On the Impact of Low-Rate Attacks

MINA GUIRGUIS AZER BESTAVROS IBRAHIM MATTA
{msg, best, matta}@cs.bu.edu
Computer Science Department
Boston University
Boston, MA 02215, USA

***Abstract*— Recent research have exposed new breeds of attacks that are capable of denying service or inflicting significant damage for TCP flows, without sustaining the attack traffic. Such attacks are often referred to as “low-rate” attacks and they stand in sharp contrast against traditional Denial of Service (DoS) attacks that can completely shut off TCP flows by flooding an Internet link. In this paper, we study the impact of these new breeds of attacks and the extent to which defense mechanisms are capable of mitigating the attack’s impact. Through adopting a simple discrete-time model with a single TCP flow and a non-oblivious adversary, we were able to expose new variants of these low-rate attacks that could potentially have high attack potency per attack burst. Our analysis is focused towards worst-case scenarios, thus our results should be regarded as upper bounds on the impact of low-rate attacks rather than a real assessment under a specific attack scenario.**

***Index Terms*— Security; TCP; Denial of Service; Low-Rate Attacks;**

I. INTRODUCTION

Denial of Service (DoS) attacks and Distributed Denial of Service (DDoS) attacks present an ongoing threat to almost every Internet Service. An adversary bent on limiting access to a network can bring down an Internet service by subjecting it to sustained levels of demand that far exceed its capacity, making that service incapable of adequately responding to legitimate requests. Although DoS attacks can inflict significant damage, preparing a DoS attack requires some additional work. In particular, it requires recruiting enough zombie clients to launch the attack. These zombie clients are typically compromised computers scattered all over the Internet. Moreover, just by their brute force nature, these attacks are easily exposed, making it possible for appropriate countermeasures to be taken once attacks are detected.[1]

Only recently, new breeds of attacks have been exposed that could deny service or degrade the quality without sustaining attack traffic [1], [2], [3], [4]. These new attacks are often referred to as “low-rate” attacks and they require relatively lower number of zombie clients in comparison to the brute force nature of traditional DoS attacks. In addition, by their nature, they are harder to detect. Thus, low-rate attacks present new challenges for defense mechanisms in terms of detection, protection and taking the appropriate countermeasures. And indeed, recent research started to address and to develop new defense mechanisms against low-rate attacks [5], [6].

[1]Since sometimes DoS attacks could be anticipated, countermeasures can be taken *before* attacks are launched.

Motivation: While detection and taking countermeasures are important problems worth tackling, a more important problem is to *assess the damage that could be inflicted by those new breeds of attacks and the extent to which defense mechanisms are capable of mitigating the attack’s impact*. To that end, this paper focuses on quantifying the damage that could be inflicted by previously exposed low-rate attacks, in addition to exposing new variants of such attacks. Towards constructive goals of our findings, we study the extent to which different defense mechanisms would be capable of mitigating the attack’s impact. We also propose a new defense mechanism that would lessen the impact of low-rate attacks.

Currently, most of the Internet traffic is carried by TCP [7]. TCP relies on feedback mechanisms to adapt its sending rate to match its “fair share” of network resources. When a resource gets congested, it drops packets. These drops are regarded by TCP as a congestion signal and it reacts to them by halving its sending rate. If no packets are being dropped by the network, TCP increases its window by one packet every Round Trip Time (RTT). Such Additive Increase Multiplicative Decrease (AIMD) mechanism serves to utilize available bandwidth when present and to alleviate congestion when it does happen. When a TCP flow is faced with high packet loss, it could potentially incur a timeout, due to either the lack of enough duplicate acknowledgments to trigger packet retransmission or a loss of a whole window of packets.

While the mechanisms employed in TCP are essential for convergence to fairness and efficiency [8], they make TCP vulnerable to the presence of packet loss that are not directly attributed to legitimate congestion. In particular, it has been shown that an adversary mounting a shrew attack can exploit the TCP timeout mechanism, causing TCP flows to continually timeout [1]. Also, an adversary mounting RoQ attacks can exploit the adaptive AIMD mechanism in TCP, causing significant degradation of quality [2].

A complete assessment of the impact of low-rate attack would be difficult to achieve in practice, due to the high number of parameters involved, such as link capacity, buffer size, round-trip time, among others. That is, in addition to other factors such as the TCP implementation used (Tahoe [7], Reno [9], NewReno [10] and SACK [11]) and the queue management implementation used (RED [12], and all other AQM schemes). The capabilities of the attacker should also be taken into account, such as the availability of any probing techniques for bandwidth and queuing delay estimation. Other non-deterministic factors are typically involved, due to timing the attack traffic, synchronization and the use of randomization. Fortunately, to derive worst-case analysis, some of the above factors could be abstracted away. For example, by assuming that lost packets belong to the victim as opposed to the attacker, one can abstract away the buffering implementation. Clearly, this doesn't usually happen in practice, but it enables us to derive close to worst-case analysis. [2] So in general, our results in this paper should be regarded as close to upper bounds rather than a real assessment of the impact of low-rate attacks. Also, our results give a relative comparison for different variants of low-rate attacks.

