

# A Novel Tuneable Low-Intensity Adversarial Attack

Salil S. Kanhere

School of Computer Science and Engineering  
University of New South Wales  
Sydney, Australia  
salilk@cse.unsw.edu.au

Anjum Naveed

School of Computer Science and Engineering  
University of New South Wales  
Sydney, Australia  
anaveed@cse.unsw.edu.au

**Abstract**—Currently, Denial of Service (DoS) attacks remain amongst the most critical threats to Internet applications. The goal of the attacker in a DoS attack is to overwhelm a shared resource by sending a large amount of traffic thus, rendering the resource unavailable to other legitimate users. In this paper, we expose a novel contrasting category of attacks that is aimed at exploiting the adaptive behavior exhibited by several network and system protocols such as TCP. The goal of the attacker in this case is not to entirely disable the service but to inflict sufficient degradation to the service quality experienced by legitimate users. An important property of these attacks is the fact that the desired adversarial impact can be achieved by using a non-suspicious low-rate attack stream, which can easily evade detection. Further by *tuning* various parameters of the attack traffic stream, the attacker can inflict varying degrees of service degradation and at the same time making it extremely difficult for the victim to detect attacker presence. Our simulation based experiments validate our observations and demonstrate that an attacker can significantly degrade the performance of the TCP flows by inducing low-rate attack traffic which is co-ordinated to exploit the congestion control behavior of TCP.

## I. INTRODUCTION

The connectionless nature of Internet partially explains its success, network being easily accessible for millions of people in the world. However, the flip side of this success is Internet's high vulnerability to malicious attacks. Over the past few years, Denial of Service (DoS) attacks have emerged as a serious vulnerability for almost every Internet service [1]. DoS attacks can have far-reaching consequences than simple disruption of services and financial losses. Further, the increasingly automated nature of these attacks results in faster exploitation of vulnerabilities and a shorter time for people to install protective patches. In recent years there have been a series of large-scale DoS attacks, which have effectively shut down popular websites such as Yahoo and Amazon [2] and have crippled DNS services [3]. The CodeRed [4] worm attack on the White House website and MyDoom worm attack [5] on website of SCO group (www.sco.com) are other examples of severe attacks.

Many computing systems depend on adaptation mechanisms to remain in quiescent operation regions. These regions are often defined using efficiency, fairness and stability properties. In this paper we intend to expose a novel category of attacks, which aims to exploit this adaptive behavior to degrade the system performance while still escaping detection. As an example, consider the congestion control mechanism

employed by end system protocols such as TCP. Additive Increase Multiplicative Decrease (AIMD) [6] policy of TCP congestion control ensures that flows react adequately to congestion without impacting the performance of each other. Consider a situation where several TCP flows share a common bottleneck link. TCP AIMD behavior eventually results in high link utilization with the competing flows receiving their fair share of the link bandwidth. Now at this point in time, a large surge of packets transmitted by an attacker, will overload the link and result in significant packet losses for all the TCP flows. Mistaking this as the onset of congestion, the TCP senders will sharply reduce their transmission rates causing underutilization of the link. However, in reality, the packet loss is a result of the transient overload artificially induced by the attacker, rather than a feedback for congestion. The attacker can simply repeat the process, forcing the system to operate in degraded mode. Subsequent throttling has serious implications on the Quality of Service (QoS) experienced by the adaptive TCP flows. Note that, in order to carry out this attack the adversary simply needs to generate a low-intensity attack stream, which could constitute a square pulse wave. Albeit simplistic, this attack illustrates how adaptation strategies may be exploited by adversaries to exploit system fidelity. Similar attacks could be used in admission controllers employed in multimedia systems [7], operating systems [8] and web/media servers [9] as well as load balancing policies used in cluster-based web servers [10] which exhibit similar adaptation mechanisms.

Rest of the paper is organized as follows. Section II discusses about denial of service attacks in general and explains how they contrast with the attacks discussed in this paper. In Section III, we explain TCP AIMD mechanism in greater detail with particular stress on the behavior that will be exploited in proposed attack. Section IV provides a detailed discussion on new low-rate tuneable attacks and illustrates using a simple example of a single TCP flow how such an attack can be engineered. Section V presents NS-2 simulation experiments to validate the effectiveness of these attacks and demonstrates that a well-orchestrated attack can cause significant performance degradation.

## II. RELATED WORK

The term *denial-of-service* was originally coined by Gilgor [11] in context of operating systems. It has since been

associated with many types of attacks and can be broadly defined as follows [12]:

*A denial-of-service attack occurs when a group of users of a specified set of shared resources intentionally uses the resources with the goal and effect of causing consumption or corruption of the shared resources in such a way that some other set of entities have their ability to access the otherwise usable resources degraded or so delayed as to render them unusable.*

Note that scope of the term *resources* in the above definition is very broad and encompasses all kinds of consumable resources such as network bandwidth, server or router CPU cycles, memory, server interrupt processing capacity, specific protocol data structures and even battery capacity in mobile devices. Example DoS attacks include TCP SYN attacks; ICMP directed broadcasts (Smurf attacks); and DNS flood attacks.

