have:

$$C = \frac{bn\_bw}{M_{TCP} \times 8} \quad \text{(packets/s)} \tag{4}$$

C. TCP's behavior under low-rate DDoS attacks

As mentioned above and according to TCP's congestion control mechanism (see [1], [3] for more details), under successful LDDoS attacks, a TCP flow is forced to be timeout and go to slow start phase. At that time, its congestion window size reduces to one packet and one packet is sent. In slow start phase, the window size will be increased by one packet each time an ACK packet arrives at TCP sender. This is equivalent to doubling the window size after each RTT period.

If the attack cycle T_a is large enough so that the window size eventually increases to a value that is greater than a slow start threshold, $ssthresh$, TCP then switches to congestion avoidance phase. In this phase, the increase of the window size is much more slowly than in the slow start phase. When an ACK packet arrives at the sender, the window size is increased by $\frac{1}{cwnd}$ packets in which $cwnd$ is the current congestion window size (as compared to the increase of one packet in the slow start phase). Thereby, the window size is increased roughly by one packet per RTT. This phase is designed to serve as a means to reduce the number of packets lost due to the exponential increase of congestion window size in slow start phase.

If the attack cycle T_a is further longer, TCP window size can even reach the capacity of the path from sender to receiver

[4]This T symbol has a different meaning with T symbol from equation (1). In equation (1) T is the attack cycle. In our study, the attack cycle is denoted by T_a.

[5]The second term in the right hand side of equation 3 is the total transmission time of a data packet and its corresponding ACK packet at the bottleneck link, and the third term is the total transmission time of a data packet and its corresponding ACK packet at the left-most and right-most links.

(e.g., CT in our network) plus the size of the queue at the bottleneck link (e.g., B in our network), at that time only one packet is dropped because the increase of the window size in congestion avoidance phase is one packet per RTT. If the congestion window size is not less than four packets when a packet is dropped, the packet loss is usually recovered through the receipt of three duplicate ACK packets. By the reason, this case is considered as a light congestion situation because only one packet has been lost but other packets still get through the network and reach their destination so that the ACK packets can go back to the TCP sender. TCP then sets its $ssthresh$ threshold to one-half of the current $cwnd$ and $cwnd$ is set to the new value of $ssthresh$ plus three packets because there are three data packets that have left the network and are being in the buffer memory of the destination machine. After that, TCP resends the seemingly lost packet and enters fast recovery phase in which it waits for an acknowledgment of the resent packet before returning to congestion avoidance phase. If the resent packet has not been acknowledged, TCP enters slow start phase with $cwnd$ set to one packet.

RTT value of a TCP flow can be different at any instant of time due to the oscillation of queue size at routers. In our network, when the congestion window size of a TCP flow is greater than CT (i.e. the number of TCP packets the pipe from sender to receiver can accomodate, including ACK packets in the reverse direction), the queue at the bottleneck link builds up and the RTT increases. In other hand, when the window size is smaller than CT, the RTT value is exactly equal to T_1 because the queue is empty.

Although the window adjustment algorithm operates in discrete manner as described above, we can approximate the window process as a continous (or partial continuous) function of time. This approximation can simplify our analysis so much but still allow estimating TCP throughput under low-rate DDoS attack with a small error.

III. A SINGLE TCP FLOW

In this section, we will estimate TCP throughput of a single TCP flow under low-rate DDoS attacks by theoretical analysis. We divide the problem into two sub-problems depending on whether the attack cycle is long enough that the congestion window size has been halved before timeout or not. We first examine the sub-problem with the window size that has not been halved before timeout and then the remaining sub-problem. Each sub-problem is further divided into several smaller cases depending on the value of CT compared to the parameters $ssthresh$, $\frac{W_{max}}{2}$ (will be discussed later), and W.

A. TCP window size has not been halved before timeout

1) **CT $<$ ssthresh:** Assume W is the final window size TCP has before timeout and TCP is in the first congestion avoidance period (i.e. the congestion window size has not been halved since TCP enters congestion avoidance phase), as depicted in Figure 3. This happens when $\frac{(T_a-1)}{T_1}$ is large enough because when $\frac{(T_a-1)}{T_1}$ is small, the TCP throughput is nearly zero.

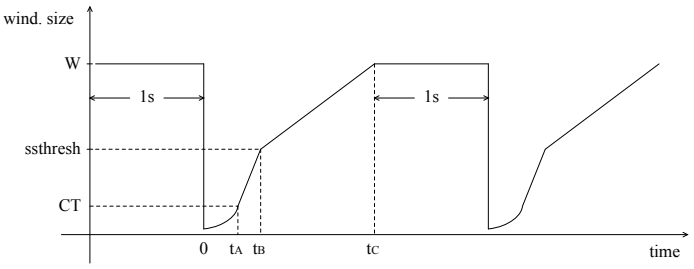

Fig. 3: Congestion window when CT $<$ ssthresh.