Paper Outline: Section II presents our discrete-time model, where we study the behavior of a single TCP flow traversing a bottleneck link. In Section III, we quantify the maximum damage inflicted by different low-rate attacks, in addition to exposing new variants of such attacks. In Section IV, we assess the capability of different defense mechanisms in mitigating the ill effect of low-rate attacks, in addition to proposing new defense mechanisms. In Section V, we briefly discuss related work, noting that throughout this paper, we point to various pieces of research work as appropriate. We conclude in Section VI with a summary and future directions.

II. Model

A. Model Derivation

We will start with a simple model that represents a single TCP connection, traversing a single bottleneck link with a fixed capacity C and a buffer size B. Our model is a modified version of those presented in [14] and [15]. In particular, we assume the link capacity is fixed over time and we introduce the buffering process. We model the slow-start, the Additive Increase Multiplicative Decrease (AIMD) and the retransmission timeout mechanism of TCP. We summarize our proposed model next.

Let time be divided into slots, where each slot is indexed by the variable i. All slots have the same time duration which we chose to be equal to the connection's round-trip time R. Let w_i represents the window size chosen by the algorithm to be sent during time slot i.[3] Typically, w_i follows the AIMD mechanism of TCP, except for the beginning of the TCP connection or following a timeout, where w_i evolves according to the slow-start mechanism. If the algorithm sends less packets than C (given in packets per time slot), all of them will be transmitted during the same time slot. If however, w_i exceeds C, some of the extra packets will be buffered to be transmitted during the next time slot, if buffer space permits.

We let b_i denote the buffer occupancy at time slot i. The buffer at any time instant is equal to the buffer occupancy during the previous time slot, plus the difference between the input and the output during that time slot, thus it can be described by the following equation:

$$b_{i+1} = b_i + w_{i+1} - C \qquad (1)$$

Equation (1) is bounded by 0 from below and by B from above.

Once a TCP connection starts or following a time-out, the window typically starts with 1 packet and grows exponentially as dictated by the slow-start mechanism. Once the congestion window size reaches the slow start threshold, s, the TCP connection switches to the AIMD mechanism.

According to the AIMD mechanism, the window size is incremented by 1 packet on successful packet transmission during the previous time slot (round-trip time) and is halved on a packet loss, thus can be described by the following equation:

$$w_{i+1} = \begin{cases} w_i + 1 & b_i \leq B \\ \frac{w_i}{2^{n_i}} & \end{cases} \qquad (2)$$

where n_i is the number of packets lost during time slot i. This equation assumes that every packet lost results in halving the window once. [4]

Since the behavior of a single TCP connection is typically dictated by the AIMD mechanism, we ignore, for now, the effect of the single slow-start that happens when the connection starts and we focus on its steady-state behavior. We assume that at time step i, the window size w_i equals C and the buffer size b_i equals 0. Thus, the additive increase component will keep filling the buffer until a packet loss occurs. Let l denotes the time slot when a loss happens. Since the buffer is acting as an integrator[5], l can be simply derived to be:[6]

$$l \approx \sqrt{2B} + 1 \qquad (3)$$

Thus, at time slot l, w_l is equal to $C + l - 1$ and b_l is equal to B.

After a packet loss and halving the window size and depending on the buffer size, B, the input may not be able to sustain the full link capacity C. According to equation (1), the buffer size at time $l+1$ is equal to:

$$b_{l+1} = B + \frac{C + l - 1}{2} - C \qquad (4)$$

If the input rate cannot sustain the link capacity, the buffer will start draining until it hits 0. To derive the number of time steps it takes to drain the buffer, we solve for d, such that, b_{l+d} is 0:[7]

$$d \approx \frac{2B}{C - l} \qquad (5)$$

The window size at time slot $l + d$, is equal to :

[2] As a matter of fact, this could happen in practice, if the router is employing a DiffServ architecture [13], where the attack traffic has a higher priority.

[3] Throughout this paper, w_i should be regarded as the congestion window dictated by the bottleneck link and not constrained by the receiver's window size.

[4] Notice that the window size cannot go below 1, thus $\log_2 w_i$ is an upper bound on the effective number of lost packets.

[5] According to (1), the buffer occupancy grows by 1,2,3,... packets at successive time slots until the buffer becomes full.

[6] This is a solution of a second-order equation under the assumption that $2B \gg 1$, which holds true for any reasonably chosen buffer size.