Common to the above attacks is the large number of compromised machines or agents involved and a "sledge-hammer" approach of high-rate transmission of packets towards attacked node. While potentially quite harmful, the high-rate nature of such attacks presents a statistical anomaly to normal network behavior, resulting in relatively easy detection. Quite a few DoS detection mechanisms [13]–[17] interpret such anomalous behavior to be an indication that an attack is underway. For example, [13] uses bloom filter/leaky bucket arrays to monitor key traffic statistics and employs a Bayesian theoretic approach to assign a "score" called Conditional Legitimate Probability (CLP) for each packet. Packets with lower CLP scores than a certain threshold are judged to be malicious packets. Once an attack has been detected, packet filters [24] are activated which either drop all traffic confirming to the attack signature or employ certain rate-limiting techniques to restrict the impact of attack traffic. Pushback co-operative mechanisms [14] can also be used whereby congested router can ask adjacent upstream routers to rate limit the suspected attack traffic.

A common feature of DoS attacks is to use spoofed IP addresses as source address. Address spoofing is easy to accomplish due to lack of any accountability in Internet. Some DoS detection schemes in fact exploit this trait by monitoring the arrival rate of new source IP addresses [18], [19]. A few other DoS prevention proposals [20]–[22] resort to probabilistically marking of packets with path information by intermediate routers. Victim can then reconstruct (Traceback) the attack path using marked packets. A more elaborate taxonomy of DoS attacks and a comprehensive survey of available defense mechanisms can be found in [23].

Low-bandwidth attacks can easily evade detection by all of the above DoS defenses whereby we would expect the attacker to send a burst of malicious packets in a short duration and then simply shut off during other durations resulting in a very low average rate. Such bursty behavior is often observed in Internet, for example during "flash crowds" [25] when a sudden large burst of traffic is experienced at

certain websites. Hence, it becomes extremely difficult for DoS detection mechanisms to differentiate between bursty attack stream and legitimate user traffic. [26] shows how an attacker could potentially shut off communication between two parties by mounting what is termed as a *TCP shrew attack*, which exploits timeout mechanism of TCP. In this paper we will show that low-rate tuneable attacks that exploit the adaptive nature of network protocols can result in even more serious consequences as compared to shrew attacks. In fact, shrew attacks can be a particular case of more general class of low bandwidth DoS attacks. [27] presents another new class of low-rate DoS attacks which exploit the fact that frequently used data structures have average case expected running time that's far more efficient than the worst case. Using an example of hash tables, the authors illustrate that the attacker can easily generate an extremely low-rate input traffic, which will always result in worst-case behavior of targeted hash table. The attacks that we discuss in this paper defer significantly from these kinds of attacks, since the intent of the attacker is to exploit adaptive nature of systems as opposed to targeting inefficiencies in worst-case performance of algorithms.

### III. OVERVIEW OF TCP CONGESTION CONTROL MECHANISM

Low-rate tuneable adversarial attacks exposed in this paper can be used to exploit system dynamics inherent in adaptive mechanisms employed by various end-systems such as admission controllers and web servers and protocols such as TCP. However, for the sake of clarity in explanation, we illustrate how these attacks can be targeted to exploit adaptive behavior exhibited by congestion control mechanism of TCP. A detailed discussion about TCP congestion control algorithms is provided in [28]. In this section we briefly provide an overview of these congestion control mechanisms with a particular emphasis on those elements that our attacks will seek to exploit.

TCP congestion control follows an adaptive approach by estimating network capacity and regulating the amount of data that can be transient in the network at any time. TCP sender maintains a variable called *Congestion Window (Cwnd)*, which represents upper bound on amount of unacknowledged data that can be transmitted. The amount of unacknowledged data that a TCP sender can inject into network is referred to as *FlightSize* which is set to  $\min(Cwnd, receiver's\ advertised\ window)$ . TCP employs Additive Increase Multiplicative Decrease (AIMD) mechanism for adjusting *Cwnd* in response to congestion. When TCP sender perceives no congestion in network (implied by correct reception of all acknowledgments), *Cwnd* is increased linearly by one MSS for each Round Trip Time (RTT). TCP sender perceives packet loss as an indication of congestion. It distinguishes between two loss events: (i) triple duplicate acknowledgments and (ii) retransmission timer timeout. In former case, *Cwnd* is reduced to  $\min(FlightSize/2, MSS)$ . Expiration of retransmission timer before receiving an acknowledgment is perceived to be an indication of extreme congestion and

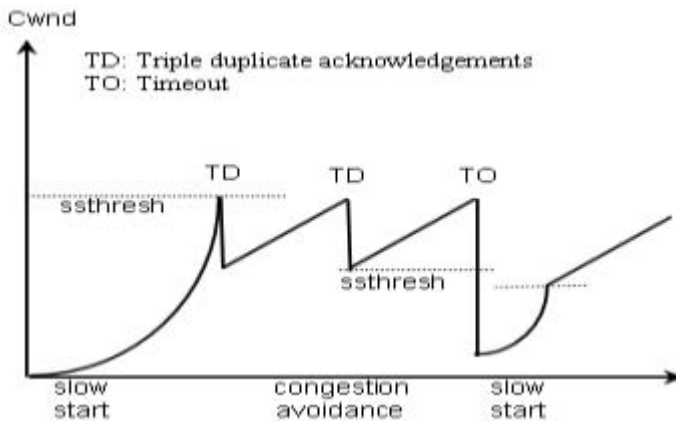


Fig. 1. Evolution of TCP Reno's congestion window

hence  $Cwnd$  is set to 1 MSS (or 2 in some implementations). Note that we have described TCP Reno's behavior so far. In TCP Tahoe,  $Cwnd$  is always reduced to 1 MSS, in response to both loss events.

