

Analysis and Detection of a Denial-of-Service Attack Scenario generated by TCP Receivers to Edge Network

V. Anil Kumar¹ and Dorgham Sisalem²
(anil@cmmacs.ernet.in, sisalem@fokus.fhg.de)

1 CSIR Centre for Mathematical Modelling and Computer Simulation, Bangalore, India

2 Fraunhofer Institute for Open Communication Systems, Berlin, Germany

Abstract-The success of the end-to-end congestion control in TCP mainly depends on the co-operation and volunteer participation of the end systems in the congestion control process. 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 the analysis and detection of a special type of flood-based DoS attack scenario, where an internal TCP server in a network is compelled to generate high volume traffic to flood its own network. We show that an attacker, by exploiting the vulnerabilities of TCP congestion control algorithms to duplicate and optimistic acknowledgement spoofing, can successfully turn a TCP server to a flood source without compromising the server. We study the potential negative impact of the attack on an edge network, which connects an organisational LAN to the Internet using a router with FIFO queue management. Our simulation results show that such an attack is highly disastrous and powerful enough to virtually detach the targeted network from the Internet. We extend our work by presenting a simple but effective method for detecting the attack by passively monitoring the inbound and outbound traffic of the targeted network. The detection is achieved by differentiating malicious stream 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 from the end points.

I. 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 a special type of DoS attack 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 attack described here exploits the transmission power of an internal machine in the targeted network to flood its own network without compromising the machine. We show 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 highly disastrous DoS flood. The potential danger arising from such a situation is explored and the results presented in this paper show that such attacks are feasible.

We study a tactical behaviour of the attacker which in turn constitute the attack, analyse the subsequent flood pattern generated by the internal machine, and quantify the impact of the flood on the Internet access router of the targeted network. The flood, which is generated by spoofing duplicate and optimistic ACKs[1], has the characteristic that it can instantly fill the output buffer associated with the external (WAN) interface of the targeted router and also maintain the buffer in the filled status for the entire duration of the attack. The router with filled output buffer has no option other than to drop any further packets traversing from the edge network to the Internet through the buffer. As a result of this, almost no genuine outbound traffic can go out of the edge network and this will virtually detach the edge network from the Internet for the entire duration of the attack.

Finally, we present our intrusion detection approach to distinguish the flood from normal traffic by detecting maliciously crafted duplicate and optimistic ACKs for generating the flood. A noticeable symptom of the attack is that the buffer occupancy of the router in the targeted network increases abruptly, whereas the Round Trip Time (RTT) of the packets belonging to the flood does not increase accordingly. Based on this observation, we define a minimum threshold RTT for outgoing packets in such a way that under normal condition a packet cannot be acknowledged before the defined RTT period elapses. The threshold RTT for this is computed as a function of the instantaneous buffer occupancy of the router, and attack is detected whenever the observed RTT falls below the threshold.

In this paper, we use the following terminologies. The term ‘attack flow’ denotes the TCP flow setup by the attacker, the term ‘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.

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 attack in detail using Reno’s congestion control. However, by considering the fact that Reno is being constantly replaced[3][4] by newer TCP variants like New-Reno and SACK[5], we briefly evaluate the threat level and the effectiveness of the attack in the cases of these TCPs also. In the same section, we also compare the attack with other widely seen DoS attacks. Section 4 covers the simulation of the attack and presents the simulation results. Section 5 describes our intrusion detection approach to detect the attack and finally in section 6, we conclude the paper.

II. CONGESTION CONTROL AND RETRANSMISSION POLICY IN TCP

Congestion Control in TCP consists of four intertwined algorithms, slow start[6], congestion avoidance,[6] fast retransmit and fast recovery [7][8]. The sender maintains a state variable called congestion window ($cwnd$), which is the sender side limit on the number of packets it can transmit before it receives an ACK. Another variable, $ssthresh$, decides whether the sender should operate in slow start or congestion avoidance. 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. 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. 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, which ever 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.

III. THE ATTACK

In this section, we first describe the tactical behaviour of the attacker through which it turns an internal server, based on Reno TCP, in the targeted network to a flood source and then evaluate the feasibility of the attack when the server is based on New-Reno and SACK variants of TCP. Finally, we also compare the attack with other widely seen DoS attacks.