[7] We assume that the buffer drains before the window exceeds the capacity, and during this short interval, the window size remains a constant.

$$w_{l+d} = \frac{C+l-1}{2} + d - 1 \quad (6)$$

Let the time steps it takes for the window to reach from w_{l+d} back to C be denoted by k, and is given by:

$$k = C - w_{l+d} \quad (7)$$

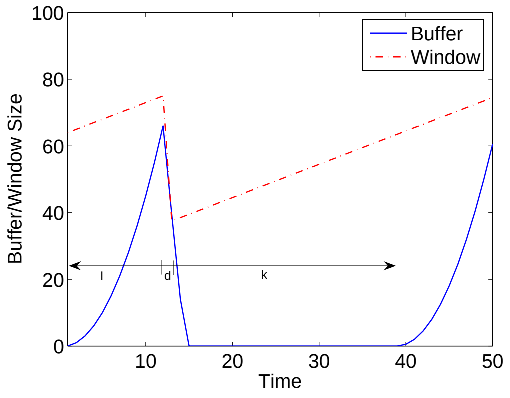

Fig. 1. Buffer process and window evolution via the AIMD mechanism; The buffer size, B, is 64 packets and the capacity, C is 64 packets.

Thus to summarize, the window and buffer behaviors can be described into three main regions. The first region starts with the connection's throughput equal to the C and the buffer is empty and ends with a packet loss and a full buffer. The second region is the period of time until the buffer drains, thus it ends with an empty buffer size. The third region, is similar to the first one, except that the buffer size will remain empty during which and it ends with the window size equal to $C-1$ and the three regions repeat. Figure 1 depicts the buffering process and the window evolution, obtained from a numerical solution. We chose B and C to be 64 packets.

B. Model Assumptions

The simple model outlined above makes the following assumptions:

(1) The connection's round-trip time, R, is a constant and doesn't change with the queuing delay at the bottleneck. In reality, the round-trip time is the sum of the propagation delay and queuing delay. This assumption has two related implications; First, the queue size behaves as a parabolic integrator rather than linear or sub-linear [16]. Second, the rule-of-thumb of the buffer size being the bandwidth-delay product of the network would not prevent buffer draining as shown in Figure 1.

(2) The effect of self-clocking is not pronounced between time slots, so the TCP connection receives its feedback over a specific slot and not across multiple slots, for the same window of packets sent.

As we argued before, these assumptions will not typically hold in practice, however, they tend to give a tractable model to work with for close to worst-case analysis.

C. No Attack Case

In order to study the impact of low-rate attacks, we define the following two performance metrics that would be our indicators for assessment:

Gain: The gain, G, of a TCP algorithm is defined as the amount of packets transmitted successfully over the link.

Loss: The loss, L, of a TCP algorithm is defined as the amount of packets that could have been sent successfully, but where not transmitted. For example the loss in slot i is the difference between w_i and C when w_i is less than C.[8]

When there is no attack underway, and ignoring the effect of the first slow start, TCP's gain over its congestion epoch, is given by:

$$\begin{aligned} G &= \sum_{i=1}^{l+d+k} w_i \\ &= \frac{l+d+k}{2}(2l+d+k+C) \quad (8) \end{aligned}$$

As Figure 1 suggests, the only period when the link is under utilized occurs throughout the third region, when the buffer is 0. If we consider the time slot, $l+d+k$, the window size at this time slot is equal to $C-1$. Thus a waste of 1 packet exists. Carrying our computation backwards, at time slot $(l+d+k)-1$, the waste was 2 packets and so on. The total loss is given by:

$$L = \sum_{i=l+d+1}^{l+d+k} (C - w_i) = \frac{k}{2}(k+1) \quad (9)$$

Notice that this loss is attributed directly to the TCP's AIMD mechanism as opposed to the presence of any attack traffic.

III. Low-Rate Attacks

In this section, we study the impact of low-rate attacks on a single TCP flow. We assume the attacker is non-oblivious, i.e. it knows *when* and *how much* traffic to send in order to cause the maximum damage or the maximum damage per attack byte. We also assume that the lost packets belong to the legitimate connection as opposed to the attacker. Such pessimistic assumptions enable us to drive close to upper bounds on the impact of low-rate attacks.

To assess the vulnerability of low-rate exploits, we follow the definition of *attack potency*, π, we proposed in [2], whereby the potency of an attack is the ratio of the *damage* caused by the attack to the attacker's *cost* for mounting the attack.[9]

$$\pi = \frac{\text{Damage}}{\text{Cost}} \quad (10)$$

[8]Throughout this paper, the loss computed would include the inefficiencies resulting from the TCP AIMD mechanism.

[9]The definitions in [2] allow for an aggressiveness index Ω, which we take to be 1.

A. Shrew Attacks

The “Shrew” attack, proposed in [1], is an example of a low-rate attack that targets a subset of connections going through a bottleneck link, with the intention of shutting them off, through exploiting the timeout mechanism. The timeout mechanism is employed in TCP to alleviate severe congestion, by preventing the connection from sending any packets for longer periods of time. In particular, if TCP fails to receive enough duplicate acknowledgments to trigger packet transmission, it will incur a timeout. During timeout, TCP reduces its window to 1 packet and does not send any packets for a period of time known as the Retransmission Time Out (RTO). It is recommended to have a base RTO of 1 second [17]. The shrew attack exploits the homogeneity in the timeout mechanism in TCP through synchronizing the attack traffic with the recommended base RTO, causing connections to continually timeout.