When a TCP connection begins,  $Cwnd$  is initialized to 1 (or 2) MSS. With linear increase of  $Cwnd$ , it would take a long time before sending rate is ramped up to a respectable value. Hence in the initial phase, known as slow start,  $Cwnd$  is increased exponentially by doubling it every RTT. This is followed by linear increase phase of  $Cwnd$ , referred to as congestion avoidance. A threshold value denoted by  $ssthresh$  governs the transition from one phase to another. As long as  $Cwnd$  is less than  $ssthresh$  slow start phase is in operation. Once  $Cwnd$  becomes greater than  $ssthresh$ , congestion avoidance kicks in. Initially  $ssthresh$  is set to a very large value. In event of either of the two packet loss events (i.e. triple duplicate acknowledgments or timeout),  $ssthresh$  is updated as follows:

$$ssthresh = \max\left(\frac{FlightSize}{2}, 2 * Segment\right) \quad (1)$$

Adaptation mechanism employed by AIMD is crucial for easing congestion in network and also to ensure that the link bandwidth is fairly allocated among multiple TCP connections sharing a common link. Continuous adaptation of  $ssthresh$  ensures that it remains close to the available network capacity. If the network is stable,  $ssthresh$  limit will become stable and hence TCP flow will be able to transmit at a steady rate averaging around  $ssthresh$  and closer to available network capacity. Main premise here is that packet loss occurs only as a result of congestion. It has been shown in [29] that TCP does not perform efficiently in presence of other causes of packet loss such as interference in wireless environments. If an adversary artificially induced packet losses at certain crucial instants during the execution of TCP's AIMD algorithm an even worse performance degradation could be expected. It is precisely these kinds of attacks that we seek to explore in this paper. By injecting artificial packet loss, attacker can force TCP flows to reduce  $ssthresh$  limit and continuously

maintain a lower value resulting in transmission at lower rate. Recall that  $ssthresh$  determines the transition from slow start to congestion avoidance. Since  $Cwnd$  increases linearly in congestion avoidance as opposed to exponential in slow start, lower value of  $ssthresh$  expires slow start prematurely resulting in a lower aggregate throughput for victim TCP flow. In subsequent section we explain in greater detail how such an attack can be launched to exploit these characteristics of a TCP flow. We also illustrate how the attacker can tune the attack stream, which minimizes the chance of detection while still causing significant performance degradation to legitimate user traffic.

#### IV. ATTACK METHODOLOGY

In this section we demonstrate how an attacker can exploit adaptive behavior of TCP's congestion control mechanism to launch a low-rate adversarial attack. For the sake of simplicity we will assume that the attacker is targeting a single TCP connection. We assume that the attacker can inject a Constant Bit Rate (CBR) traffic stream that shares the bottleneck link for TCP connection. Note that, the attack methodology can be easily extended to target multiple TCP flows. TCP flow is first monitored to estimate various important TCP congestion control variables such as  $ssthresh$  and  $FlightSize$  and also the network capacity of bottleneck link. These values are then used to generate CBR attack traffic stream, which consists of a short data burst sufficient to overrun the capacity of shared bottleneck link. This ensures that TCP flow will experience packet loss, which is interpreted as a sign of congestion. In reaction, the AIMD mechanism of TCP flow reduces  $ssthresh$  limit and  $Cwnd$  to a considerably lower value, significantly reducing aggregate throughput. Thus, by creating a false notion of congestion, the attacker is able to exploit the adaptive mechanism of TCP, which limits achievable throughput of the flow. Subsequently, the attacker continues to monitor TCP flow and repeats short bursts at intelligently chosen intervals (for example to coincide with congestion avoidance phase of AIMD) such that TCP flow continues to experience loss. Attacker can tune the size and frequency of these bursts to create an attack stream, which is not overly anomalous as compared to legitimate user traffic, thus making detection of attack as difficult as possible. Note that, this attack is in total contrast to a high-rate sustained DoS attack where the attacker simply bombards the link with a high-rate attack traffic thus preventing legitimate user traffic to acquire any share of bandwidth. However, consistent barrage of attack traffic makes it considerably easy to detect these DoS attacks. On the other hand, in case of well-orchestrated low-rate tuned attack discussed here, it is next to impossible for victim(s) to figure out the malicious intent of the attacker. The victim(s) is(are) in fact convinced that low performance being experienced is either due to low capacity of bottleneck link or congestion within network, which is not out of ordinary in Internet.

The process of launching such an attack can be said to consist of four phases. Each of the phases is described in greater detail in following subsections. We reiterate that for

ease of explanation we are assuming that victim TCP flow is the sole flow operating over a particular bottleneck link.

### A. Monitoring Phase

We define phase one to be *monitoring phase*, wherein malicious agents (i.e. sniffers) deployed by the attacker observe the dynamics of TCP flow to estimate values of certain crucial TCP state variables such as *ssthresh* and *FlightSize*. The success of attack depends upon how effectively the malicious agents can estimate these parameters. Passive packet sniffers freely available on Internet such as Ethereal [30] and Snort [31] can be used as monitoring agents. Alternatively, the attacker may use a custom developed sniffer. It is sufficient to deploy sniffer software at any point along end-to-end path between two communication end-points of TCP connection. For simplicity we will assume that one end of TCP connection is sending data (source) and other end is simply receiving data and sending back acknowledgments (receiver). Sniffer works in a passive mode and is able to observe all the traffic to and from a TCP source. Since protocol headers (TCP, IP, etc) are transmitted unencrypted, the sniffer can readily examine information contained therein. As a result, it is able to extract sequence number of last packet (Before sniffer performs calculation) transmitted by source (*LastSeqNum*), size of this packet (*PktLen*) and last acknowledgment number (*LastAckNum*) received. With this information in hand, *FlightSize* can be calculated as shown in Equation 2. *FlightSize* is continuously updated as sniffer observes a newly transmitted segment or acknowledgment.

