

# TCP based Denial-of-Service Attacks to Edge Network: Analysis and Detection

V. Anil Kumar<sup>1</sup> and Dorgham Sisalem<sup>2</sup>

**1 CSIR Centre for Mathematical Modelling and Computer Simulation, Bangalore, India**  
**2 Fraunhofer Institute for Open Communication Systems, Berlin, Germany**

## Abstract

End-to-end congestion control algorithms in TCP are designed for a highly co-operative environment with the assumption that the end hosts voluntarily participate in it and obey the congestion control rules. The steady growth of malicious activities such as Denial-of-Service attacks (DoS) on the Internet reveals that the Internet no longer remains as a network of only trusted entities. The focus of this paper is on a special class of DoS attacks targeted to edge networks by exploiting the vulnerabilities of TCP congestion control algorithms to duplicate acknowledgement and optimistic acknowledgement spoofing. We analyse two DoS attack scenarios namely pulse and sustained attack arising from two different behaviours of the attacker and compare them with other widely seen DoS attacks. Our simulation results show that such attacks are feasible and also reveal the negative impact of the attacks on the target. We extend our work by presenting a simple but effective method for detecting such attacks by passively monitoring the inbound and outbound traffic of the targeted network. The detection is achieved by differentiating malicious streams of duplicate and optimistic acknowledgments from normal acknowledgments. The proposed detection technique fits well into the framework of an intrusion detection system, which can operate independently without any kind of co-operation or service from the end points.

## 1. Introduction

End-to-end congestion control as in TCP is the primary mechanism used in today's Internet to control the transmission behaviour of communication systems. A major shortfall of the congestion control employed in TCP is that it is voluntary in nature and end-system's participation in it cannot be guaranteed. There could be various motivations behind the end-systems not voluntarily participating in the congestion control process. For example, as in [1] a greedy web client can reduce the file download time by violating the congestion control principles. On the other hand, a server implementation could attempt to serve its clients faster than the standard implementation[2]. While these sorts of misbehaviour can cause unfairness among competing flows, a more dangerous consequence of violating congestion control is denial-of-service (DoS) attacks caused by such flows. The focus of this paper is on the analysis and detection of special type of DoS attacks targeted at edge networks, like the LAN of an organisation, connected to the Internet. Unlike conventional DoS attacks where external sources are used to flood the targeted network, the attacks described here exploit the transmission power of an internal machine in the targeted network to flood its own network without compromising the machine. We show through simulation that a malicious attacker by acting as a TCP client can setup a connection with a server in the targeted network and remotely control the transmission behaviour of the server to generate flood traffic of desired pattern. The potential danger arising from such a situation is explored and the results presented in this paper show that such attacks are feasible.

We analyse, two DoS attack scenarios caused by two different behaviours of the attacker, the subsequent flood patterns generated by the internal machine, and the impact of these floods on the Internet access router of the targeted network. In the first case the attacker, by suitably changing its acknowledgment (ACK) generation strategy, forces the server to generate periodic flood of ON-OFF patterns. This is called periodic pulse attack and the main victim of this attack is outbound TCP flows from the targeted network. The flood for this attack is generated in such a way that it can instantly fill the buffer of the targeted router and also maintain the buffer in the filled status for some duration. The flood thereby enforces high packet loss to TCP flows traversing the router so that the flows will experience timeouts. The periodicity in TCP's retransmission policy allows the attacker to roughly

synchronise the attack pulse with the retransmission time of TCP flows and as a result of this, TCP flows will experience frequent timeouts and severe throughput degradation. The main characteristic of the pulse attack is that even though the flood is generated for a short duration of time, it has high negative impact on the performance of TCP flows. In the second case, we study the scenario where the server is forced to generate sustained flood which can fill and maintain the router buffer for a longer period. We call this as sustained attack. The sustained attack can affect both TCP and non-TCP flows and can virtually detach the targeted network from the Internet. The desired flood patterns for both the attacks are generated by exploiting the vulnerability of TCP congestion control to duplicate and optimistic ACK spoofing [1].

Finally we present our intrusion detection approach to detect maliciously crafted duplicate and optimistic ACKs which are used to force the internal machine to generate the flood. A common characteristic of both the attacks is that the buffer occupancy of the router in the targeted network increases abruptly and this feature is used in our detection system. We define a minimum threshold RTT for outgoing packets in such a way that under normal conditions a packet cannot be acknowledged before the defined RTT period elapses. The threshold RTT is computed as a function of the instantaneous buffer occupancy of the router, and attacks are detected whenever the observed RTT falls below the threshold.

In this paper, the term ‘attack flow’ denotes the TCP flow setup by the attacker, the terms ‘sender’ or ‘server’ denotes the end-point of the attack flow on the internal machine, and the term ‘attacker’ or ‘receiver’ or ‘client’ denotes the end-point of the attack flow on the attacker’s machine. Further, the term edge or targeted router is used to refer the Internet access router of the targeted network and the term ISP router is used to refer the router located at the Internet Service Provider through which the targeted network is connected to the Internet.

We use Reno, a widely deployed and popular version of TCP, with congestion control algorithms as specified in RFC 2581 as the reference TCP model to analyse and simulate the attacks. However, by considering the fact that Reno is being replaced[23][24] by newer TCP variants like NewReno and SACK[9], we briefly evaluate the threat level in the case of these two TCPs also.

The rest of the paper is organised as below: In section 2, we present an overview of TCP’s congestion control and retransmission policies which are relevant to our discussion. In section 3 we describe the attacks in detail using Reno’s congestion control and then briefly evaluate the effectiveness of the attacks in the cases of NewReno and SACK TCPs also. In the same section, we also compare the attacks with other widely seen DoS attacks. Section 4 covers the simulation of the attacks and presents the simulation results. Section 5 describes our Intrusion detection approach to detect the attacks and finally in section 6 we conclude the paper.

## 2. Congestion Control and Retransmission Policy in TCP

Congestion Control in TCP consists of four intertwined algorithms, slow start[3], congestion avoidance,[3] fast retransmit and fast recovery [4][5]. The sender maintains a state variable called congestion window (*cwnd*), which is the sender side limit on the number of packets the sender can transmit before it receives an ACK. Another variable called *ssthresh* is used to decide whether the sender should operate in slow start or congestion avoidance. These variables are updated as the data transfer progresses and whenever network congestion is detected. When the *cwnd* is less than *ssthresh*, the sender operates in slow start and otherwise in congestion avoidance. The transmission criteria of the sender is that whenever the *cwnd* is less than the receiver advertised window (*rwnd*) and more than the number of packets outstanding in the network, the sender sends new packets until the *cwnd* becomes equal to the number of outstanding packets. After establishing the connection, the sender initialises its *cwnd* to 2 and starts with slow start phase. For each incoming ACK that acknowledges new data, the sender increments the *cwnd* by one and sends additional packets if the transmission criteria permits. During slow start, the *cwnd* and hence the number of outstanding packets in the network grows exponentially with RTT. Once the *cwnd* reaches *ssthresh*, the sender enters in to congestion avoidance and in this phase the congestion window is incremented by one in each RTT

resulting a linear growth in  $cwnd$ . Once the  $cwnd$  exceeds the  $rwnd$ , the TCP reaches steady state where the transmission criteria of the sender is independent of  $cwnd$  and the sender sends new packets as long as the number of outstanding packets is less than the  $rwnd$ . The sender detects packet loss either when it receives three duplicate ACKs or when its retransmission timer expires. When the third duplicate ACK arrives TCP goes through fast retransmission and fast recovery. First, the  $ssthresh$  is reduced to half of the number of outstanding packets or 2, whichever is greater. The lost packet is then retransmitted and the  $cwnd$  is set to  $ssthresh$  plus three. For each additional duplicate ACK, the  $cwnd$  is incremented by one and when the  $cwnd$  exceeds the number of outstanding packets, a new packet is transmitted. This is the fast recovery phase. When a new packet is acknowledged the sender exits fast recovery by setting the  $cwnd$  to  $ssthresh$  and continues with congestion avoidance. If the packet loss is detected by RTO expiry,  $ssthresh$  is reduced in the same manner as in the beginning of fast retransmission, but the  $cwnd$  is set to one and the sender proceeds with slow start.