To account for timeout in our model, we let α denote the number of slots that represents the base RTO, i.e., $base\ RTO = \alpha R$ We ignore the effect of exponential back-off of the RTO on successive failures of a timeout triggered loss. We refer the reader to [18] for additional information on the timeout mechanism. We also assume that timeout occurs due to the loss of a whole window of packets as opposed to lack of enough duplicate acknowledgments. Following a timeout, the TCP connection would do slow start until the slow-start threshold, s, is reached and then the TCP AIMD mechanism is used.

The attack traffic is synchronized in such a way that it hits the connection whenever it is about to exit from timeout, i.e., when its window size is 1 packet. In order to cause the loss of this packet, the attacker has to send enough attack traffic, that would both, fill the buffer and saturate the link capacity, in a single round-trip time. Thus the total attack traffic is simply $C + B$. The damage caused by such a burst is total waste of bandwidth of C for the duration of the timeout α. Thus the connection was unable to send any packet.

If we instantiate the attack potency, π, as defined in [2] as the damage in packets per unit attack packet, we get an upper bound on the impact of a classic shrew attack:

$$\pi_{shrew} = \frac{\alpha C}{C + B} \tag{11}$$

Notice that the attack here is repeated every α time slots and in the worst case, it causes a complete denial of service since the connection couldn’t send any packet across the bottleneck. Thus, $G_{shrew} = 0$ and $L_{shrew} = \alpha C$.

B. More Potent Shrew Attacks

While the above shrew attack can completely shut off TCP connections, causing the maximum absolute damage, it doesn’t maximize the damage per unit attack byte for each attack burst. To illustrate this point, we expose two variants of the shrew attack.

Shrew Attack at Saturation ($shrew^{\dagger}$): Following a timeout, the attacker would wait until the window ramps up again to reach C, as opposed to attacking when the window is 1 packet. Such wait has the advantage of keeping the connection paying the price of under-utilization until it reaches C. The total under-utilization before reaching a window of size C is composed of two parts; the first part is due to slow-start and the second part is due to the AIMD mechanism. Assuming that the slow-start threshold, s, is set to $\frac{C+\sqrt{2B}}{2}$, it would take $\log s$ for the window to grow from 1 to s. Once the window reaches the slow-start threshold, the AIMD mechanism will be used until the window reaches C. Thus, the gain is given by:

$$G_{shrew^{\dagger}} = \sum_{i=0}^{\log s} 2^i + (C - s + 1)(\frac{C - s + 2}{2} + s) \tag{12}$$

The total under-utilization from such attack is given by:

$$\begin{aligned} L_{shrew^{\dagger}} &= (C \log s) - \sum_{i=0}^{\log s} 2^i \\ &+ (C - s + 1)\frac{C - s + 2}{2} + \alpha C \end{aligned} \tag{13}$$

The attack here is repeated every $\log s + (C - s + 1) + \alpha$ time slots. Figure 2 represents the the under-utilization due to a single attack burst.

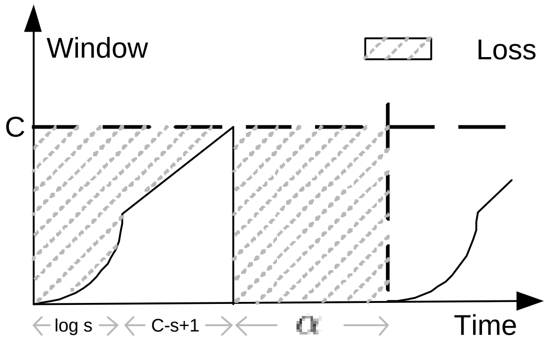

Fig. 2. Shrew attack at saturation; shaded area represents the under-utilization due to a single attack burst. The attack is repeated every $\log s+(C-s+1)+\alpha$ time slots.

In order to cause a loss of C packets, that attacker has to saturate the link and fill the buffer, injecting $C + B$ packets in one time slot. Thus the potency of such attack is given by:

$$\pi_{shrew^{\dagger}} = \frac{L_{shrew^{\dagger}}}{C + B} \tag{14}$$

Shrew Attack at Full Buffer ($shrew^{\ddagger}$): In this exposed attack and following a timeout, the attacker would wait until the buffer is full. This has the advantage of reducing the cost of the attack. In particular, the attacker now only needs to send C packets to cause the loss of the current window of packets. Thus the gain from this attack is:

$$
\begin{aligned}
G_{shrew^{\ddagger}} &= \sum_{i=0}^{\log s} 2^i + (C-s+1)(\frac{C-s+2}{2}+s) \\
&+ \frac{\sqrt{2B}}{2}(\sqrt{2B}+1) + C\sqrt{2B} \qquad (15)
\end{aligned}
$$

and the loss is:

$$
\begin{aligned}
L_{shrew^{\ddagger}} &= (C\log s) - \sum_{i=0}^{\log s} 2^i \\
&+ (C-s+1)\frac{C-s+2}{2} + \alpha C \qquad (16)
\end{aligned}
$$

and the potency is:

$$
\pi_{shrew^{\ddagger}} = \frac{L_{shrew^{\ddagger}}}{C} \qquad (17)
$$

This attack extends the attack period to be repeated every $\log s + (C-s+1) + l + \alpha$ time slots. Figure 3 represents the the under-utilization due to a single attack burst. Similar to the $Shrew^{\dagger}$ case, extending the attack period, reduces the attacker's rate since it waits for a longer period between two successive attack bursts.