We always have:

$$W \leq B + \lfloor CT \rfloor + 2 \tag{5}$$

Let $W_{max} = B + \lfloor CT \rfloor + 2$. So the inequation means that we always have $W \leq W_{max}$. The reason of the inequation is as follows. Assume that at an instant of time, the congestion window size is the maximum possible number of packets the pipe from sender to receiver can accommodate, that is, $B + \lfloor CT \rfloor$ (not including the fractional part), and an ACK packet arrives at the sender that makes the window reach $B + \lfloor CT \rfloor + 1$, TCP is in congestion avoidance phase[6] and there is no attack. Then TCP sends out two packets back to back, one compensating for the leaving of the data packet whose ACK packet has just arrived and one because of the new value of the window. But the extra packet makes the pipe overload, and when it arrives at router R0, it is dropped because the queue is full. However, from the time the packet was sent to the time the sender realizes that the packet has been lost, a window's worth of packets have been sent that makes the window increase to about $B + \lfloor CT \rfloor + 2$ before it is halved. This is a normal case with no attack, when the attack happens the maximum value of the window size may even be smaller. So the inequation holds in any circumstance.[7] According to TCP's congestion control mechanism,[8] we also have:

$$ssthresh = \left\lfloor \frac{W}{2} \right\rfloor \tag{6}$$

The problem is that we don't know the value of W, and thus the value of $ssthresh$. We only have the known parameters of the attack: T_a, T_b, R_b and the known parameters of the network environment: T, C, B. We must find the value of

[6]TCP cannot be in slow start phase because it has learned the maximum value before.

[7]The inequation does not hold when TCP is in slow start phase when it has just started its transmission. In this case, the window size can reach to about two times of the maximum window size W_{max}. But after that and because of a TCP packet that is dropped due to buffer overflow at the bottleneck link, TCP learns that window size and sets its slow start threshold $ssthresh$ to one-half of the window size. This process (a packet drop, and then $ssthresh$ is set to one-half of the current window size) is repeated again and again until $ssthresh$ is smaller than W_{max} and thereby when TCP reaches W_{max} it is definitely in congestion avoidance phase. So the inequation (5) holds as long as we still assume that TCP is long-lived.

[8]Actually, we follow behavior of TCP in NS-2 simulation environment.

W from these known parameters and then estimate TCP throughput accordingly.

In Figure 3, we choose the origin of time is the time at which TCP's congestion window size is one packet after a timeout. Assume t_A, t_B, and t_C is the time at which the window size is CT, $ssthresh$, and W, respectively.

When $0 \leq t \leq t_A$, the window size is less than CT, so the RTT of data packets is T_1 because the queue at the bottleneck link is empty. Furthermore, TCP is still in slow start phase, so the window size is doubled every T_1 seconds. Therefore, we approximate the window process during this time period by the following continuous function:

$$W(t) = 2^{\frac{t}{T_1}}, \quad t \in [0, t_A]$$

Replace $t = t_A$ into the equation and note that $W(t_A) = CT$, we have:

$$CT = 2^{\frac{t_A}{T_1}} \quad \Rightarrow \quad t_A = T_1 \log_2 CT \tag{7}$$

When $t_A \leq t \leq t_B$, the window size is greater than CT, so the RTT is greater than T_1 because the bottleneck link's queue has built up. This means that during this time period, the bottleneck link transmits at its full rate of C TCP packets/s and the TCP receiver returns ACK packets with the same rate of C ACK packets/s. Using differential equation, we have:

$$\frac{\mathrm{d}W}{\mathrm{d}t} = \frac{\mathrm{d}W}{\mathrm{d}a}\frac{\mathrm{d}a}{\mathrm{d}t}$$

in which $a(t)$ is the number of ACK packets received by TCP sender during the time period $[0, t]$ and $\mathrm{d}a/\mathrm{d}t$ is the rate at which TCP sender receives ACK packets. Because TCP is still in slow start phase, we have:

$$\frac{\mathrm{d}W}{\mathrm{d}a} = 1$$

and because of TCP receiver returning ACK packets with the rate of C packets/s, we have:

$$\frac{\mathrm{d}a}{\mathrm{d}t} = C$$

Equation above depends strictly on the fact that ACK packets never encounter a queue on their way to the sender and any two consecutive ACK packets arrive at the sender with a spacing equal to the transmission time of a data packet at the bottleneck link. If there is cross traffic in the reverse direction (e.g., traffic in the Internet), the equation does not hold. Finally, we have:

$$\frac{\mathrm{d}W}{\mathrm{d}t} = \frac{\mathrm{d}W}{\mathrm{d}a}\frac{\mathrm{d}a}{\mathrm{d}t} = 1 \cdot C = C \quad \Rightarrow \quad \mathrm{d}W = C\,\mathrm{d}t$$