In [1] the authors show that the congestion control in TCP is vulnerable to duplicate and optimistic ACK spoofing. Through an experimental implementation of a TCP receiver, called TCP Daytona, they show that the receiver can download files from the Internet much faster than its normal rate. Duplicate ACK spoofing represents the situation where the receiver sends large number of ACKs to the same packet and compels the sender to respond with new data packets. In optimistic ACK spoofing, the receiver sends ACKs to packets which are sent by the sender but have not yet reached the receiver. In this paper, a combination of optimistic and duplicate ACK spoofing is used to force the sender to generate the DoS flood. Next, we explain why flood with ON-OFF pattern is effective in attacking TCP flows.

TCP flows suffer major performance degradation whenever packet loss recovery is achieved through retransmission timer expiry[6][7]. This is because in the recovery process the sender not only reduces the  $cwnd$  to one but also waits for a period of at least one second, which is the minimum recommended value for RTO, before the lost packet is retransmitted[8]. Though fast retransmission and fast recovery algorithms are designed to allow a TCP sender to recover from packet loss before RTO expires, there are three major situations under which these algorithms cannot recover packet loss. First, if  $cwnd$  is less than four and a packet loss occurs, the sender cannot receive three duplicate ACKs and the recovery algorithms will not be triggered. Second, if a retransmitted packet is lost, the sender has no option other than to wait till the RTO expires before transmitting the lost packet again. Third, when multiple packets are lost within a window of packets, the recovery algorithms are not often successful even if they are triggered. For example, when two or more packets are dropped from a window of data and the  $cwnd$  is less than 10 packets, TCP Reno is forced to wait for retransmission timeout[9]. In pulse attack, DoS pulses with high intensity and short duration impose one of the three conditions mentioned above to normal TCP flows and compel them to undergo RTO based recovery process periodically.

### 3. Description of Attacks

In this section, we first describe the attacks in detail using Reno TCP, then evaluate the effectiveness of the attack on other variants of TCP and finally compare the attacks with conventional DoS attacks.

#### 3.1 Pulse attack

Consider multiple TCP flows with RTO 1 second sharing a bottleneck router. If the router is bombarded with a short duration high intensity burst which can instantly fill the router buffer and maintain the filled status for some duration, say  $t$ , all TCP flows with RTT less than  $t$  are likely to experience packet loss leading to timeout[25]. The flows will resume transmission again after 1 second and if the burst frequency is also 1 second, they will again experience timeout and the process will continue as per TCP's retransmission policy. In [25] the authors demonstrate the inability of TCP flows to co-exists with such high intensity periodic bursty traffic.

The typical DoS flood suitable for pulse attack should possess three important characteristics: First, the burst intensity should be sufficient to instantly fill the buffer of the targeted router. Second, the

flood should be able to maintain the router queue in the filled status for the desired duration. Finally, the flood period should be as close as possible to one second so that timeouts are imposed as soon as the flows restart. We next explain how an internal machine on the targeted network can be turned to a flood source without compromising it. The attacker in the form of a genuine client establishes a TCP connection with a server in the targeted network. After receiving a data packet from the server, the attacker sends a large number of duplicate ACKs forcing the server to perform a fast retransmit and then enter into fast recovery. In fast recovery, as soon as the *cwnd* exceeds the number of outstanding packets, the server starts to respond with data packets which act as the DoS flood. The number of duplicate ACKs should be of the order of the buffer size of the targeted router. Note that since the size of an ACK packet is only 40 bytes, the possibility of ACKs arriving at the server as a burst is much higher than in the case where the actual flood is directly generated by the attacker. This will ensure that the response packets from the server will be a burst. Receiver window is the only upper limit on the number of packets that the server transmits in fast recovery, and the attacker can enhance this limit by advertising high window size, typically more than the router buffer size, at the time of establishing the connection. While it is possible for the attacker to force the sender to generate the flood in slow start phase itself by generating optimistic ACKs, the fast recovery has an added advantage that packet loss will not interrupt the flood generation process. If the attacker generates optimistic ACKs by incrementing the ACK number in each successive ACKs with the assumption that the arrival of previous ACK at the server has triggered the data packet which is being acknowledged by the current ACK packet and if the previous ACK is lost, the server will stop sending new packets. However in the case of duplicate ACKs this will not happen as the server treats all the duplicate ACKs in the same manner.

The filled queue has to be maintained for the desired duration to enforce packet loss to normal TCP flows. This is possible only if new packets are added to the router buffer at the same rate as packets are dequeued. The attacker achieves this goal by shifting the attack flow from the fast recovery mode to the congestion avoidance mode by sending optimistic ACKs. For this, the attacker has to estimate the number of packets that the server has sent in fast recovery and this is explained below with the help of an example.

Consider the case where the server, after establishing the connection, sets the *cwnd* to 2 and sends packet number 1 and 2. The attacker, after receiving both the packets, sends ACK to packet 1 and then sends 50 duplicate ACKs to packet 1. Actually the ACK to 1 and duplicate ACKs are exactly the same but are mentioned here separately only for clarity of explanation. When the server receives ACK to 1, its *cwnd* becomes 3 and after receiving the 3<sup>rd</sup> duplicate ACK *cwnd* becomes 5,  $[\max(cwnd/2, 2) + 3]$ . The 50<sup>th</sup> duplicate ACK will inflate the *cwnd* to 52 and this means the server now has 52 unacknowledged packets in the network. Using this information the attacker can send optimistic ACK and clear the server's account on the number of outstanding packets. Optimistic ACKs are continued to maintain the queue for the desired duration. For example, if the targeted network's link is of 1mbps speed and the attacker's aim is to maintain the queue for 120 ms, it has to send 10 optimistic ACKs at a rate of one ACK in each 12 ms. This is because the ACKs will trigger at least one packet of size 1500 bytes in each 12 ms which is the time required for the router to dequeue 1500 bytes through 1mbps link. After sending sufficient number of ACKs, the attacker abruptly stops ACKs and this will force the attack flow to timeout. Note that the normal flows through the router are also likely to experience a timeout around the same time. The transmission process of attack and normal flows will resume after one second and the attacker repeats the whole process as soon as it sees a packet from the server.

Though the attacker can estimate the sequence number of the last packet sent by the server and generate optimistic ACK to that packet, such a precision is not mandatory. As far as the optimistic ACK satisfies the following two conditions, the attack will be successful. 1) The optimistic ACK does not acknowledge packets which are not yet sent by the server and 2) The optimistic ACK reduces the number of outstanding packets to the extent that the new value of the *cwnd* exceeds the number of outstanding packets.