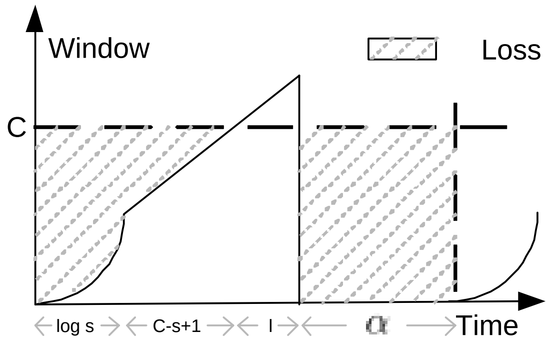

Fig. 3. Shrew attack at full buffer; shaded area represents the under-utilization due to a single attack burst. The attack is repeated every $\log s + (C-s+1) + l + \alpha$ time slots.

Notice that per attack burst, it is easy to see that $\pi_{shrew^{\ddagger}} > \pi_{shrew^{\dagger}} > \pi_{shrew}$. However, since each attack has a different period, over longer time scales (multiples of attack periods), such ordering of different potencies could be different based on the parameters.

Figure 4 illustrates this point. We plot the normalized potency (calculated as the potency divided by the attack period) on the Y-axis. We fixed the capacity to 500 packets per slot and we vary the buffer size from 100 packets to 3000 packets, on the X-axis. One can see that the $shrew$ is always more potent than $shrew^{\dagger}$. However, $shrew^{\ddagger}$ can be more potent once the buffer exceeds a certain threshold (around 700, in our example). Once the buffer size is large, it gets harder for the attacker to fill the buffer and since $shrew^{\ddagger}$ doesn't try to fill the buffer, since it relies on the TCP mechanism to achieve this task, it becomes more potent at large buffer sizes.

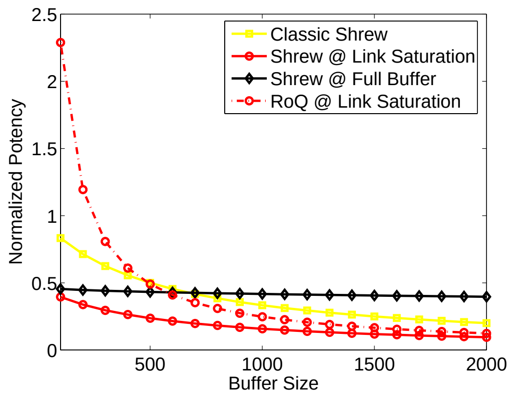

Fig. 4. Normalized Potency represented on the Y-axis versus the buffer size represented on the X-axis, for different attack variants.

C. Reduction of Quality Attacks

RoQ attacks [2] exploit the AIMD adaptation mechanism to maximize the damage per unit attack traffic, i.e., the attack potency. A worst-case RoQ attack would force multiple losses causing the window to drop from its current value to 1 packet. Since the connection did not timeout, the window will ramp up again through the AIMD mechanism, i.e., without the slow-start mechanism. We illustrate the following two variants of RoQ exploits.

RoQ Attack at Saturation ($RoQ^{\dagger}$): A worst-case RoQ attack, carried when the window is C packets, would cause multiple losses causing the window to drop from C to 1 packet. To cause such a damage, the attacker must fill the buffer and inject additional $\log_2 C$ packets, that would cause the window to drop from C to 1 packet. The attack is repeated every C time slots. The TCP gain achieved under this attack is:

$$
G_{RoQ^{\dagger}} = \frac{C}{2}(C+1) \qquad (18)
$$

and the maximum loss is given by:

$$
L_{RoQ^{\dagger}} = \frac{C}{2}(C-1) \qquad (19)
$$

Thus the RoQ attack potency is upper bounded by:

$$
\pi_{RoQ^{\dagger}} = \frac{L_{RoQ^{\dagger}}}{B+\log_2 C} \qquad (20)
$$

Figure 5 represents the the under-utilization due to a single attack burst.