Take integral of two sides of the above equation from t_A to t and note that $W(t_A) = CT$, we have:

$$\int_{t_A}^{t} \mathrm{d}W = \int_{t_A}^{t} C\,\mathrm{d}t \quad \Rightarrow \quad W(t) - CT = C\,(t - t_A)$$

Replace $t = t_B$ into the above equation and note that $W(t_B) = ssthresh$, we have:

$$t_B - t_A = \frac{ssthresh - CT}{C} = \frac{ssthresh}{C} - T \tag{8}$$

When $t_B \leq t \leq t_C$, because the window size is still greater than CT, so the bottleneck link's queue is not empty, but TCP has been switched to congestion avoidance phase. In this phase, every time an ACK packet is received at TCP sender, the window size is increased by $\frac{1}{cwnd}$, so we have:

$$\frac{\mathrm{d}W}{\mathrm{d}t} = \frac{\mathrm{d}W}{\mathrm{d}a}\frac{\mathrm{d}a}{\mathrm{d}t} = \frac{1}{W} \cdot C = \frac{C}{W} \quad \Rightarrow \quad W\,\mathrm{d}W = C\,\mathrm{d}t$$

Take integral of two sides of equation (9) from t_B to t and note that $W(t_B) = ssthresh$, we have:

$$\int_{t_B}^{t} W\,\mathrm{d}W = \int_{t_B}^{t} C\,\mathrm{d}t \;\Rightarrow\; W^2(t) - (ssthresh)^2 = 2\,C\,(t - t_B)$$

Replace $t = t_C$ into the above equation and note that $W(t_C) = W$, we have:

$$t_C - t_B = \frac{W^2 - (ssthresh)^2}{2\,C} \tag{9}$$

From (7), (8), and (9) after adding left-side quantities and right-side quatities separately, we have:

$$t_C = \frac{W^2 - (ssthresh)^2}{2\,C} + \frac{ssthresh}{C} + T_1 \log_2 CT - T \tag{10}$$

Solving this equation and note that $t_C = T_a - 1$, we can get the value of W if it exists, and thus the value of $ssthresh$. To be valid, the value of W must satisfy the conditions $CT < ssthresh$, and of course, $W \leq B + \lfloor CT \rfloor + 2$.

For estimating TCP throughput under the attack, we only have to count the number of TCP packets sent in the period $[0, t_A]$ because when the window size is greater than CT the TCP throughput is C packets/s. For this purpose, we use the concept of round. A round begins when TCP sender sends a whole window of packets into the network in a back-to-back manner and it finishes when an ACK packet whose data packet is one of the packets in the window arrives at the sender, at that time a new round begins.

Assume n is an integer satisfying that:

$$2^{n-1} < CT \leq 2^n, \quad n = 1, 2, \ldots \tag{11}$$

We don't consider the case with $CT \leq 1$ because it is a trivial case. In that case, TCP will utilize full bandwidth of the bottleneck link after each timeout because the congestion window size is not less than one packet at any time. So TCP throughput under attack is very easy to compute.

Figure 4 shows the number of packets sent in each round when the TCP flow ramps up after a timeout. Each packet is denoted by a square. The number of packets sent is illustrated directly by its value or indirectly by the number of the squares. TCP starts from a window of only one packet, and in the next rounds the window size is two, and then four packets and

so on until it reaches 2^{n-1} packets. Note that in this case, every round has the same length of T_1 seconds because the congestion window size is always less than CT.

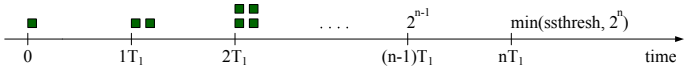

Fig. 4: Timeline of a TCP flow under a low-rate DDoS attack provided that CT $<$ ssthresh.