$$FlightSize = LastSeqNum + PktLen - LastAckNum \quad (2)$$

To estimate *ssthresh* value, the sniffer has to wait until TCP flow encounters either one of the two possible loss events: (i) triple duplicate acknowledgments or (ii) timeout. In either case, TCP source will retransmit lost packets. Sniffer is able to detect this by noticing the non-increasing nature of sequence numbers in segments transmitted from source. In fact, it can detect triple duplicate acknowledgment event prior to actual retransmissions by monitoring acknowledgments from receiver. In event of packet loss, TCP flow will reduce its *ssthresh* value to maximum of half the *FlightSize* and MSS (see Equation 1). Since the attacker is aware of value of *FlightSize* when the loss event occurred (from Equation 2), given that MSS can easily be deduced by examining TCP headers, its easy to estimate *ssthresh*.

The packet sniffer continuously monitors *ssthresh* and *FlightSize* for targeted TCP flow. These values are used in subsequent phases to generate an appropriate attack traffic pattern.

### B. Link Capacity Estimation Phase

Given the attacker is aware of *FlightSize* and *ssthresh*, next task involves determining the amount of transit data that the link is capable of supporting. Let  $F1$  be the *FlightSize* at time instant when TCP flow experiences a packet loss event and  $N$  represent the link transit data capacity, Equation 3 provides

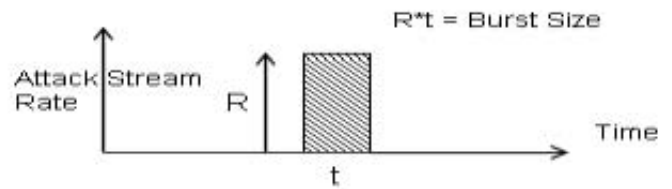


Fig. 2. Attack Burst

a good approximation of link capacity. Logic behind this argument stems from the fact that if TCP flow is in slow start phase, it is trying to double its *Cwnd* when it experiences packet loss. This implies that link has sufficient capacity to transmit data equal to  $F1$  but it has limited capacity to transmit twice that amount.

$$F1 \leq N \leq 2 * F1 \quad (3)$$

Note that, if the flow was in congestion avoidance, then the upper bound for Equation 3 would be  $F1 + MSS$ . However, bounds in Equation 3 encompasses both situations. Given an approximate estimate of network capacity, the attacker can generate a burst of approximate size using following equation, where  $N$  is initially assumed to be equal to lower bound,  $F1$ .

$$BurstSize = N - FlightSize \quad (4)$$

Note that *FlightSize* here refers to current value of this variable when the attacker is planning to transmit the burst. Attacker should transmit the burst at maximum possible transmission rate (say  $R$  bps), thus minimizing the duration of burst ( $t$  in Figure 2). Current estimate of link capacity is over a wide range of values as indicated by Equation 3. Attacker still needs to fine-tune this to determine a more accurate estimate. Hence, the attacker continues to send the burst with an increasing value of  $N$ , starting with the lowest value in predicted range from Equation 3 (i.e.  $F1$ ) until the sniffer is able to determine that TCP flow experienced packet loss. Let  $BS$  denote the size of burst that creates packet loss event, link capacity is accurately estimated using the following equation:

$$N = BurstSize + FlightSize \quad (5)$$

Since link capacity needs to be estimated only at the start of launching attack, current phase is activated only once.

### C. Initial Attack Phase

Equipped with the knowledge of link capacity and current values of *ssthresh* and *FlightSize* available from above two phases, the attacker is now ready to launch attack. The aim of the attacker is to induce momentary congestion in bottleneck link of TCP flow, which generates a false feedback signal for TCP source and forces it to adjust its transmission rate to a lower value. Recall from Section III, that continuous adaptation of *ssthresh* ensures that it remains close to available network capacity. Transient congestion induced by the attacker succeeds in reducing *ssthresh* limit for TCP flow

to a considerably lower value. TCP flow infers this as an indication of limited link capacity of bottleneck, and adjusts its transmission rate accordingly. A secondary goal of the attacker is to try and cause retransmission timer of TCP sender to timeout. Recall that in this situation TCP source reduces its  $Cwnd$  to a very low value (1 or 2 segments), thus throttling transmission rate significantly. In addition, reducing  $ssthresh$  results in premature expiry of aggressive slow start phase causing the flow to operate in congestion avoidance phase mostly. Since  $Cwnd$  increases conservatively (linearly) during this phase, it reduces the rate at which the flow can increase its throughput.

The attacker can very easily induce transient congestion by simply transmitting a short burst of packets at a constant bit rate (i.e. CBR traffic), which is sufficient to overrun the capacity of bottleneck link. Ideally the attacker should try and reduce  $ssthresh$  to as low a value as possible to achieve severe performance degradation, however, this would mean transmitting a very large burst of packets and hence risk of detection. It is advisable to pick a suitable value for  $ssthresh$ , which acts as a good compromise between cost of launching attack (i.e. packet burst) and damage caused (i.e. throughput degradation). Note that, initially  $ssthresh$  value would either be very large (if TCP flow has just been initiated) or roughly close to half maximum link capacity (assuming flow is in steady state). Recall that monitoring operations discussed in first phase are performed continuously and attacker has up-to-date information about current values of  $FlightSize$  and  $ssthresh$ . Lets assume that  $ssthresh-t$  denotes desired value of threshold. Recall that a loss event causes TCP sender to reduce its  $ssthresh$  to half the size of its current  $FlightSize$ . Theoretically, the attacker may be able to reduce threshold directly to  $ssthresh-t$  by transmitting a sufficiently large burst to induce congestion when current  $FlightSize$  is twice  $ssthresh-t$ , however, it is quite likely that size of burst needed for achieving this instant reduction is very large and could thus expose attack to detection. A better strategy involves sending multiple bursts and reducing  $ssthresh$  to desired value gradually. A simple way to achieve this is to transmit a sufficiently large burst (calculated using Equation 4) when the current  $FlightSize$  is equal to previously recorded  $ssthresh$ . The loss event resulting from burst will cause TCP flow to reduce its  $ssthresh$  to roughly half its previous value. By repeating this pattern TCP flow will exponentially decrease its  $ssthresh$  until an agreeable value is reached. Note that this strategy involves sending a considerably short burst of packets thus preventing detection and also converges to the desired  $ssthresh$  value very quickly. Our simulation experiments (discussed in Section V) show that a TCP flow can easily be forced to use just 25% of available capacity with very limited attack cost.