RoQ Attack at Full Buffer ($RoQ^{\ddagger}$): If the RoQ attack is carried at a full buffer, then the attacker would only need to inject enough packets to drop the window from $C+\sqrt{2B}$ to 1 packet, that is $\log_2 C + \sqrt{2B}$. The attack is repeated every $C+l$ time slots. The TCP gain under this attack is:

$$
G_{RoQ^{\ddagger}} = \frac{C}{2}(C+1) + \frac{\sqrt{2B}}{2}(\sqrt{2B}+1) + C\sqrt{2B} \qquad (21)
$$

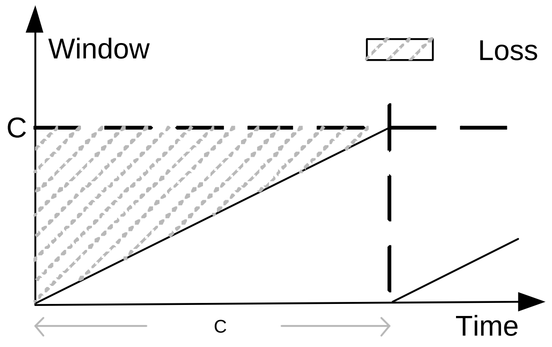

Fig. 5. RoQ attack at link saturation; shaded area represents the under-utilization due to a single attack burst. The attack is repeated every C time slots.

and the maximum loss is given by:

$$L_{RoQ^{\ddagger}} = \frac{C}{2}(C-1) \tag{22}$$

Thus the RoQ attack potency is upper bounded by:

$$\pi_{RoQ^{\ddagger}} = \frac{L_{RoQ^{\ddagger}}}{\log_2{(C+\sqrt{2B})}} \tag{23}$$

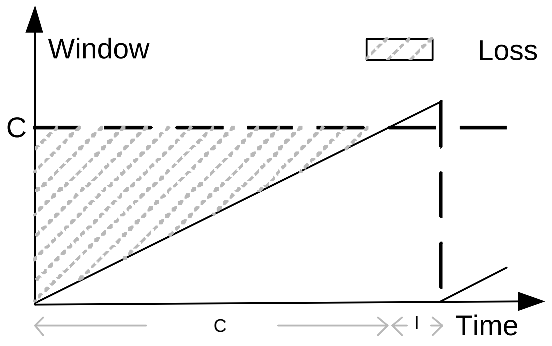

Fig. 6. RoQ attack at full buffer; shaded area represents the under-utilization due to a single attack burst. The attack is repeated every $C+l$ time slots.

Figure 6 represents the the under-utilization due to a single attack burst.

Figure 4 shows the normalized potency for $RoQ^{\dagger}$. One can see that it has higher potency (than all shrew attacks) at lower buffer sizes. Once the buffer size becomes large, it gets harder for the attacker to fill the buffer and the potency would start decreasing. $RoQ^{\ddagger}$ was able to achieve more than 10 times the normalized potency in comparison to all other attacks, hence we didn't include it in Figure 4.

IV. Defenses against Low-Rate Attacks

In this section, we evaluate the effect of defense mechanisms that could mitigate the impact of low-rate attacks. RTO randomization, proposed in [1] and studied in [6] is evaluated. We then we propose a new defense mechanism.

A. RTO Randomization

A natural defense against the Shrew attacks [1], is to randomize the base RTO. Such randomization would prevent the attacker from synchronizing its attack burst at the right time, causing a complete denial of service for the TCP flows. The main drawback of this approach is that TCP connections still timeout; *i.e.*, the damage is done. Clearly, the damage is less than that when the base RTO was fixed to 1 second, since connections now can utilize some of the bandwidth available between attack bursts. As we will demonstrate, the achievable throughput depends on the round-trip time. TCP connections with shorter round-trip time would benefit from RTO randomization and can recover between attack bursts. On the contrary, connections with long round-trip time would not be able to open up their windows and will not achieve much throughput before the next attack burst comes.

To study the effect of RTO randomization, we assume that once a connection times out, it will choose its RTO timeout uniformly at random between two values, RTO_{min} and RTO_{max}, where the average RTO is preserved to 1 second. For example, it can use a random RTO value between 0.5 and 1.5 seconds.

We consider the attacker sending its attack burst every 1 second. After the first timeout, the connection, and depending on its random RTO value, will commence in the slow start phase and would be able to utilize the time until the next attack burst when this cycle repeats. If we let RTO_r denotes the random value chosen, then we have the following three cases:

(1) RTO_r falls in the same attacked time slot: In this case, the connection will timeout again and the gain is 0.

(2) RTO_r less than 1 second: In this case, the number of RTTs that can be utilized by the connection is $\approx \frac{1-RTO_r}{R}$.

(3) RTO_r is larger than 1 second: In this case, the number of RTTs that can be utilized by the connection is $\approx \frac{2-RTO_r}{R}$.

One can easily observe that if R is small, short RTT connections would have many slots to ramp up. However, if R is large, long RTT connections would have a few number of slots to ramp up and may not even leave slow start before they got hit again.

Figure 7 illustrates the impact of RTO randomization for 3 different round-trip connections, where every time slot represents the round-trip time. Figure 7(left) shows the behavior of a TCP connection with a 100 msec round-trip time, the attacker sends its burst every one second (that is every 10 slots). Figure 7(middle) shows the behavior of a TCP connection with a 10 msec round-trip time, the attacker sends its burst every one second (that is every 100 slots). Figure 7(right) shows the behavior of a TCP connection with a 1 msec round-trip time, the attacker sends its burst every one second (that is every 1000 slots). RTO randomization was fixed between 0.5 to 1.5 seconds.