In the round at time nT_1, TCP sends a window of $ssthresh$ or 2^n packets depending on which value it reaches first. Regardless of the value the window size will take, they are both greater than or equal to CT. This observation is based directly on TCP's congestion control algorithm. Imagine that if $2^n \leq ssthresh$, then $min(ssthresh, 2^n) = 2^n$, the window size at time nT_1 is exactly 2^n because TCP is still in slow start phase. If $ssthresh < 2^n$, and thus $min(ssthresh, 2^n) = ssthresh$, the window size at time nT_1 is $ssthresh$ by the following reason. Consider the moment at which the window size has just reached the value $ssthresh$ due to the arrival of an ACK packet, named p_1. It must happen at time nT_1 because $2^{n-1} < CT < ssthresh < 2^n$. It means that p_1 belong to the burst of ACK packets corresponding to a whole window of data packets TCP sent at time $(n-1)\,T_1$. When congestion window size reaches $ssthresh$, TCP automatically switches to congestion avoidance phase. The next ACK packet received makes the window size increase to the value:

$$ssthresh + \frac{1}{ssthresh}$$

and the next ACK packet received makes the window size increase to:

$$\left(ssthresh + \frac{1}{ssthresh}\right) + \frac{1}{ssthresh + \frac{1}{ssthresh}}$$

and so on. In the increment regime of congestion avoidance phase, the window size is increased roughly by one packet every RTT after the sender receives the burst of ACK packets.[9] But in this case, at least one ACK packet, that is, p_1, is used to increase the window size from 2^{n-1} packets of the previous round to $ssthresh$ packets, so that there is not enough ACK packet to increase the window size to the value of $ssthresh+1$. The number of data packets actually sent in this round is the floor of the window size and will be $ssthresh$.

From above analysis, we only consider about the time period $[0, nT_1]$ because after the time nT_1, the window size is greater than CT so that the TCP throughput is C packets/s. The number of TCP packets sent in the period $[0, nT_1]$ is:

$$1 + 2 + 4 + \cdots + 2^{n-1}$$

The TCP throughput under LDDoS attack is calculated by:

$$\frac{1 + 2 + 4 + \cdots + 2^{n-1} + (t_C - nT_1)\,C - \lfloor W \rfloor}{T_a} \quad (12)$$

[9]The value of the increase is exactly less than one packet as described in [4].

because a successful LDDoS attack must force the TCP flow lose a whole window of data packets.

2) **ssthresh $<$ CT $<$ W:** This case is depicted in Figure 5.

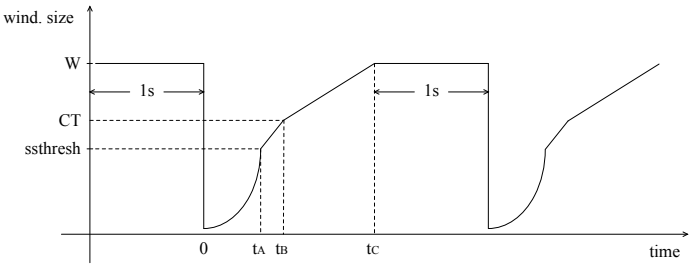

Fig. 5: Congestion window when ssthresh $<$ CT $<$ W.

Assume m is an integer satisfying that:

$$2^{m-1} < ssthresh \leq 2^m, \quad m = 1, 2, \ldots \quad (13)$$

We can assume as such because the slow start threshold, $ssthresh$, is not permitted to be less than two packets as specified in [5]. Apply the concept of round again and consider Figure 6.

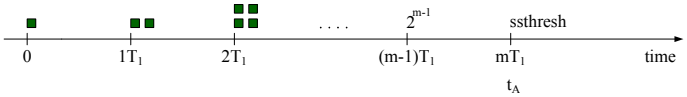

Fig. 6: Timeline of a TCP flow under a low-rate DDoS attack provided that ssthresh $<$ CT $<$ W.

From the figure, we can easily see that:

$$t_A = mT_1 = T_1 \left\lceil \log_2 ssthresh \right\rceil \quad (14)$$

Note that at the time $t = t_A = mT_1$, the window size is $ssthresh$ rather than 2^m as expected because of the TCP's congestion control algorithm. The analysis is similar as the previous case.

When $t_A \leq t \leq t_B$, because TCP has been switched to congestion avoidance phase and the rate of the ACK packets returning to the source equals to the sending rate, so we have:

$$\frac{\mathrm{d}W}{\mathrm{d}t} = \frac{\mathrm{d}W}{\mathrm{d}a}\frac{\mathrm{d}a}{\mathrm{d}t} = \frac{1}{W}\frac{W}{T_1} = \frac{1}{T_1} \quad \Rightarrow \quad \mathrm{d}W = \frac{1}{T_1}\,\mathrm{d}t$$

Take integral of two sides of the above equation from t_A to t and note that $W(t_A) = ssthresh$, we have:

$$\int_{t_A}^{t} \mathrm{d}W = \int_{t_A}^{t} \frac{1}{T_1}\,\mathrm{d}t \quad \Rightarrow \quad W(t) - ssthresh = \frac{1}{T_1}\,(t - t_A)$$