A. Description

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 consecutive duplicate ACKs with minimum possible inter-packet delay so that the stream of duplicate ACKs will force 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. If the number of duplicate ACKs sent by the attacker is slightly more than the buffer size of the targeted router, the resulting flood can instantly fill the router buffer. 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.

Once the router buffer is filled, the next stage of the attack is to maintain the buffer in the filled status. This is possible only if new packets are added to the buffer at the same rate as packets are dequeued from the buffer. The attacker achieves this goal by shifting the attack flow from the fast recovery mode to the congestion avoidance mode by sending optimistic ACKs. The attacker estimates the highest packet that the server has sent in fast recovery and then generates optimistic ACK to that packet. Optimistic ACKs are sent at constant interval in

such a way that the ACK number of each ACK packet is incremented by one packet (Sender's Maximum Segment Size bytes of the TCP sender in the case of byte level granularity). For example, if i^{th} ACK packet has an ACK number of n , then $(i+1)^{\text{th}}$ ACK packet will have an ACK number of $n+1$. The process of generating optimistic ACKs is further explained below with the help of an example.

Consider the usual 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 3rd duplicate ACK $cwnd$ becomes 5, $[\max(cwnd/2, 2) + 3]$. The 50th 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 the first optimistic ACK and clear the server's account on the number of outstanding packets. Additional optimistic ACKs are sent at constant interval. For example, if the bandwidth of the access link of the targeted network is 1Mbps, the attacker has to send optimistic ACKs at a rate of one ACK in each 12 ms. This is because each ACK 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 a 1Mbps link.

Though the attacker can estimate the sequence number of the highest packet sent in fast recovery 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.

B. Attack in General Scenario

Here, we evaluate the success of the attacker in generating the flood when the targeted system is based on New-Reno or SACK TCP. The flood pattern of New-Reno 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[5] (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. This is because SACK is implemented as an option in TCP and it is always negotiated by the client at the time of establishing the connection. Hence the attacker can easily turn off SACK option at the time of establishing the connection. From the above points we conclude that most of the widely deployed TCP servers could be exploited as a flood source.

C. Comparison

The attack is significantly different from widely seen flood based DoS attacks like Trinoo[9], TFN[10], SYN-flood[11] etc. In these attacks, the flood sources are located outside the targeted network and the attacker or a set of machines on behalf of the attacker (in the case of Distributed DoS) generates high intensity flood and directs it towards the victim. The success of such attacks depends on a major portion of the flood reaching the target. Intermediate routers with advanced features like RED[12], per-flow scheduling[13] or aggregate based flood control[14] etc. may deliberately drop a portion of the flood and this could reduce the intensity of the attacks. The attack described in this paper will be more effective as the flood source is located inside the targeted network and hence the flood will not be regulated by the intermediate routers. Another notable characteristic of the attack is high flood amplification. Typically, the attacker sends ACK packets of size 40 bytes and gets it amplified to data packets of size 1500 bytes. This results an amplification of as high as about 38 times.

IV. SIMULATION

The attack is simulated in network simulator[15] version 2 using the topology shown in Fig. 1. Nodes S1-S3 are the sender nodes to which TCP senders are attached and nodes D1-D3 are the destination nodes to which TCP receivers are attached. R1 and R2 are the routers configured with drop-tail queues and queue size is limited to 50 packets. The packet and window sizes used for the simulation are 1500 bytes and 64 packets respectively. Reno TCP is used as the TCP sender and TCP Sink without delayed ACK is used as the receiver. The attack flow is realised by modifying the ACK generation strategy of the TCP sink and no modification is done to the TCP sender.

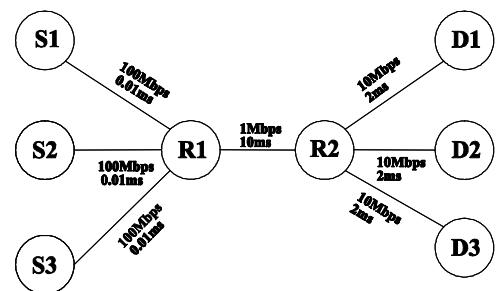


Fig. 1. Simulation setup

We study the impact of the attack on both TCP and UPD flows emerging from the targeted network. First, we focus on 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 two upper plots in Fig. 2 show the time versus packet sequence (no. of packets transmitted by the sender) plot of the two normal flows attached to S2-D2 and S3-D3 pairs when the flow from S1 to D1 is also a normal TCP flow. Next, we replace the receiver at D1 with the attack flow, which generates 50 continuous duplicate ACKs and then optimistic ACKs at a rate of one ACK in each 12 ms period for a duration of 30 seconds. This represents the attack scenario and the two lower plots in Fig. 2 represent the sequence plot of the normal flows in presence of the attack flow. The impact of the attack on normal flows is clearly visible in Fig. 2. Both the normal flows experience high packet loss, which lead them to repeated timeouts and as a result of this the flows cannot progress.