We ran each experiment for 1000 seconds and Table I shows the ratio between the gain obtained by TCP divided by the maximum throughput that could be achieved. Connections with $R = 100$ msec are able to achieve only 3% of the

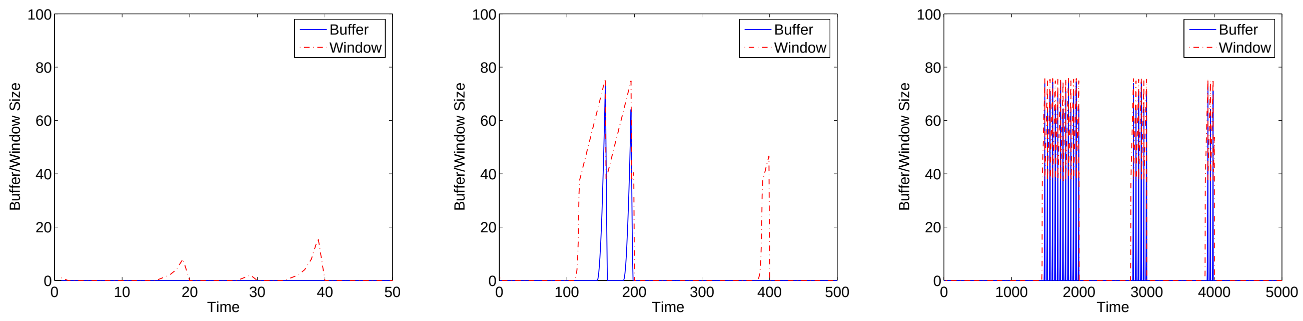

Fig. 7. Assessment of RTO randomization for connections with different round-trip times. Left, middle and right plots represent the throughput of a TCP connection with 100 msec, 10 msec and 1 msec round-trip time, respectively.

Round-trip Time (R)	**Gain Ratio**
R = 100 msec	0.03
R = 10 msec	0.18
R = 1 msec	0.22

TABLE I

GAIN RATIO FOR DIFFERENT ROUND-TRIP TIME TCP CONNECTIONS; GAIN RATIO = $\frac{Gain}{Maximum\ Gain}$

available bandwidth. This suggests that RTO randomization is not the right solution for long RTT connections.

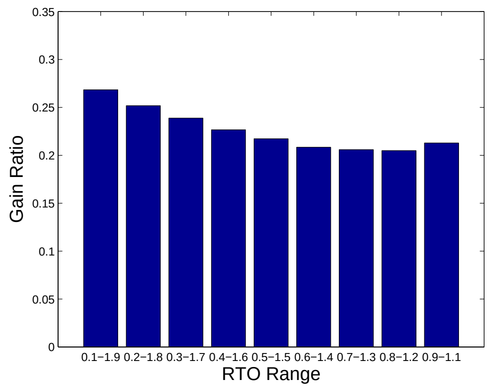

Fig. 8. Impact of different ranges of randomization, for a TCP connection with 1 msec RTT under a periodic attack.

Figure 8 studies the impact of different ranges of randomization, for connections with 1 msec RTT. Notice that the analysis above is close to the one presented in [6], except that we do not assume that the connections will utilize the full bandwidth before the next timeout, but will follow slow-start and AIMD subject to their RTT.

B. An Outline for a Defense Mechanism

One of the reasons that makes TCP vulnerable against low-rate attacks is that it constantly tries to send packets. However, if TCP can infer that an attack burst would happen at a certain RTT slot, it can stop sending any packets during that time slot and hence would not experience any packet loss[10]. Of course such information is not available to current versions of TCP. We are currently working on a modified version of TCP that is capable of probing the network's state and identifying the attack's periodicity by observing its own packet delays. Once the attack's period is identified, a TCP connection would not send any packets when the attack traffic is expected. The exact details of the modified TCP is outside the scope of this paper.

C. RTO Randomization vs. Randomized Attacker

The implication of the above solution is that attackers can now randomize the time between attack bursts to hinder the period identification process. We show that such randomization lessens the impact of the attack, even with RTO randomization in place.

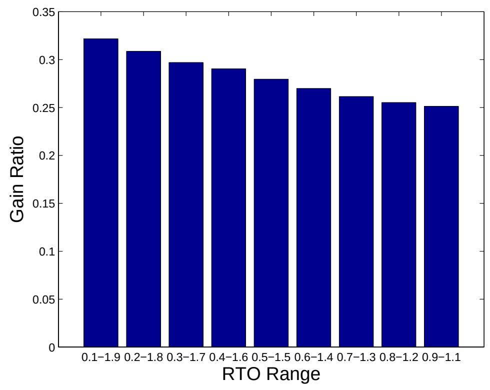

Fig. 9. Impact of different ranges of randomization, for a TCP connection with 1 msec RTT under a randomized attack period.