Replace $t = t_B$ into the equation and note that $W(t_B) = CT$, we have:

$$t_B - t_A = T_1\,(CT - ssthresh) \quad (15)$$

When $t_B \leq t \leq t_C$, TCP is still in the congestion avoidance phase as well as the window size is greater than CT, so we have:

$$\frac{\mathrm{d}W}{\mathrm{d}t} = \frac{\mathrm{d}W}{\mathrm{d}a}\frac{\mathrm{d}a}{\mathrm{d}t} = \frac{1}{W} \cdot C \quad \Rightarrow \quad W\,\mathrm{d}W = C\,\mathrm{d}t$$

$$\int_{t_B}^{t} W \, \mathrm{d}W = \int_{t_B}^{t} C \, \mathrm{d}t \quad \Rightarrow \quad W^2(t) - (CT)^2 = 2C\,(t - t_B)$$

Replace $t = t_C$ into the above equation and note that $W(t_C) = W$, we have:

$$t_C - t_B = \frac{W^2 - (CT)^2}{2C} \tag{16}$$

From (14), (15), and (16), we have:

$$t_C = T_1 \lceil \log_2 ssthresh \rceil + T_1\,(CT - ssthresh) + \\ + \frac{W^2 - (CT)^2}{2C} \tag{17}$$

After solving this equation, if there exists a value of W satisfying, we must validate it using the condition $ssthresh < CT < W$. If the condition is satisfied, we then go to step two of estimating TCP throughput. The number of TCP packets sent in the period $[0, t_A]$ is:

$$n_A = 1 + 2 + 4 + \cdots + 2^{m-1}$$

Assume $R(t)$ is the instantaneous TCP throughput of the TCP flow at time t. The number of packets sent in the period $[t_A, t_B]$ is related to $R(t)$ as follows:

$$n_B = \int_{t_A}^{t_B} R(t) \, \mathrm{d}t = \int_{t_A}^{t_B} \frac{W(t)}{T_1} \, \mathrm{d}t = \frac{(CT)^2 - (ssthresh)^2}{2}$$

The number of TCP packets sent in the period $[t_B, t_C]$ is:

$$n_C = (t_C - t_B)\,C = \frac{W^2 - (CT)^2}{2}$$

because in this period, the window size is greater than CT, so the bottleneck link transmits at a full rate of C TCP packets/s. The total number of TCP packets that can get through the bottleneck link in the period $[0, t_C]$ is:

$$\sum = 1 + 2 + 4 + \cdots + 2^{m-1} - \lfloor W \rfloor + \\ + \frac{W^2 - (ssthresh)^2}{2} \tag{18}$$

and the TCP throughput under attack is calculated by $\sum / T_a$.

3) **W < CT:** This case is depicted in Figure 7.

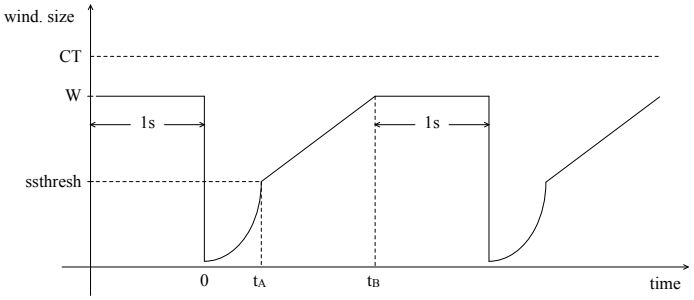

Fig. 7: Congestion window when W < CT.

In this case, we have:

$$t_B = T_1 \lceil \log_2 ssthresh \rceil + T_1\,(W - ssthresh) \tag{19}$$

and the TCP throughput under attack is completely similar to the previous case.

B. TCP window size has been halved one or more times before timeout

Three previous cases happen when TCP congestion window has not been halved before timeout or in other words, TCP is only in the first congestion avoidance period. If the window has been halved one or more times until timeout, we have four following cases. We will explore the case with $CT < ssthresh$ first.

1) **CT < ssthresh:** This case is depicted in Figure 8.

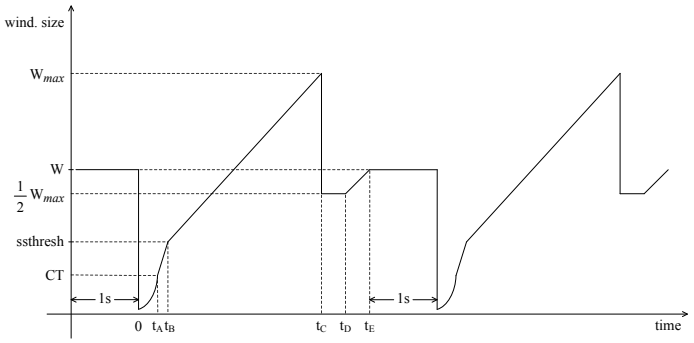

Fig. 8: Congestion window when it has been halved once before timeout and CT < ssthresh.