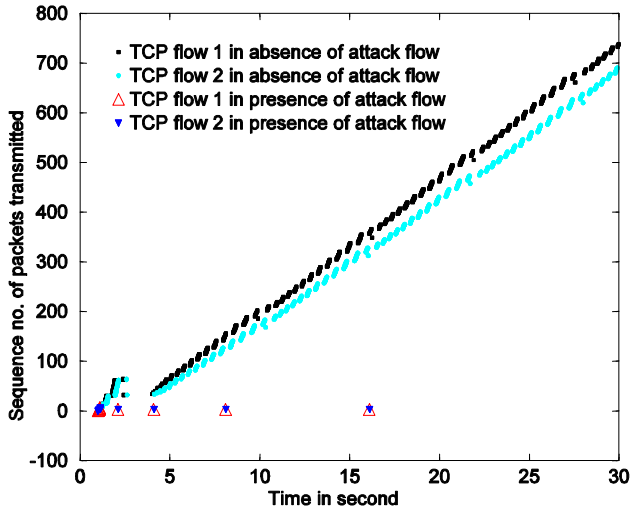


Fig. 2. Sequence no. verses time of TCP flows in presence and absence of the attack flow

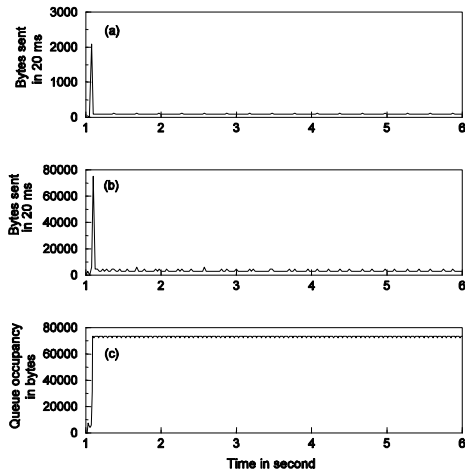


Fig. 3. (a) Transmission pattern of the attacker, (b) Transmission (flood) pattern of the sender and (c) Queue occupancy of the targeted router during sustained attack

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 Fig. 3c is sampled at 10 ms interval. The high intensity peaks in Fig. 3a and 3b 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. There are two noteworthy observations in the plot. First, the intensity of the actual flood is about 38 times the amount of data transmitted by the attacker. Second, the buffer occupancy of the router remains at its maximum throughout the attack.

Next, we investigate the impact of the attack on UDP flows. Two UDP flows are attached to S2-D2 and S3-D3 pairs and a normal TCP flow is attached to S1-D1 pair. We compute the fraction of UDP packets dropped at router R1 as the ratio of the number of UDP packets dropped by R1 per second to the number of UDP packets received by R1 per second. The two lower plots in Fig. 4 show this ratio when there is no attack. The TCP flow from S1 to D1 is then replaced with the attack flow and the two upper plots show the fraction of UDP packets dropped during the attack. The impact of the attack is reflected as sustained and high packet drop ratio.

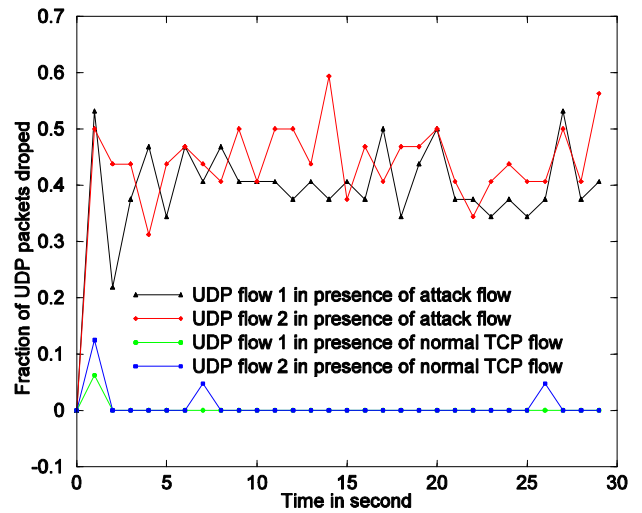


Fig. 4. Fraction of UDP packets dropped per second in presence and absence of the attack flow

V. DETECTION

In this section, we present our Intrusion Detection System (IDS) to detect the attack by passively monitoring the inbound and outbound traffic of the monitored network. In rest of the paper, the term monitor is also used to refer the IDS. The monitor, in our simulation, is implemented in the form of an agent in ns[15], and it is attached to the router R1 in Fig. 1.