### 3.2 Sustained Attack

While the pulse attack explained above can cause significant damage to TCP flows, non-TCP flows are not likely to be affected much. In fact, it is possible that the attack may facilitate non-TCP flows because of its ability to suppress TCP flows. However, a slight modification in the ACK generation policy of the attacker can result in a sustained flood which can potentially affect all traffic of the targeted system. In order to generate the sustained flood, the attacker first spoofs sufficient number of duplicate ACKs so that the resulting flood will instantly fill the router buffer and then spoofs optimistic ACKs to maintain the buffer in the filled status. Unlike pulse attack, where the optimistic ACKs are stopped after some duration, here the attacker continues to generate optimistic ACKs as long as it desires to continue the attack.

The sustained attack, in the first look, seems to be very similar to UDP[10] flows without end-to-end congestion control. However, UDP flows normally transmit at constant rate and their transmission behaviour and rate are not governed by the receiver. In the worst case, the attacker can try to launch an attack by simultaneously initiating multiple flows and forcing the sender to send at high rate. Such attempts could be prevented by restricting the number of flows permitted at any time.

### 3.3 Attacks in General Scenarios

Here we evaluate the effectiveness of the attack on NewReno and SACK TCP. We concentrate on two aspects of the attacks: First the success of the attacker in generating the DoS flood when the targeted system is based on New Reno or SACK TCP. Second the impact of the flood on NewReno and SACK based normal flows. The flood pattern of NewReno in response to spoofed duplicate and optimistic ACKs will be exactly the same as in Reno. This is because both Reno and New Reno have the same transmission criteria except that New Reno uses partial ACKs[9] (ACKs that acknowledge new packets but not the highest sent before entering into fast recovery) for speedy recovery from multiple packet losses without RTO expiry. SACK implementation in the server, irrespective of how the server processes duplicate and optimistic ACKs, does not reduce the attacker's ability to exploit it as a flood source because SACK is only an option, which the attacker can easily turn off while establishing the connection. Further, since all TCP implementations are supposed to use the same value of 1 second as the minimum RTO, flood frequency close to one second is achievable irrespective of the server TCP version. The points mentioned above indicate that most of the widely deployed TCP servers could be exploited as a flood source.

The impact of pulse attack on normal flows is likely to vary depending upon the TCP version of normal flows. But all flows, irrespective of their version, will experience the same throughput degradation if the packet loss is imposed when the *cwnd* is less than 4. If the packet loss occurs when *cwnd* is greater than or equal to 4, both NewReno and SACK flows will perform better compared to Reno because of their ability to recover from multiple packet loss without RTO expiry. Between NewReno and SACK, NewReno will be affected more because it needs one RTT to recover each loss, where as SACK can recover multiple packet losses in single RTT[9].

### 3.4 Comparison

The two attacks described above have characteristics which make them significantly different from widely seen flood based DoS attacks like Trinoo[11], TFN[12], SYN-flood[13]etc. The first and most important difference is on the physical location of the flood source. In conventional DoS attacks, the flood sources are normally located outside the targeted network, and the attacker or a set of machines on behalf of the attacker (in case of DDoS) generates high intensity flood and directs it towards the victim. The flood travels all the way from the attacker to the victim through the Internet and the success of such attacks depends on a significant portion of the flood reaching the target. Intermediate routers with advanced features like RED[14], per-flow scheduling[15] or aggregate based flood control[16] etc. will deliberately drop a considerable portion of the attack traffic and this could reduce the intensity of the attacks. Further, a bottleneck or congested link somewhere between the attacker and the victim will in fact shift the point of attack from the victim to the bottleneck link. The attacks are likely to be more effective if the flood source is closer to the target, ideally inside the targeted network itself. Pulse and sustained attacks have the advantage that the flood source is located inside

the target and hence flood is not regulated by the intermediate routers. Only ACK packets, which are much smaller in size, travel through the intermediate routers. The location of the flood source is even more significant in the case of attacks similar to pulse attacks, but are attempted using external flood sources. In such cases, the flood may experience deformation on its way from the source to the target making it ineffective. For example, the inter-packet delay of the flood may increase as it is being routed by intermediate routers and this makes the flood no more a suitable burst for pulse attack.

The second notable characteristic of the attacks is high flood amplification. The attacker sends only small sized ACK packets and gets it amplified to maximum sized packets. In a typical case, each ACK of 40 bytes is translated to packets of size 1500 bytes resulting an amplification of as high as about 38 times. In the case of commonly known DoS attacks like reflector attacks[17], Trinoo[11], TFN[12], such high packet size amplification is not likely once the flood leaves its source.

#### 4. Simulation

The attacks described in previous section are simulated using network simulator[18] version 2. The simulation topology is as shown in figure 1. Nodes S1, S2 and S3 are the sender nodes to which TCP senders are attached and nodes D1, D2 and D3 are the destination nodes to which TCP receivers are attached. R1 and R2 are the router nodes configured with drop-tail queues and the maximum queue size is limited to 50 packets. The packet size used in all simulation is 1500 bytes. Reno TCP is used as the TCP sender and a TCP Sink without delayed ACK is used as the TCP receiver. The window size used is 64 packets and the `singledup_` option on TCP sender is turned off to prevent the sender from sending new packets before it receives three duplicate ACKs. In order to generate the attack flow, the ACK generation strategy of TCP sink is modified to force the sender to generate the attack pulse and no modification is done to the TCP sender.