In Figure 8, we only show the case with just one halving of the window before timeout. When TCP detects a packet loss through the receipt of three duplicate ACK packets at time t_C, the window size is halved from W_{max} down to $\frac{W_{max}}{2}$, the lost packet is resent and TCP switches to fast recovery phase. In this phase, TCP sender maintains and does not increase the window size until time t_D when it receives the ACK packet which informs about the receipt of the lost packet. From that time, TCP re-enters the congestion avoidance phase and the window size continues to increase until it reaches W. Talking again about the resending of the lost packet, when the packet arrives at the bottleneck link, it encounters a queue with $(B-1)$ packets before it. This is because just before the time the packet is resent, TCP still transmits packets with a window size that makes the queue full. So that the RTT of the packet is T_1 plus $(B-1)$ times of the transmission time of a data packet at the bottleneck link. Denote T_{pause} is the time needed by TCP to recover one single lost packet in such a way. So we have:

$$T_{pause} = T_1 + (B-1)\,\frac{M_{TCP} \times 8}{bn\_bw} \quad \text{(s)} \tag{20}$$

In this case, we have:

$$t_E = \frac{W^2 - (ssthresh)^2}{2C} + \frac{ssthresh}{C} + \\ + T_1 \log_2 CT - T + N \times \alpha_1 \tag{21}$$

in which $t_E = T_a - 1$ and N is the number of times the congestion window has been halved before timeout and

$$\alpha_1 = \frac{3\,W_{max}^2}{8C} + T_{pause} \tag{22}$$

We must solve equation (21) with N from 1 to $\left\lfloor \frac{t_E}{\alpha_1} \right\rfloor$ to find W. If existed, the value of W must satisfy the conditions $CT < ssthresh$ and $W \leq B + \lfloor CT \rfloor + 2$.

The TCP throughput is calculated the same as in equation (12), except that in this case t_C is replaced by t_E because we always have $CT < ssthresh \leq \frac{W_{max}}{2}$. If congestion window has been halved N times before timeout, we only have to replace t_C by $T_a - 1$ because after time t_C TCP repeats itself in congestion avoidance phase and the window size shows the sawtooths pattern as described in [6], thereby the window size is always greater than CT and the bottleneck link always transmits at its full rate.

2) **ssthresh $<$ CT $<$ $\frac{\mathbf{W_{max}}}{\mathbf{2}}$:** This case is depicted in Figure 9.

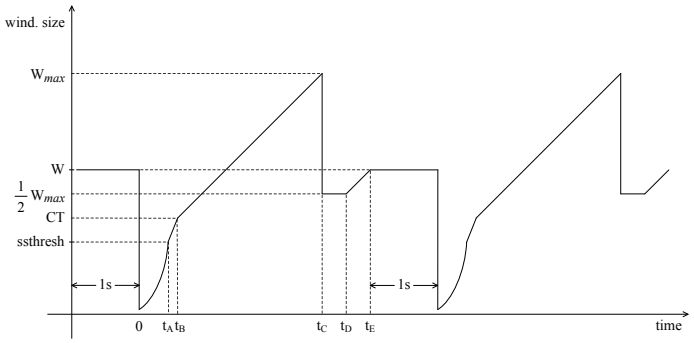

Fig. 9: Congestion window when it has been halved once before timeout and ssthresh $<$ CT $<$ $\frac{W_{max}}{2}$.

In this case, we have:

$$t_E = T_1 \lceil \log_2 ssthresh \rceil + T_1 (CT - ssthresh) + \\ + \frac{W^2 - (CT)^2}{2C} + N \times \alpha_1 \quad (23)$$

in which α_1 is the same as in equation (22). We must solve equation (23) with N from 1 to $\left\lfloor \frac{t_E}{\alpha_1} \right\rfloor$.

The total number of TCP packets that can get through the bottleneck link in the attack period is:

$$\sum = 1 + 2 + 4 + \cdots + 2^{m-1} - \lfloor W \rfloor + \\ + \frac{W^2 - (ssthresh)^2}{2} + N \times \frac{3}{8} W_{max}^2 \quad (24)$$

in which m is an integer satisfying the condition $2^{m-1} < ssthresh \leq 2^m, \quad m = 1, 2, \ldots$ The TCP throughput under attack is calculated accordingly by $\sum / T_a$.