#### D. Continuous Attack Phase

Previous phase discusses how the attacker can successfully reduce  $ssthresh$  value of TCP flow. However, it is not sufficient to merely achieve this initial reduction since continuous probing mechanism employed by TCP will eventually enable

the flow to ramp up its transmission rate. To create a long-term impact on aggregate throughput, the attacker should recreate transient congestion on a frequent basis by repeating the attack bursts several times. This ensures that  $ssthresh$  value never approaches actual network capacity. In fact, by manipulating interarrival time and size of bursts, the attacker can tune the dynamics of  $ssthresh$  value and thus control the resulting throughput of TCP flow. Note that, in between subsequent bursts, the attacker can refrain from transmitting data to be as evasive as possible. It is a common observation in many Internet applications such as Web browsing that typical user traffic is indeed bursty in nature. Hence attack traffic can easily blend in as normal traffic and not be detected.

There are two choices available to the attacker while deciding inter burst durations. A simple choice would be to try and maintain  $ssthresh$  at same value as in initial attack phase. To achieve this, attack burst should be transmitted when  $FlightSize$  grows to twice the value of desired  $ssthresh$  value from third phase. This can be easily achieved by allowing current  $FlightSize$  to grow to twice of desired  $ssthresh$  value. Attack burst is transmitted at this time, with burst size calculated using Equation 4. Congestion caused by attack burst will result in  $ssthresh$  roughly remaining close to desired value. We refer to this policy as *periodic continuous attack phase*. Though this will result in a low average rate for victimized TCP flow, the attack traffic pattern is steady and more or less periodic with approximately same size of the burst. An intelligent victim might be able to detect this pattern. Advantage however, is that the attacker has to perform very little computation in terms of determining the size of burst and suitable time for transmitting the same. Further the attacker is also able to accurately estimate the damage incurred.

To put victim off-guard, the attacker may instead resort to second alternative, which involves varying the interval between subsequent bursts and also their size to prevent the attack stream from exhibiting any repetitive patterns. This phase adds a pre-calculated increment  $\Delta$  to  $ssthresh$  limit as follows; current  $FlightSize$  of TCP flow is allowed to increase up to the value of  $2 * ssthresh + \Delta$ . At this time, the attack burst is transmitted, size of which is calculated using Equation 4. Note that, in this case following the loss event, TCP source will update its  $ssthresh$  limit to new value of  $ssthresh(old) + \Delta/2$ . Each subsequent increase of  $\Delta$  will result in a continuous increase in  $ssthresh$  value of the TCP flow, every time it experiences loss. Attacker can allow this increase until a pre-calculated higher limit for  $ssthresh$  is reached at which time initial attack phase can be re-entered to reduce  $ssthresh$  limit again. Note that, as  $ssthresh$  limit increases, required size of attack burst decreases gradually. Moreover, interval between two consecutive bursts also increases, resulting in a lower average rate as compared to the periodic attack alternative discussed above. However, performance degradation achieved is slightly lower than the former policy. Irregular traffic pattern and low cost of initiating attack makes this option referred to as *incremental continuous attack phase*, a highly suitable option.

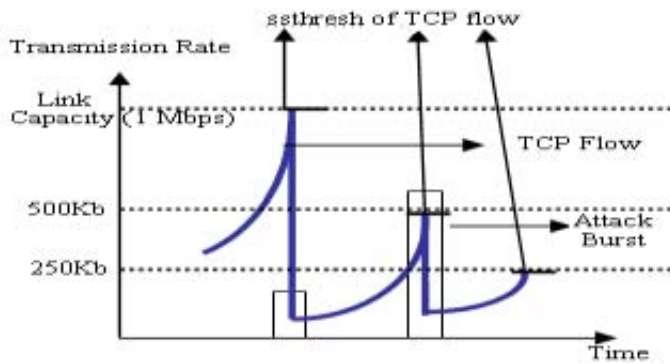


Fig. 3. Initial Attack Phase

### E. An Example Attack

We not illustrate a simple example depicting how such an attack can be launched with relative ease. Assume that a single TCP flow is traversing a bottleneck link of capacity 1 Mbps. Assume that the attacker has deployed a passive sniffer, which is able to monitor data and acknowledgment packets for this flow. Figures 3-5 illustrate various phases in attack and their effect on  $Cwnd$  and hence throughput of TCP flow.