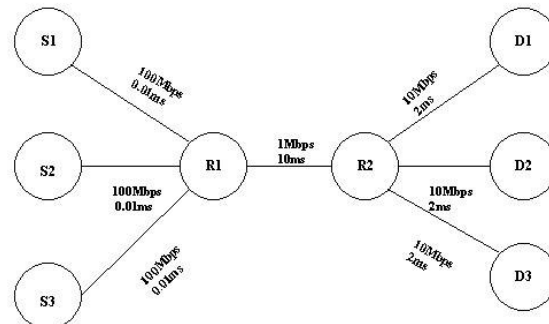


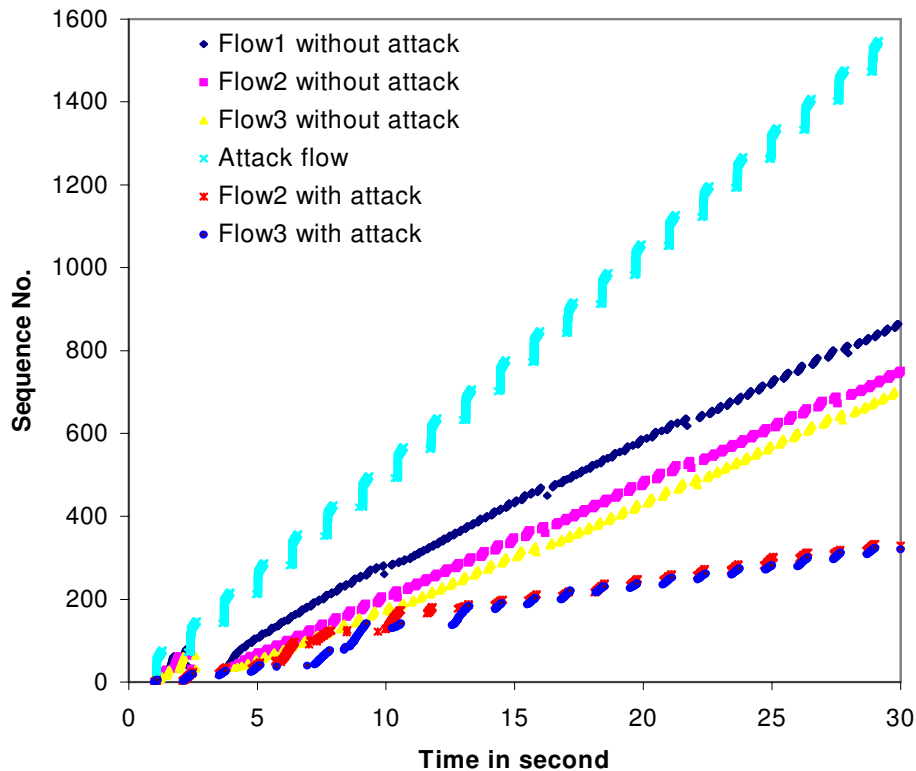
Figure 1: Simulation setup

##### 4.1 Pulse Attack

The attack is first simulated in a simple setup consisting of three TCP flows. Three normal TCP flows, one to each S-D pair, are attached to the nodes and the simulation is run for 30 seconds. The three middle plots in figure 2 show the time versus sequence plot of the flows when all the three are normal flows. Next, the receiver attached to D1 is replaced with the attack flow and the simulation is again done for the same duration. The receiver at D1 generates 50 continuous duplicate ACKs and then 20 optimistic ACKs at a rate of one in each 12 ms period. The uppermost plot gives the sequence plot of the attack flow ( flow from S1 to D1) and the two lower plots represent the sequence plot of the normal flows in presence of the attack flow. Unlike the normal flows, the sequence number of the attack flow does not reflect the number of packets that the flow transmitted through the bottleneck link. This is because some packets of the attack flow are dropped at R1, but are still acknowledged by the receiver at D1. Whenever the attack flow generates the pulse, timeout is imposed on the normal flows causing throughput degradation to normal flows. While the highest sequence number transmitted by the two normal flows attached to S2-D2 and S3-D3 without attack flow is about 700, the value has come down to about 300 in presence of the attack flow.

Figures 3a, 3b and 3c show the transmission pattern of the attacker (the receiver at D1), the transmission pattern of the sender at S1, which is used as the flood source, and the instantaneous output queue occupancy of the router R1 respectively. The values shown in 3a and 3b are the total number of bytes transmitted in each 20 ms duration and the queue occupancy shown in figure 3c is sampled at 10 ms interval. There are two noteworthy observations in the plot. First, the attacker is able to achieve high amplification (about 38 times) to the flood, because for each 40 bytes ACK of the attacker, the sender transmits 1500 bytes packets. The second point is about the period of the DoS flood generated by the sender. Though the ideal period to cause maximum damage is 1 second, it is difficult for the attacker to achieve this. This is because, once the flood generation is started the attacker sends optimistic ACKs and the RTO timer starts only after the optimistic ACKs are stopped. Once the optimistic ACK stops, the sender's retransmission starts after one second and the flood will again be restarted only after one RTT from the retransmission time. This is clear from figure 3b where the flood has a period of about 1.3 second, which is the sum of RTO, optimistic ACK period and one RTT.

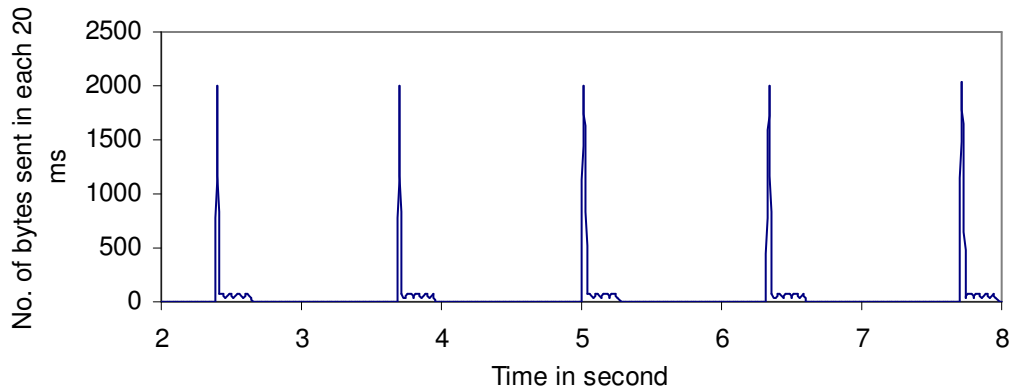
Figure 4 gives an enlarged picture of the transmission pattern of the first two pulses given in figure 2. The attacker, after acknowledging packet 2, sends 50 duplicate ACKs for packet 2. The sender retransmits packet 3 and also sends packet no. 7 to 53. Note that packets up to 7 have been transmitted when the normal ACK to packet 2 reached the sender. The sender exits fast recovery when it receives the first optimistic ACK (which is for 52 here) and sends packet 54. For subsequent optimistic ACKs (53 to 71) the sender sends packets 55 to 77. Packet 72 experiences a timeout and is retransmitted after one second which enables the attacker to induce the second pulse.



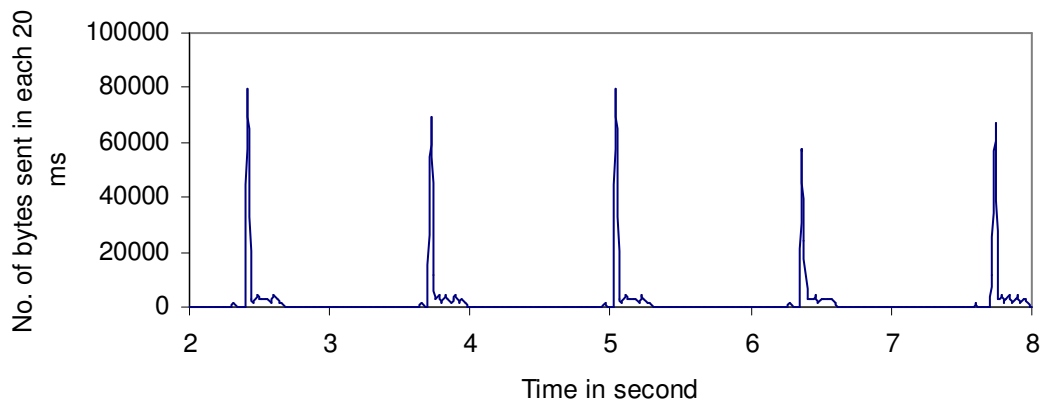
**Figure 2: Sequence no. verses time of TCP flows with and without attack flow**

Next, we investigate the impact of a single attack flow on multiple TCP connections. The number of flows are varied from 1 to 62 by adding flows to S2-D2 and S3-D3 pairs. For each configuration the simulation is done twice: first with normal flow at S1-D1 and then by replacing the normal flow at S1-D1 with attack flow. Figure 5 shows the aggregate throughput of all flows at S2-D2 and S3-D3 with and without the attack when the simulation is run for 30 seconds. As the number of flows increases, the impact of the attack decreases. For example, when the number of flows varies from 1 to 10, the throughput of the flows with attack varies from 35% to 66% of the throughput without the attack. The throughput variation is in the range of 68% to 75% as the flows varies from 12 to 22 and when the