A noticeable symptom of the attack 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 the attack

as the bursty[16] Internet traffic can also occasionally 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 attack, 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 resulting flood rate 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 flow to distinguish it from 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 the following finite delays introduced to the packet: (1) Queuing delay at the edge router. This is a function of the number of bytes already there in the queue when the packet reaches the queue. (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, $2 \cdot p$, where p is the one direction propagation delay. (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:

$$RTT_{Min-thresh} = (Q_{inst} + L + 40) \times (1/B) + 2p . \quad (1)$$

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) makes the assumption that the access link is symmetric with same bandwidth and propagation delay in both directions. Also, the RTT in equation (1) is only a lower bound. The actual RTT 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. The values seen in the TCP timestamp option should not be used to compute the RTT[17] because the attacker can manipulate the timestamps and force the monitor to compute a wrong observed RTT.

Q_{inst} 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 Q_{inst} by the size of the

outbound packet provided that the new value of Q_{inst} does not exceed the maximum buffer size of the router. Q_{inst} is decremented to account for the size of outgoing packets and this is done at regular intervals. In our simulation, we have used a link of 1Mbps, which can transmit a packet of 1500 bytes in 12 ms and hence Q_{inst} is decremented by 1500 bytes in each 12 ms period.

A. Detection of Malicious Duplicate Acknowledgments

Detecting malicious duplicate ACKs is difficult because the exact number of genuine 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 Fig. 5.

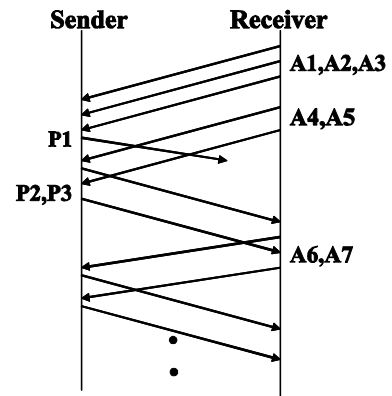


Fig. 5. Duplicate ACK stream due to retransmission packet loss

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.

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 RTT_{thresh} 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.

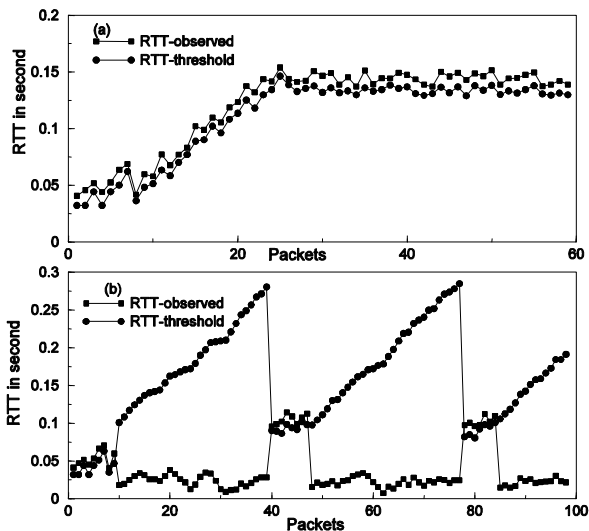


Fig. 6. (a) Threshold and observed RTT when duplicate ACKs are generated due to retransmission loss and (b) Threshold and observed RTT for malicious duplicate ACK

The algorithm described above is simulated in two different scenarios using the same topology shown in Fig. 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. Fig. 6a 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.

Fig. 6b shows the simulation result where the duplicate ACKs are generated by 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. Packets 10 to 40 are sent in fast recovery in response to the malicious duplicate ACKs and their

observed RTT is less than the threshold RTT. Note that retransmitted packets and packets sent but not acknowledged before starting fast recovery are not shown in Fig. 6b.