Assume that initial monitoring needed to estimate initial  $ssthresh$  and the estimation of link capacity has already been completed. Assume that  $ssthresh$  is currently set at 1 Mbps. Hence as shown in Figure 3, the flow will be operating in slow start phase and doubling the  $Cwnd$  with each RTT. Recall that the sniffer deployed by the attacker is continuously monitoring  $FlightSize$ . Now at the time when  $FlightSize$  approaches  $ssthresh$  the attacker launches initial attack phase by transmitting a small burst calculated using Equation 4, as depicted in Figure 3. Assume that resulting transient congestion causes TCP sender retransmission timer to expire. As a result, TCP flow reduces  $ssthresh$  to half (500 Kb) and congestion window is set to a very low value (usually  $2 * MSS$ ). The flow then enters slow start and again starts exponentially increasing its  $Cwnd$ . When the current  $FlightSize$  reaches 500Kb (current  $ssthresh$ ), attack flow transmits a burst of size greater than 500Kb, which will again result in loss and subsequently reduces  $ssthresh$  limit to 250Kb (75% reduction of the original value). Further reduction of  $ssthresh$  beyond this value would require a burst greater than 750Kb, which may increase the chance of detection. The attacker thus decides to force the TCP flow to maintain this value and enters the continuous attack phase.

Figure 4 illustrates the *periodic continuous attack phase* wherein; the attacker ensures that  $ssthresh$  value is consistently maintained at 250Kb. As is evident, this is achieved with minimal effort by simply transmitting attack bursts of a constant size (less than 500Kb) when current  $FlightSize$  becomes greater than desired  $ssthresh$  (250Kb). One can readily see from Figure 4 that average network capacity being used by TCP flow during this phase is approximately equal to 250Kb, which results in a 75% utilisation of link capacity. Further,

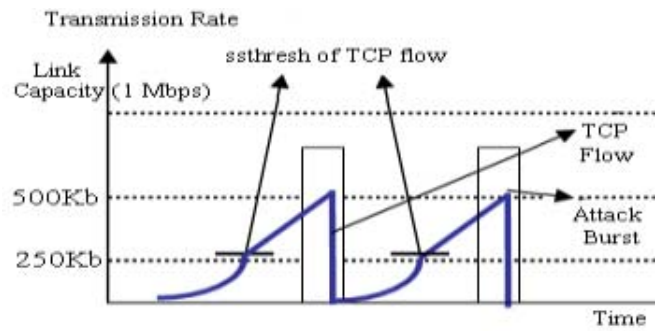


Fig. 4. Periodic Continuous Attack Phase

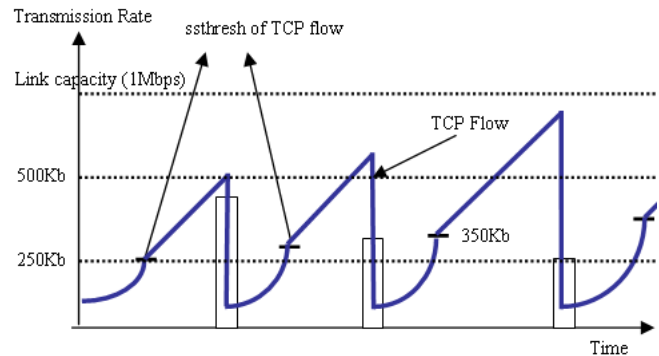


Fig. 5. Incremental Continuous Attack Phase

since the attack stream constitutes of short bursts of packets at regular intervals interspersed with long inactive periods, its average transmission rate is very low. This demonstrates severe performance degradation that can be achieved by tuneable low-rate attacks with minimal costs.

Periodic nature of above attack stream could possibly give away the game for the attacker. Hence, after a pre-calculated period of time, attacking flow shifts to the *incremental continuous attack phase*, which allows  $ssthresh$  limit to increase up to 450Kb in steps of 50Kb, the value of  $\Delta$  being 100Kb, as illustrated in Figure 5. Average network capacity being utilized by TCP flow is approximately 350Kb, which is slightly higher than that in the periodic phase, but the benefit in this case is non-periodic nature of attack stream, making the attacker virtually invisible to victim.

## V. SIMULATIONS

In this section we demonstrate using NS-2 simulations that low-rate tuneable attacks discussed in this paper can be effectively launched to target adaptive behavior exhibited by TCP flows. We also evaluate the extent of performance degradation that can be achieved by these attacks. Our initial results indicate that well-orchestrated low-rate attacks can severely degrade the throughput of a well-behaved TCP flow.

We consider the same scenario used in our example in Section IV-E. We assume that a single TCP flow is transmitting over a 1 Mbps link. We have experimented with both TCP Tahoe and TCP Reno in our study and have successfully

created the attack scenarios for both implementations. The cost of launching the attack with TCP Reno was slightly higher as compared to TCP Tahoe. This is due to the fact that TCP Reno does not always timeout due to the congestion and hence does not reduce its  $Cwnd$  to a very low value. Due to lack of space, we only provide the results for TCP Tahoe in this section. For the results discussed below, we have used a CBR traffic source to create the attack bursts.