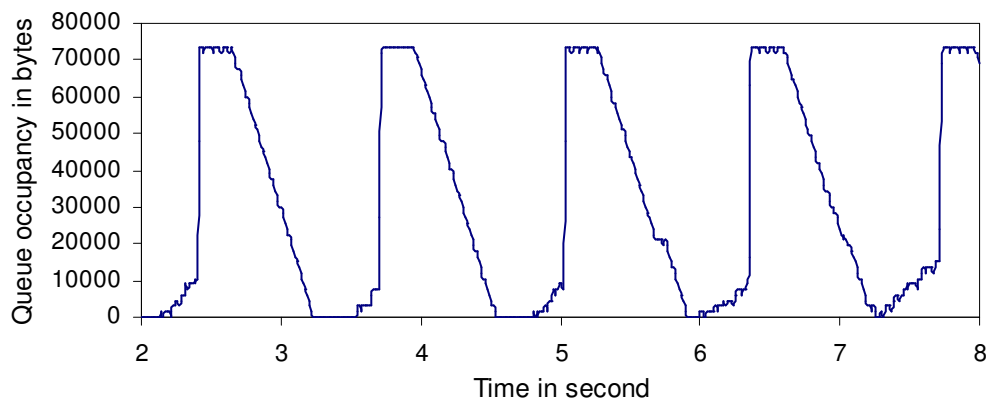
number of flows is above 28 the throughput is more than 80%. The performance degradation is caused by two factors: the first is due to the frequent timeouts imposed on the normal flows and the second is due to the unresponsiveness of the attack flow. When the number of flows is below 20, timeouts are the dominating cause for throughput degradation, whereas when the number of flows is above 20, the total timeouts with and without attacks are almost constant and throughput degradation is due to unresponsiveness of the attack flow.



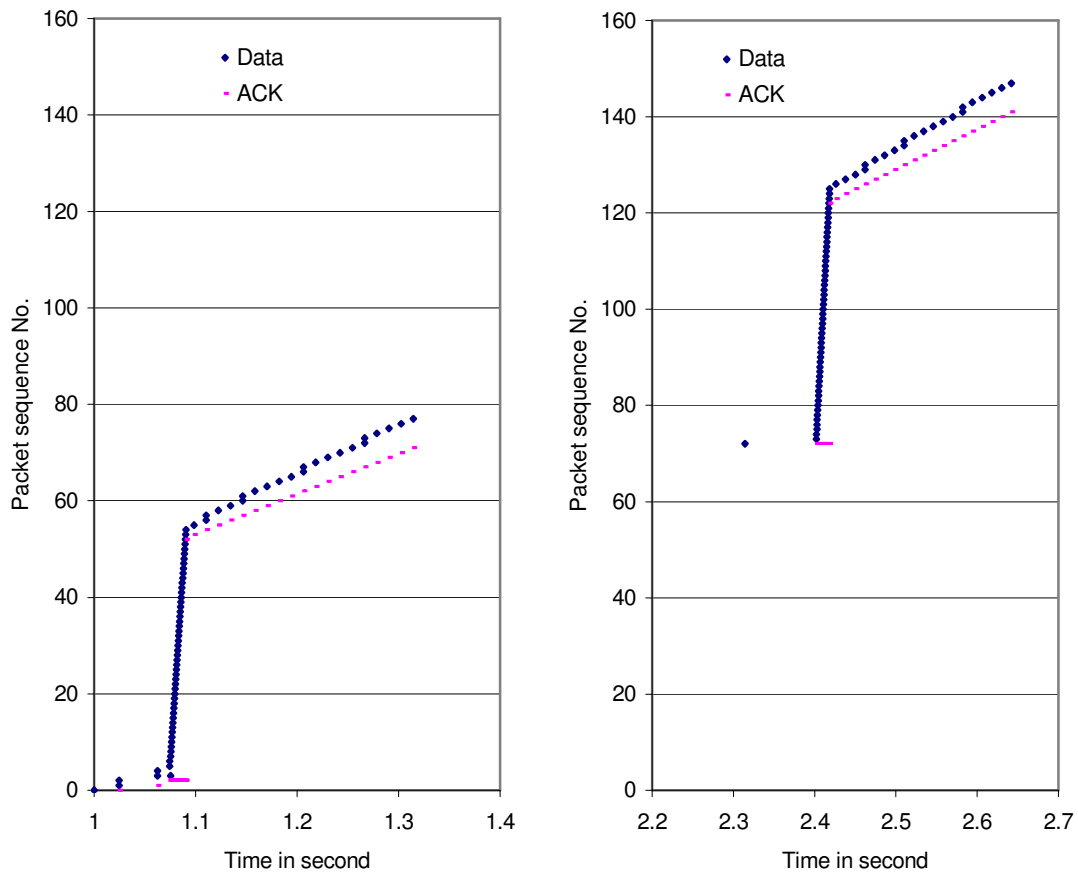
**Figure 3a: Transmission pattern of the attacker during pulse attack**



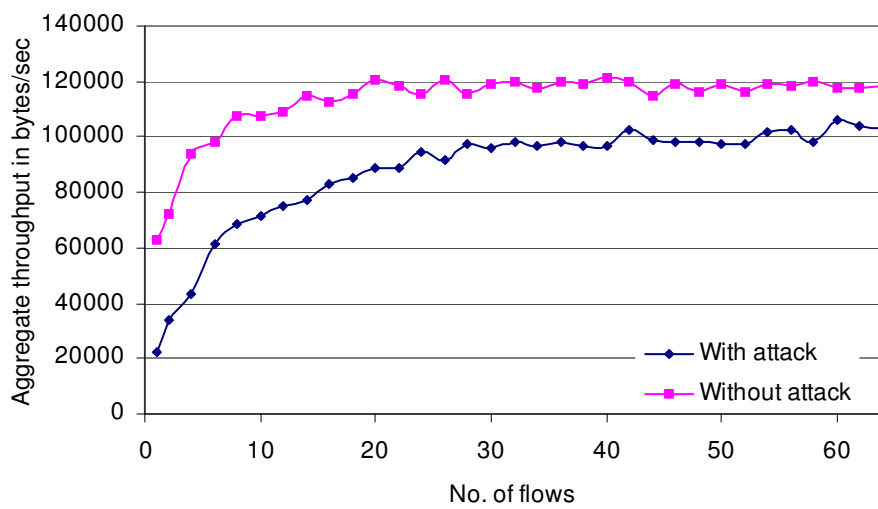
**Figure 3b: Transmission (flood) pattern of the sender during pulse attack**



**Figure 3c: Queue occupancy of the targeted router during pulse attack**



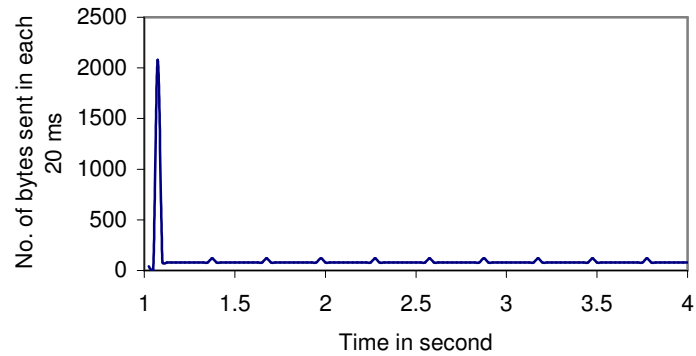
**Figure 4: Sequence no. verses time of the flood traffic of pulse attack**