Figure 9 shows the gain ratio over different ranges of RTO randomization. The attacker here was not periodic with the attack period selected uniformly at random with an average of 1 sec. Comparing this figure to Figure 8, one can see that randomizing the attack bursts actually hurts the attacker. When the attacker selects the inter-burst period at random, two bursts could be either close to each other or far from each other. In the first case, the second burst would not cause any harm to the

[10]Due to error in measurements, it may not send any packets for multiple time slots, around the one being identified.

connections that are already in timeout. In the second case, two attack bursts that are far from each other, would cause TCP to utilize the time in between to ramp up its bandwidth usage.

V. Related Work

This work is inspired by the work done in [14] and [15], where the authors studied the congestion control problem through an optimization framework. In particular, they studied the performance of different probing mechanisms in utilizing the available bandwidth where the available bandwidth is chosen by an adversary, with limited power. The problem was casted as an online algorithm with an eye on the competitive ratio, defined as the performance/penalty of an algorithm, in comparison to the off-line version that knows the available bandwidth in advance. The work presented here is different in many ways. First, our model is more TCP specific, with AIMD, slow-start and timeout mechanisms employed. We did not consider other probing algorithms in comparison to TCP. Moreover, we focused on studying the impact of low-rate attacks, such as the shrew attacks [1] and RoQ attacks [2], through the potency metrics.

We have also studied RTO randomization as a defense mechanism, first proposed in [1] and evaluated in [6]. The approach in [6] was an optimistic one, assuming TCP would be able to utilize the full bandwidth before the next attack burst. The approach we followed here is different. We studied the behavior of a single TCP flow where bandwidth is utilized by TCP mechanisms (slow-start and AIMD) and we concentrated on worst-case analysis.

VI. Conclusion

In this paper, we have focused on studying the impact of a larger family of low-rate attacks on a single TCP flow. We have carried our studies through the notion of attack potency, that describes the trade-offs between damage inflicted and the cost involved to mount the attack. Despite that some of our assumptions would not likely hold in a typical setting, our results, however, tend to give upper bounds on the worst-case impact of such attacks. We believe that recently exposed low-rate attacks, in addition to the ones exposed here, present new challenges for defense mechanisms in order to mitigate the attack's impact.

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*, karlsruhe, Germany, August 2003.

[2] M. Guirguis, A. Bestavros, and I. Matta, "Exploiting the Transients of Adaptation for RoQ Attacks on Internet Resources," in *Proceedings of ICNP'04: The 12th IEEE International Conference on Network Protocols*, Berlin, Germany, October 2004.

[3] M. Guirguis, A. Bestavros, and I. Matta, "Bandwidth Stealing via Link Targeted RoQ Attacks," in *Proceedings of CCN'04: The 2nd IASTED International Conference on Communication and Computer Networks*, Cambridge, MA, November 2004.

[4] M. Guirguis, A. Bestavros, I. Matta, and Y. Zhang, "Reduction of Quality (RoQ) Attacks on Internet End-Systems," in *Proceedings of Infocom'05: The IEEE International Conference on Computer Communication*, Miami, FL, March 2005.

[5] H. Sun, J. Lui, and D. Yau, "Defending Against Low-Rate TCP Attacks: Dynamic Detection and Protection," in *Proceedings of ICNP'04: The 12th IEEE International Conference on Network Protocols*, Berlin, Germany, October 2004.

[6] G. Yang, M. Gerla, and M. Sanadidi, "Defense against Low-rate TCP-targeted Denial-of-Service Attacks," in *Proceedings of ISCC: The 9th IEEE Symposium on Computers and Communications*, Alexandria, Egypt, June 2004.

[7] V. Jacobson, "Congestion Avoidance and Control," in *Proceedings of ACM SIGCOMM*, Standford, CA, August 1988.

[8] D. Chiu and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Computer Networks and ISDN Systems*, 1989.

[9] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," *RFC 2581*, April 1999.

[10] S. Floyd and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm," *RFC 2582*, April 1999.

[11] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options," *RFC 2018*, October 1996.

[12] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *Transactions on Networking*, August 1993.

[13] S. Blake, D. Black, M. Carlson, E. Davies, Z.Wang, and W. Weiss, "An Architecture for Differentiated Services," *IETF RFC 2475*, December 1998.

[14] R. Karp, E. Koutsoupias, C. Papadimitriou, and S. Shenker, "Optimization Problems in Congestion Control," in *Proceedings of IEEE Symposium on Foundations of Computer Science*, Redondo Beach, CA, November 2000.

[15] S. Arora and B. Brinkman, "A Randomized Online Algorithm for Bandwidth Utilization," in *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, CA, January 2002.

[16] G. Appenzeller, I. Keslassy, and N. McKeown, "Sizing Router Buffers," in *Proceedings of ACM SIGCOMM*, Portland, OR, September 2004.

[17] M. Allman and V. Paxson, "On Estimating End-to-End Network Path Properties," in *Proceedings of ACM SIGCOMM*, Cambridge, MA, August 1999.

[18] V. Paxson and M. Allman, "Computing TCP's retransmission Timer," *RFC 2988*, November 2000.