We first demonstrate how the attack flow can successfully control the  $ssthresh$  value of the TCP flow. Figure 6 plots the congestion window of the TCP flow (in terms of packets) as a function of time (green line). We assume that the MSS is set to 1000 bytes and all packets are of the same size. The figure also illustrates the attack traffic bursts (red line). The short black lines indicate the transition from slow start to congestion avoidance. The initial monitoring for estimating the  $ssthresh$  and the network capacity is not illustrated in this graph. The initial value of the  $ssthresh$  limit is 28 packets. Notice that at  $t = 0.4sec$ , when the current  $FlightSize$  becomes equal to this threshold, the attacker initiates the initial attack phase by transmitting a short burst of packets. As seen from Figure 6, overruns the capacity of the link and the TCP flow encounters packet loss and subsequently times out. Consequently the  $ssthresh$  value is reduced to half, i.e. 14 packets. Also notice that the  $Cwnd$  is set to a very low value ( $2 * MSS$ ) and the flow enters slow start. Now at  $t = 1sec$ , the  $FlightSize$  approaches the current  $ssthresh$  of 14 packets and the attacker transmits a slightly larger burst to repeat the above effect and reduce the  $ssthresh$  further to 7 packets. Notice that, the initial attack phase has reduced the  $ssthresh$  from its initial value of 28 packets down to 7, i.e. a 75% decrease. Having reduced the TCP  $ssthresh$  to a reasonable lower value, the attack flow enters the periodic continuous attack phase and tries to maintain the  $ssthresh$  at this low value. As shown in Figure 6, to achieve this attack bursts are transmitted such that the TCP flow experiences loss at time,  $t = 2.4and3.5sec$  respectively. Following this the attack flow transitions to the incremental continuous attack phase and allows the  $ssthresh$  limit to increase to 8 and 9 packets in consecutive burst cycles. To achieve this attack bursts are transmitted at time,  $t = 4.8and6.3sec$  respectively. This clearly indicates that the attack flow can easily adjust the  $ssthresh$  limit of the TCP flow by careful estimation of the  $FlightSize$ .

Next we evaluate the effect of the attack traffic on the throughput of the TCP flow. Figure 7 shows the evolution of the TCP bandwidth both with and without the presence of the attack traffic as a function of time along with the attack traffic. These results correspond to the same experiment as in Figure 6. The graph clearly indicates the detrimental effect of injecting a well-orchestrated attack burst. In the absence of the attack stream, the TCP flow can achieve almost 100% utilization of the link capacity. The average link utilization drops to 65% of the link capacity during the periodic continuous attack phase. Notice that, this significant reduction in the throughput is achieved by a low-rate attack stream, which utilises just 9% of the link capacity. Thus, the

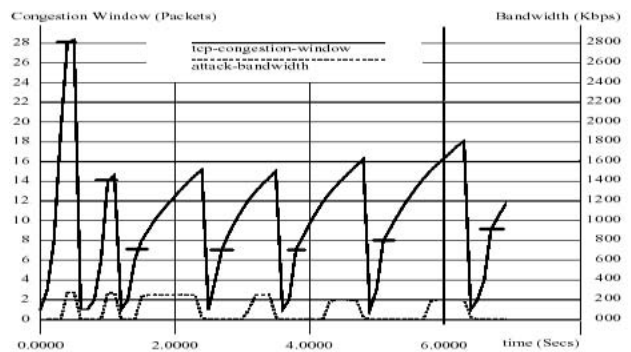


Fig. 6. Effect of the attack on the  $ssthresh$  and  $cwnd$

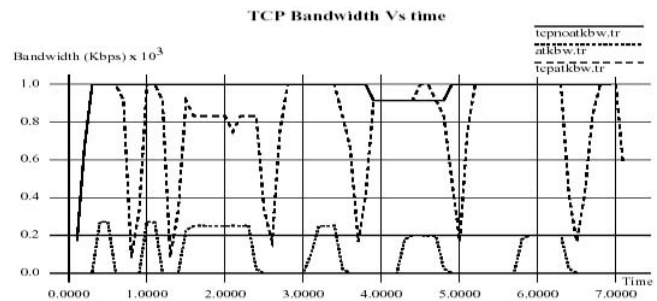


Fig. 7. Effect of attack stream on TCP throughput

cost involved in launching this attack is minimal and also makes it hard to detect the attack. The incremental continuous attack stage sacrifices the performance degradation by a small margin (68% bandwidth utilisation for the TCP flow in this phase) but generates a non-uniform attack stream, which is even more difficult to detect.

Note, even though we have only considered throughput degradation, we can expect similar effect on other QoS parameters of targeted TCP flow such as delay, delay jitter, etc.

## VI. CONCLUSION

In this paper we presented a novel class of low-rate tuneable adversarial attacks, which seek to exploit the adaptive behavior exhibited by several network end-systems and protocols. In particular, we show how such attacks can be launched to target the congestion control mechanisms employed by TCP. Using a step-by-step approach we demonstrated the relative ease with which an attacker can achieve significant performance degradation in the throughput of TCP flows by using a simple packet sniffer and some basic computations. The attack stream consists of short bursts of packets sufficient to create transient congestion, which is extremely hard to detect due to its non-anomalous behavior. Simulation experiments corroborate our findings and demonstrate that the throughput of a victimized TCP flow can be reduced by almost 50% by injecting a non-obvious low-rate attack stream, which on average utilizes only 10% of the bandwidth of the targeted link. Though, the simulations presented concentrated on the reduction in throughput, we can expect similar degradation in other QoS

parameters such as delay, jitter, etc. In fact, the attacker can easily tailor the attack stream to target any specific QoS performance metric that he wishes to degrade. Lastly, though this paper has focussed on exploiting the congestion control behaviour of TCP flows, these attacks can be used to exploit similar system dynamics that are produced as a consequence of the adaptive mechanisms in place in various systems such as web servers, admission controllers, etc.

There are several avenues that we intend to explore in the future. First and foremost we intend to mathematically analyze the effects of these attacks on the performance of legitimate TCP flows. We also intend to extend the attack techniques for targeting multiple TCP flows. Empirical experiments are also planned to evaluate performance degradation experienced by various real applications and also to demonstrate the ineffectiveness of current DoS prevention techniques in detecting these attacks. Most importantly, we also intend to develop an appropriate defense mechanism for preventing these attacks.