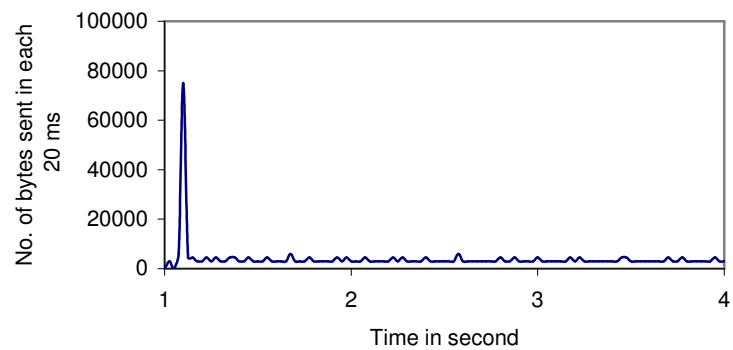
**Figure 5: Throughput variation of flows with and without pulse attack**

## 4.2 Sustained Attack

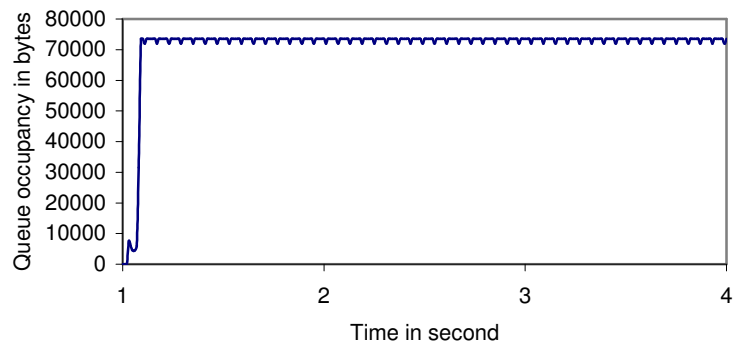
The sustained attack is simulated using the same topology given in figure 1. Two normal TCP flows are attached to S2-D2 and S3-D3 pairs and the attack flow is attached to S1-D1 pair. The ACK generation strategy of the receiver at D1 is modified in such a way that it first generates 50 duplicate ACKs and then generates continuous optimistic ACKs at a rate of one in each 12 ms. The generation of optimistic ACKs is continued for the entire duration of the simulation, which is 30 second as in the previous case. Figures 6a, 6b and 6c show the transmission pattern of the attacker, that of the sender and the queue occupancy of the router R1. The integration and sampling criteria used to generate the plots are same as in figures 3a, 3b and 3c. The high intensity peaks in figure 6a and 6b correspond to the duplicate ACKs of the attacker and the resulting flood generated by the server, and the sustained values are caused by optimistic ACKs. The buffer occupancy of the router remains at its maximum throughout the attack.



**Figure 6a: Transmission pattern of the attacker during sustained attack**



**Figure 6b: Transmission (flood) pattern of the sender during sustained attack**



**Figure 6c: Queue occupancy of the targeted router during sustained attack**

## 5. Detection

In this section we present our detection technique in detail to detect the attacks. The detection system fits well in the framework of an Intrusion Detection System (IDS) and is based on passively monitoring both the inbound and outbound traffic of the monitored network. In rest of the document, the term monitor is used to refer the detection system. A common assumption in most of the IDS is that the monitor has access to the traffic in the monitored network and the same assumption is followed in this paper too[19]. The monitor, in our simulation, is implemented in the form of an agent in ns[18], and it is attached to the router R1 in figure 1.

A common characteristic of the attacks explained earlier is that the output queue occupancy of the WAN interface of the edge router increases suddenly and stays at a high value during the attack. This, of course, cannot be taken as an exclusive signature of attacks because the bursty[20] nature of the Internet traffic can also frequently fill and maintain the router queue. However, if the queue is occupied by genuine packets, the queuing delay caused by buffering the packets will be reflected in the RTT of the packets, where as in the case of attacks, the attacker generates ACKs at high rate before it actually receives the data packets. In fact, for launching an effective DoS attack the ACKs should be generated in such a way that the rate of the resulting flood is of the order of the bandwidth of the targeted network. In such situations, the router queue occupancy increases, but the RTT of malicious packets will be independent of the queue size. We use this characteristic of the attack flows to distinguish them from the normal flows.

The monitor, as explained later, computes and maintains the instantaneous queue occupancy of the output interface of the router. Let  $Q_{inst}$  represent this value. For each outbound TCP packet, the monitor computes a minimum threshold RTT, called  $RTT_{min-thresh}$ , as a function of  $Q_{inst}$ .  $RTT_{min-thresh}$  is defined as the sum of various finite delays introduced to the packet as explained below:

- 1) Queuing delay at the edge router. This is a function of the number of bytes in the queue when the packet reaches there.
- 2) Transmission time of the packet at the link. If  $B$  is the bandwidth in bytes/second, the transmission time for a packet of length  $L$  bytes is  $L/B$ .
- 3) Bi-directional propagation delay of the point-to-point link between the edge router and the ISP router,  $2*p$ , where  $p$  is the one direction propagation delay. The propagation delay is independent of the packet and is constant for a given link.
- 4) Transmission time of the ACK packet at the ISP side of the link.

By considering the default size of 40 bytes for ACK packets, the  $RTT_{min-thresh}$  is computed as follows:

$$RTT_{min-thresh} = (Q_{inst} + L + 40) \times (\frac{1}{B}) + 2p \quad \text{----- (1)}$$

If the observed RTT of the packet falls below the value computed using equation (1) then we conclude that it belongs to a malicious flow.

Equation (1) is derived with the assumption that the access link between the monitored network and its ISP is symmetric with same bandwidth and propagation delay in both directions. It is important to note that the RTT in equation (1) is only a lower bound. The observed value depends on the delay and congestion in the rest of the path that the packet has to travel and its value could be much higher than that in equation (1). The bottom line is that during normal operation, the observed RTT cannot be less than the  $RTT_{min-thresh}$ .

In order to obtain the actual RTT, the monitor records the time at which the TCP packet is transmitted. When the monitor receives an ACK, the time is again recorded and the difference in these two is the observed RTT. Monitor is the reference point in measuring both the time. It is important that the values seen in the TCP timestamp options should not be used to compute the RTT[21]. This is because

the attacker can spoof the timestamps echoed in the ACK packets and force the monitor to compute a wrong observed RTT.

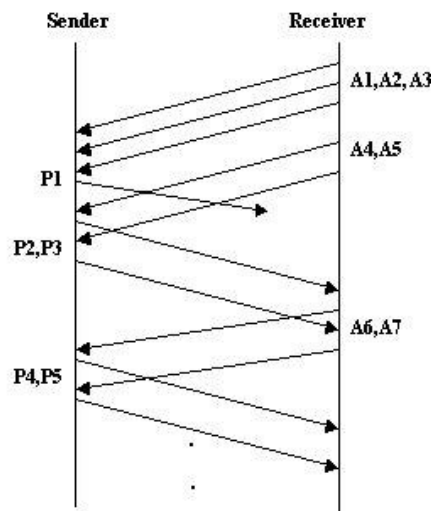
For practical implementation, the processing and storing requirement of the monitor can be brought down by adopting appropriate and more realistic sampling criteria rather than sampling each outgoing TCP packet. For example, the monitor can decide to sample only those TCP packets whose size exceeds some predetermined value. A good choice of the minimum packet size for this purpose could be the minimum of the Maximum Segment Size (MSS) of all the TCP connections which are in established state at any particular time. This is feasible because during the attack, the sender will be sending segments with size of MSS and the attacker do not have any control on the segment size once the connection is established[22]. Also, the monitor could decide to sample only if the Qinst value exceeds some threshold, say for example 50% of the maximum queue size configured on the router. This is sensible because in a real attack, the flood is likely to fill the router buffer.