3) $\frac{\mathbf{W_{max}}}{\mathbf{2}}$ **$<$ CT $<$ W:** This case is depicted in Figure 10. In this case, we have:

$$t_F = T_1 \lceil \log_2 ssthresh \rceil + T_1 (CT - ssthresh) + \\ + \frac{W^2 - (CT)^2}{2C} + N \times \alpha_2 \quad (25)$$

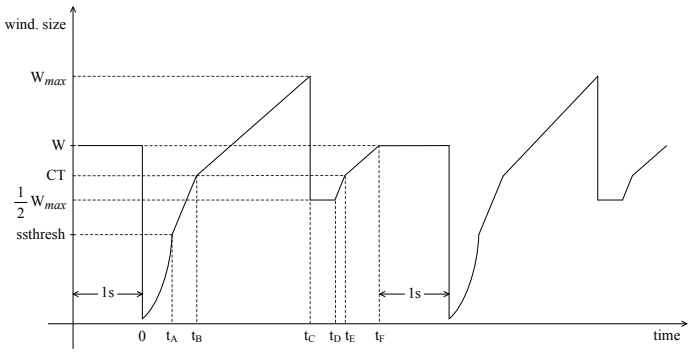

Fig. 10: Congestion window when it has been halved once before timeout and $\frac{W_{max}}{2}$ $<$ CT $<$ W.

in which $t_F = T_a - 1$ and

$$\alpha_2 = \frac{W_{max}^2 - (CT)^2}{2C} + T_1 (CT - \frac{1}{2} W_{max}) + T_{pause} \quad (26)$$

We must solve equation (25) with N from 1 to $\left\lfloor \frac{t_F}{\alpha_2} \right\rfloor$. The TCP thoughput is calculated the same as the previous case.

4) **CT $>$ W:** This case is depicted in Figure 11.

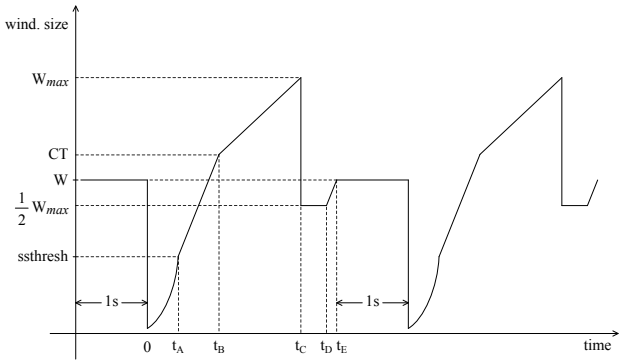

Fig. 11: Congestion window when it has been halved once before timeout and W $<$ CT.

In this case, we must have:

$$t_E = T_1 \lceil \log_2 ssthresh \rceil + T_1 (W - ssthresh) + N \times \alpha_2 \quad (27)$$

in which $t_E = T_a - 1$ and α_2 is still computed by (26). We must solve equation (27) with N from 1 to $\left\lfloor \frac{t_E}{\alpha_2} \right\rfloor$. The TCP throughput under attack is calculated the same as the two previous cases.

IV. Simulation Results

In this section, we will verify our theoretical estimation through conducting NS-2 simulations with platform taken from the address of [7]. We perform 4 attack scenarios with network topology in Figure 1 in which $N_c = 1$. This means that in all simulations, there is only one long-lived TCP

flow. The TCP flow originates from TCP-1 and terminates at Receiver and has a maximum window size of 400 packets such that its congestion window size does not ever exceed this upper bound throughout this experiment. The flow transmits packets of constant size of $M_{TCP} = 1040$ bytes (already including 40 bytes of header). The returning ACK packets are 40 bytes each. The parameters we will vary in this experiment is the network delay nt_dl and the attack cycle T_a. Low-rate DDoS attacks are created with 20 attackers (not shown in Figure 1 but can be found in our previous papers [3], [8]). We divided the attack sources into 20 groups, each group has only one source. Each attack group transmits attack packets of $M_{UDP} = 50$ bytes with the burst length $T_b = 200$ ms. If there are more than one attack flows in each attack group, the attack flows are synchronized to start and stop trasmitting packets at the same time in every burst. The aggregate burst rate of the attack is R_b and is the total burst rate of all attack flows in each attack group (note that in this experiment, there is only one attack flow in each attack group). The attack cycle T_a is the gap between the starting times of two consecutive attack groups. Attack scenario 1 has the network delay $nt\_dl = 22$ ms, attack scenario 2 has $nt\_dl = 2$ ms, and attack scenario 3 has $nt\_dl = 7$ ms. All three attack scenarios has the same attack cycle $T_a = 2$ s. Attack scenario 4 has the network delay $nt\_dl = 2$ ms but with the attack cycle $T_a = 5$ s. In our experiment, we set $nt\_bw = 100$ Mbps, $bn\_bw = 5$ Mbps, $bn\_dl = 6$ ms, the queue size at the bottleneck link $B = 50$ packets. All simulations start at time 0 and end at time 240 in which the TCP flow starts at time 20 and stops at time 240 while LDDoS attacks start at time 120 and stop at time 220 (in units of seconds). With each attack scenario, we record TCP throughput in the time period from time 160 to time 180. This time period is fully within the attack period (from time 120 to time 220) because we aim at considering TCP throughput in steady state rather than average TCP throughput over the attack period, so we have omitted the transient fluctuation of TCP throughput in the beginning of the attacks. We then compare the simulation results to our theoretical estimation in each attack scenario and as in [2], we also use relative error as a means to quantify the accuracy of our method. The relative error is calculated by:

$$\text{Relative Error} = \frac{|\,\text{Theoretical Result} - \text{Experimental Result}\,|}{\text{Experimental Result}}$$

The comparison results of TCP throughput in each scenario are shown in Figure 12. From the figure, we can see that the relative errors are smaller than 10% for all attack scenarios (the average relative error of the 4 attack scenarios is 3.7%, much smaller than 10.34% of Luo's method). Specially, the relative error is very small when the two-way propagation delay T is quite small compared to $T_a - 1$, or if the attack cyle T_a is large. This confirms that our estimation method is more accurate than Luo's method in some specific cases. Aside from, it is simpler as well because to find the value of W, we only have to solve one equation with one variable instead of solving a system of two equations with two variables as in [2]. Although we have just investigated only a few scenarios above, we believe that the accuracy of our estimation method still remains high when more experiments are conducted in the future.

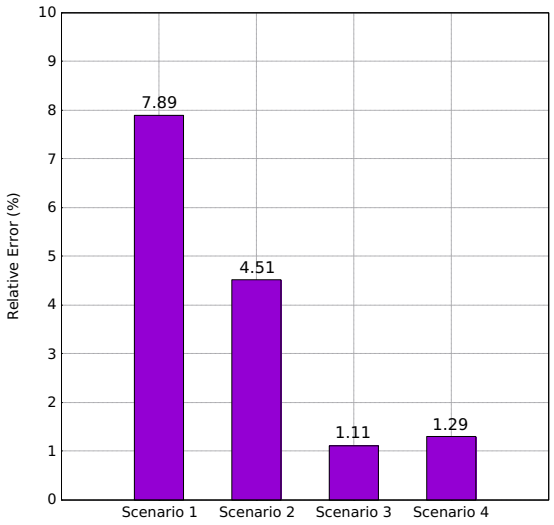

Fig. 12: Comparison results with one TCP flow.

V. Conclusion

In this paper, we have proposed a method for estimation of TCP throughput of a single TCP flow under low-rate DDoS attacks. Our estimation is quite simple but it can estimate TCP throughput with high accuracy. We have conducted somes NS-2 simulations with a specific network topology and the simulation results show that the relative error of our method is small under a wide diversity of round trip time of the TCP flow as well as under a change of the attack cycle T_a. We plan to conduct more experiments with this case and then explore the cases with homogeneous and heterogeneous TCP flows in the future.

References

[1] A. Kuzmanovic and E. Knightly, "Low-Rate TCP-Targeted Denial of Service Attacks (The Shrew vs. the Mice and Elephants)," in *Proceedings of ACM SIGCOMM*, 2003, pp. 75–86.

[2] J. Luo, X. Yang, J. Wang, J. Xu, J. Sun, and K. Long, "On a Mathematical Model for Low-Rate Shrew DDoS," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 7, pp. 1069–1083, 2014.

[3] M. Kieu, D. Nguyen, and T. Nguyen, "Using CPR Metric to Detect and Filter Low-Rate DDoS Flows," in *Proceedings of ACM SoICT*, 2017, pp. 325–332.

[4] S. Shenker, L. Zhang, and D. Clark, "Some observations on the dynamics of a congestion control algorithm," *ACM Computer Communication Review*, vol. 20, no. 5, pp. 30–39, 1990.

[5] W. Stevens, *TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms*, 1997, RFC 2001.

[6] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing router buffers," *ACM Computer Communication Review*, vol. 34, no. 4, pp. 281–292, 2004.

[7] "AQM&DoS simulation platform," https://sites.google.com/site/cwzhangres/home/posts/aqmdossimulationplatform/, 2011.

[8] M. Kieu, D. Nguyen, and T. Nguyen, "Techniques for Improving Performance of the CPR-Based Approach," in *Proceedings of ACM SoICT*, 2018, pp. 163–168.