B. Detection of Malicious Optimistic Acknowledgements

Fig. 7 shows the observed and threshold RTT verses time plotted during an attack in which the attacker first sends duplicate ACKs and then optimistic ACKs at a rate of one in each 12 ms. Though the observed RTT is less than the threshold RTT during the initial phase of the attack (so the attack can be detected), 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 Fig 8.

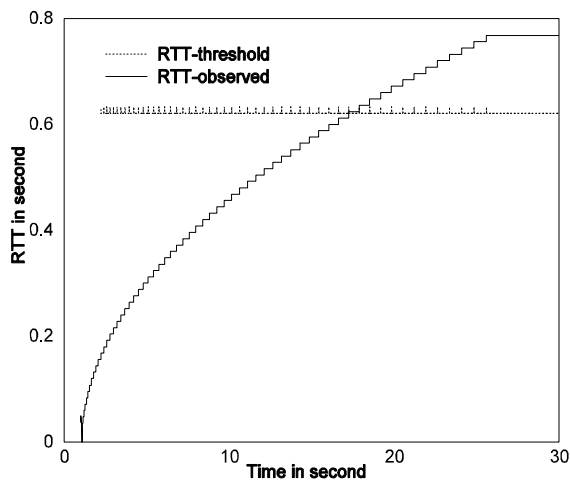


Fig. 7. Threshold and observed RTT for optimistic ACKs

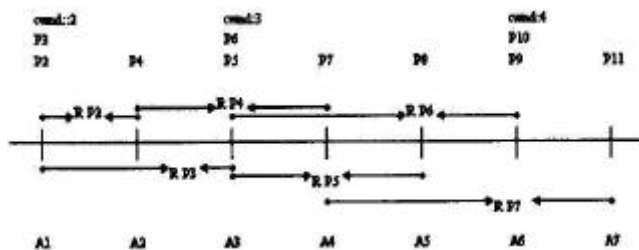


Fig. 8. Growth of *cwnd* and observed RTT of TCP sender while receiving optimistic ACKs in congestion avoidance mode

In Fig. 8, the horizontal line represents time and each tick mark corresponds to 12ms. A1, A2, etc. are the optimistic ACKs, P2, P3, 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 (64x12).

VI. CONCLUSION

In this paper, we have explored the possibility and threat level of a special type of Denial-of-Service attack where an internal machine in the targeted network is exploited as the flood source. We studied a tactical behaviour of the attacker which constitute the attack and analysed the subsequent flood traffic generated by the internal machine. We have evaluated and quantified the potential negative impacts of the attack on the targeted system. Our simulation results revealed that the attack is highly disastrous and powerful enough to virtually detach the targeted network from the Internet. We have also proposed an Intrusion Detection System to detect the attack 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] J. Padhye and S. Floyd, "Identifying the TCP behaviours of Web Servers", Technical Report 01-002, ICSI, 2001.
- [4] J. Padhye and S. Floyd, "On Inferring TCP Behavior", *ACM SIGCOMM*, August 2001.
- [5] K. Fall and S. Floyd, "Simulation based Comparison of Tahoe, Reno, and SACK TCP", *Computer communication Review*, July 1996.
- [6] V. Jacobson, "Congestion avoidance and control", In proceedings of *SIGCOMM*, page 314-329, 1988.
- [7] V. Jacobson, "Modified TCP congestion Avoidance Algorithm", Technical report LBL, April 1990.
- [8] M. Allman, V. Paxson, W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.
- [9] D. Dittrich, The DoS Project's "trinoo" distributed denial of service attack tool, <http://staff.washington.edu/dittrich/misc/trinoo.analysis>.
- [10] D. Dittrich, "The Tribe Flood Network distributed denial of service attack tool" <http://staff.washington.edu/dittrich/misc/tfn.analysis>.
- [11] 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.
- [12] S. Floyd, V. Jacobson, "Random Early Detection Gateways for congestion avoidance" *Transactions on Networking*, August 1993.
- [13] S. Floyd and K. Fall "Promoting the Use of End-to-End Congestion Control in the Internet", *IEEE/ACM Transactions on Networking*, August 1999.
- [14] 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.
- [15] network simulator version 2, <http://www.isi.edu/nsnam/ns>.
- [16] 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.
- [17] V. Jacobson, R. Braden, D. Borman, "TCP Extension for High Performance", RFC 1323.