Qinst is computed and maintained in the same manner as in the case of routers with FIFO queue management. We assume that the monitor knows the maximum buffer size configured on the router interface. For each outbound packet, both TCP and non-TCP, the monitor increments the Qinst by the size of the outbound packet provided that the new value of Qinst does not exceed the maximum buffer size of the router. Next, the value of Qinst has to be decreased to account for the size of outgoing packets and this is done at regular intervals whenever Qinst is greater than zero. In our simulation, we have used a link of 1mbps, which can transmit a packet of 1500 bytes in 12 ms. The Qinst is decremented by 1500 bytes in each 12 ms period.

### 5.1 Detection of Malicious Duplicate Acknowledgments

Detecting malicious duplicate ACKs is difficult because the exact number of duplicate ACKs expected during fast recovery cannot be quantified. This is because, if a retransmitted packet is lost, the sender is likely to receive large number of duplicate ACKs. This characteristic is explained with the help of figure 7.

As soon as the sender receives 3 duplicate ACKs (A1, A2, A3), it retransmits the lost packet (P1) and begins fast recovery. Additional duplicate ACKs (A4, A5) cause the sender to transmit new packets (P2, P3). Suppose the retransmitted packet (P1) is lost and at the same time P2 and P3 are successful in reaching the receiver. Since P1 is lost, P2 and P3 are treated as out of order packets and the receiver generates duplicate ACKs (A6 and A7) and this duplicate ACKs will again cause the sender to send new packets (P4 and P5). This cyclic process will be terminated either when P1 is retransmitted again (after RTO expiry) or when the number of outstanding packets exceeds the *rwnd*. So the challenge is to differentiate duplicate ACK stream triggered due to retransmission loss from those maliciously generated by the attacker. Next we explain our algorithm to distinguish between these two types of duplicate ACKs.

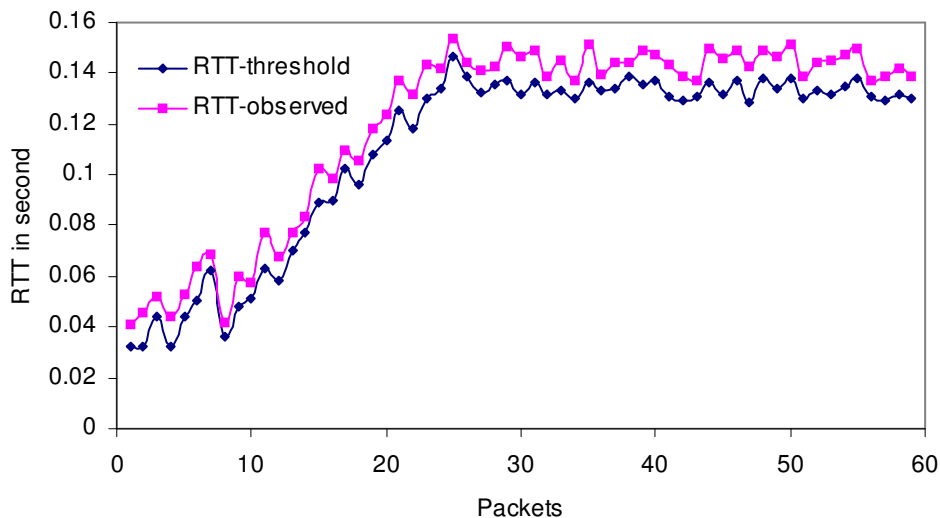


**Figure 7: Duplicate ACK stream due to retransmission packet loss**

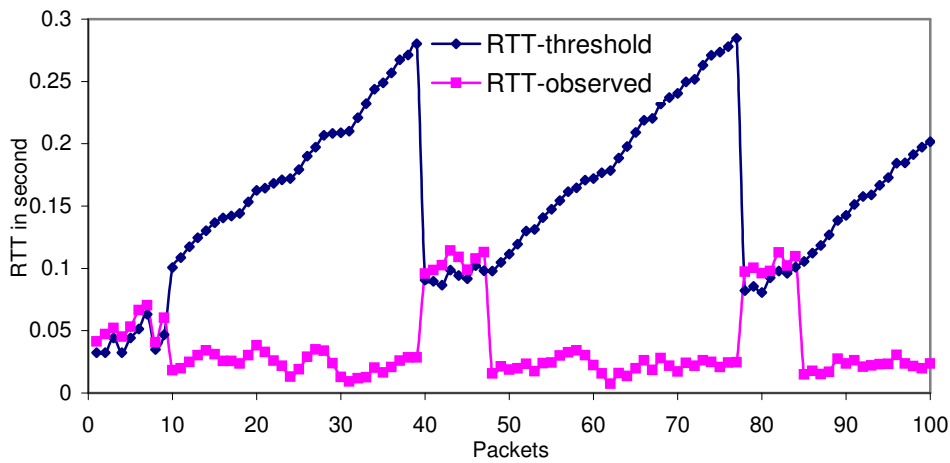
When the monitor detects the first duplicate ACK, it computes the maximum number of duplicate ACKs expected (`max_dupack`) under the assumption that the retransmission, if happens, will not be lost. The monitor records the `RTTthresh` and time of all the packets transmitted in fast recovery. If the number of duplicate ACKs exceeds the `max_dupack`, additional dupACKs are marked suspicious and indexed sequentially starting with one (the first dupACK which exceeds `max_dupack` as 1, the second as 2 and so on). If the suspicious dupacks are the result of a retransmission packet loss, the first marked suspicious ACK will be generated by the first packet sent in fast recovery and the second suspicious ACK by the second packet sent in fast recovery and so on. These suspicious ACKs are used to compute the observed RTT for packets sent in fast recovery and if the observed RTT falls below its threshold value, the suspicious ACKs are identified as malicious ACKs.

The algorithm described above is simulated in two different scenarios using the same topology shown in figure 1. The first simulation depicts the case where duplicate ACKs are generated due to retransmission packet loss. A single TCP flow is set up between node S2 and D2 and the router R2 is modified to drop packet with sequence number 26 and its retransmission. Figure 8a shows the threshold RTT calculated for each packet when they are sent and the corresponding observed RTT measured when the ACKs are received. Packet number 26 is actually the first packet transmitted in fast recovery and packet no. 27 is the second and so on. Observed RTT of packets sent in fast recovery are computed using duplicate ACKs and its value is always greater than the threshold RTT.

Figure 8b shows the simulation result where the duplicate ACKs are generated by the attacker. The simulation uses the same setup as in the previous case except that the TCP receiver is replaced with the attacker. The attacker first behaves like a normal receiver, then abruptly changes its behaviour and generates a large number of duplicate ACKs and then returns to normal operation. While receiving packet number 1 to 9 the attacker behaves as a normal receiver and acknowledges up to and including packet 9. The observed RTT for these packets are greater than the threshold RTT. The attacker then changes its behaviour and generates duplicate ACKs for 9 and the sender begins fast recovery. Packet number 10 to 40 are sent in fast recovery in response to the malicious duplicate ACKs. All these packets have observed RTT smaller than the threshold RTT. The attacker then repeats the process. Note that retransmitted packets and packets sent but not acknowledged before starting fast recovery are not shown in figure 8b.