#### REFERENCES

- [1] CERT Coordination Center, *Denial of Service Attacks*, [http://www.cert.org/tech\\_tips/denial\\_of\\_service.html](http://www.cert.org/tech_tips/denial_of_service.html).
- [2] M. Williams, *Ebay, Amazon, Buy.com hit by attacks*, <http://www.nwfusion.com/news/2000/0209attack.html>.
- [3] CAIDA, *DNS Nameserver Dos Attack, October 2002*, <http://www.caida.org/projects/dns-analysis/oct02dos.xml>.
- [4] CERT, *Code Red Worm Exploiting Buffer Overflow in IIS Indexing Service DLL*, [http://www.cert.org/incident\\_notes/IN-2001-08.html](http://www.cert.org/incident_notes/IN-2001-08.html).
- [5] PC Magazine, *MyDoom Hobbles Internet E-mail*, <http://www.pcmag.com/article2/0,4149,1463442,00.asp>.
- [6] V. Jacobson and M. Karels, *Congestion Avoidance and Control*, in Proceedings of ACM SIGCOMM, 1988.
- [7] E. Knightly and N. Shroff, *Admission Control for Statistical QoS: Theory and Practice*, in IEEE Networks, vol. 13, no.2, 1999, pp. 20-29.
- [8] M. Welsh and D. Culler, *Overload Management as a Fundamental Service Design Primitive*, in Proceedings of the Tenth ACM SIGOPS European Workshop, France, September 2002.
- [9] L. Cherkasova and P. Phaal, *Predictive Admission Control Strategy for Overloaded Commercial Web Server*, in Proceedings of MACOTS, August 2000.
- [10] M. Aron, P. Druschel and W. Zwaenepoel, *Cluster Reserves: A Mechanism for Resource Management in Cluster-Based Network Servers*, in Proceedings of Measurement and Modelling of Computer Systems, 2000.
- [11] V. D. Gilgor, *A Note on the Denial of Service Problem*, in Proceedings of 1983 Symposium on Security and Privacy, 1983, pp. 139-149.
- [12] C. Shields, *What do we mean by Network Denial of Service*, in Proceedings of the 2002 IEEE Workshop on Information Assurance and Security, West Point, USA, June 2002.
- [13] Y. Kim, W. Lau, M. Chuah and H. Chao, *PacketScore: Statistics-based Overload Control against Distributed Denial of Service Attacks*, in Proceedings of IEEE INFOCOM 2004, Hong Kong, March 2004.
- [14] J. Mirkovic, G. Prier and P. Reiher, *Attacking DDoS at the Source*, in Proceedings of the IEEE International Conference on Network Protocols, 2002.
- [15] BBN Technologies, *Intrusion Tolerance by Unpredictability and Adaptation*, <http://www.bbn.com/infosec/itua.html>.
- [16] R. Mahajan, S. Bellovin, S. Floyd, V. Paxson and S. Shenker, *Controlling High Bandwidth Aggregates in the Network*, ACM Computer Communication Review, Vol. 32, no. 3, July 2002.
- [17] Arbor Networks, *The Peakflow Platform*, <http://www.arbornetworks.com>.
- [18] H. Kim and I. Kang, *Real-Time Visualization of Network Attacks on High-Speed Links*, in IEEE Network, September 2004, pp. 30-39.
- [19] T. Peng, C. Leckie and K. Ramamohanarao, *Proactively Detecting Distributed Denial of Service Attacks Using Source IP Address Monitoring*, in Proceedings of IEEE Networking, 2004.
- [20] U. Tupakula and V. Varadharajan, *Tracing DDoS Floods: An Automated Approach*, in *Journal of Network and Systems Management*, Vol. 12, No. 1, March 2004, pp. 111-135.
- [21] T. Peng, C. Leckie and K. Rammamohanarao, *Protecting from Distributed Denial of Service Attack Using History-based IP Filtering*, in Proceedings of IEEE ICC 2003, Alaska, USA, May 2003.
- [22] P. Ferguson and D. Senie, *Network Ingress Filtering: Defeating Denial of Service Attacks which Employ IP Source Address Spoofing*, IETF RFC 2267.
- [23] J. Mirkovic and P. Reiher, *A Taxonomy of DDoS Attacks and DDoS Defense Mechanisms*, in ACM Computer Communication Review, vol. 34, no. 2, April 2004.
- [24] W. Stallings, *Cryptography and Network Security*, 3rd Edition, Prentice Hall, 2004.
- [25] J. Jung, B. Krishnamurthy and M. Rabinovich, *Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites*, in Proceedings of the International World Wide Web Conference, May 2002, pp.252-262.
- [26] 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 03, Karlsruhe, Germany, August 2003.
- [27] S. Crosby and D. Wallach, *Denial of Service via Algorithmic Complexity Attacks*, in Proceedings of the 12th USENIX Security Symposium, August 2003, pp.29-44.
- [28] M. Allman, V. Paxson, W. Stevens, *TCP Congestion Control*. April 1999. RFC 2581.
- [29] H. Balakrishnan, V. Padmanabhan, S. Seshan and R. Kartz, *A Comparison of Mechanisms for Improving TCP over Wireless Links*, in ACM SIGCOMM, 1996.
- [30] *Ethereal: A Network Protocol Analyzer*, [www.ethereal.com](http://www.ethereal.com)
- [31] *Snort: The defacto standard for intrusion detection/prevention*, [www.snort.org](http://www.snort.org)