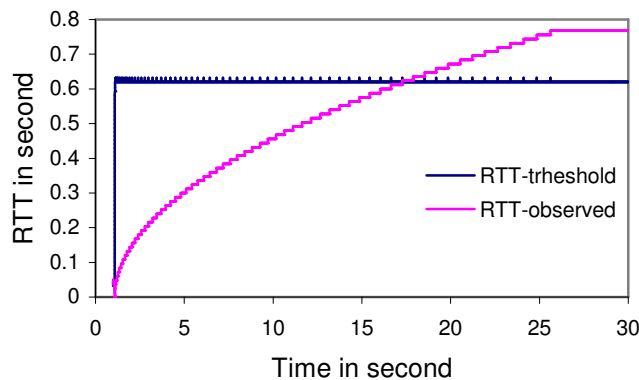
**Figure 8a: Threshold and observed RTT when duplicate ACKs are generated due to retransmission loss**



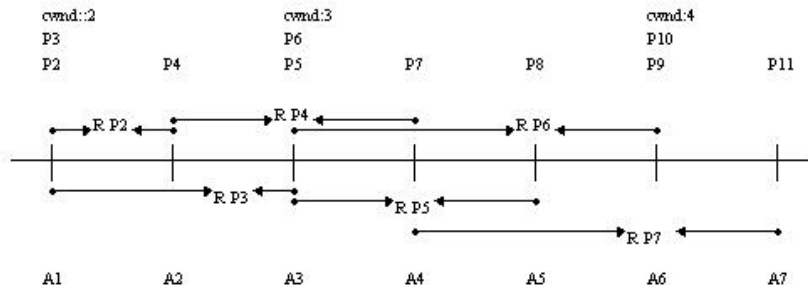
**Figure 8b: Threshold and observed RTT for malicious duplicate ACKs**

## 5.2 Detection of Malicious Optimistic Acknowledgements

Optimistic ACKs spoofed for launching the attacks are differentiated from normal ACKs whenever the observed RTT of packets falls below the minimum threshold RTT. Figure 9 shows the observed and threshold RTT versus time plotted during an attack in which the attacker first sends duplicate ACKs to fill the router buffer and then continuously generates optimistic ACKs at a rate of one in each 12 ms. An important observation is that during the attack, the observed RTT increases with time. Around 17 second, the observed value exceeds the threshold value and it reaches the stable value of 768 ms after 25 seconds. This is because the sender is in congestion avoidance phase in which the *cwnd* and hence the number of outstanding packets increases in response to the optimistic ACKs. This is further explained below with the help of figure 10.



**Figure 9: Threshold and observed RTT for optimistic ACKs**



**Figure 10: Growth of *cwnd* and observed RTT of TCP sender while receiving optimistic ACKs**

In figure 10, the horizontal line represents time and each tick mark corresponds to 12ms. A1, A2, etc. are the optimistic ACKs, P1, P2, etc. are the data packets sent and RP represents the observed RTT of packet P. The growth of *cwnd* as ACKs arrive is also marked. Assume that when the first ACK, A1, arrives, the *cwnd* is 2 and P2 and P3 are transmitted. ACK to P2 (A2) comes 12 ms after P2 is transmitted (so P2 has an RTT of 12 ms). However, the ACK for P3 (A3) comes 24 ms after P3 is sent. Similarly, P7 is acknowledged 36 ms after it is transmitted. This growth in RTT continues until the *cwnd* becomes equal to *rwnd*. In the simulation, *rwnd* is set to 64 and therefore the maximum observed RTT is 768ms (64\*12).

## 6. Conclusion

In this paper, we have explored the possibility of a class of Denial-of-Service attacks where an internal machine in the targeted network is exploited as the flood source without compromising the machine. We studied two different behaviours of the attacker which result in two disastrous flood patterns. Through simulation, we have evaluated the potential negative impacts of these attacks on the targeted system. We have also proposed an Intrusion Detection System to detect the attacks by passively monitoring the targeted network.

## References

- 1) S. Savage, N. Cardwell, D. Wetherall and T. Anderson, "TCP congestion control with Misbehaving receiver", Computer Communication Review, 29(5), pp.71-78, October 1999.
- 2) B. Braden, D. Clark, C. J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, Fiberlane, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998
- 3) V. Jacobson , "Congestion avoidance and control", In proceedings of SIGCOMM, page 314-329, 1988
- 4) V. Jacobson, "Modified TCP congestion Avoidance Algorithm", Technical report LBL, April 1990
- 5) M. Allman, V. Paxson, W. Stevens, "TCP Congestion Control", RFC 2581, April 1999
- 6) H. Balakrishnan, V. N. Padmanabhan, S. Seshan, M. Stemm, R. H. Katz, "TCP Behavior of a Busy Internet Server: Analysis and Improvements", In Proceedings of IEEE Infocom, March 1998
- 7) M. Allman, H. Balakrishnan, and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", RFC 3042, January 2001.
- 8) V. Paxson, M. Allman "Computing TCP's Retransmission Timer", RFC 2988, November 2000
- 9) K. Fall and S. Floyd, "Simulation based Comparison of Tahoe, Reno, and SACK TCP", Computer communication Review, July 1996
- 10) J. Postel, "User Datagram Protocol", RFC 768, August 1980

- 11) D. Dittrich, The DoS Project's "trinoo" distributed denial of service attack tool, <http://staff.washington.edu/dittrich/misc/trinoo.analysis>.
- 12) D. Dittrich, The "Tribe Flood Network" distributed denial of service attack tool <http://staff.washington.edu/dittrich/misc/tfn.analysis>.
- 13) C. Schuba, I. Krsul, M. Kuhn, E. Spafford, A. Sundaram, and D. Zamboni. "Analysis of a Denial of Service Attack on TCP", Proc. of IEEE Symposium on Security and Privacy, 1997
- 14) S. Floyd, V. Jacobson, "Random Early Detection Gateways for congestion avoidance" Transactions on Networking, August 1993
- 15) S. Floyd and K. Fall "Promoting the Use of End-to-End Congestion Control in the Internet", IEEE/ACM Transactions on Networking, August 1999
- 16) R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker  
"Controlling High Bandwidth Aggregates in the Network" *ACM SIGCOMM CCR*, Vol 32, No. 3, July 2002
- 17) V. Paxson, "An analysis of using reflectors for distributed denial-of-service attacks", ACM Computer Communication Review, 31 (3), July 2001.
- 18) network simulator version 2, <http://www.isi.edu/nsnam/ns>
- 19) V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time", Computer Networks, 31(23-24), pp. 2435-2463, 14 Dec. 1999
- 20) V. Paxson and S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modelling", IEEE/ACM Trans. Networking, Vol 3, No.3, pp.226-244, June 1995
- 21) V. Jacobson, R. Braden, D. Borman, "TCP Extension for High Performance", RFC 1323
- 22) J. Postel, "Transmission Control Protocol", RFC 793, September 1981
- 23) J. Padhye and S. Floyd, "Identifying the TCP behaviours of Web Servers", Technical Report 01-002, ICSI, 2001
- 24) J. Padhye and S. Floyd, "On Inferring TCP Behavior", ACM SIGCOMM, August 2001
- 25) A. Kuzmanovic and E.W. Knightly, "Low-Rate TCP-Targeted Denial of Service Attacks", in proceedings ACM SIGCOMM, August 2003