On a New Class of Pulsing Denial-of-Service Attacks and the Defense

Xiapu Luo and Rocky K. C. Chang
Department of Computing
The Hong Kong Polytechnic University
Kowloon, Hong Kong, SAR, China
{csxluo,csrchang}@comp.polyu.edu.hk

Abstract

In this paper we analyze a new class of pulsing denial-of-service (PDoS) attacks that could seriously degrade the throughput of TCP flows. During a PDoS attack, periodic pulses of attack packets are sent to a victim. The magnitude of each pulse should be significant enough to cause packet losses. We describe two specific attack models according to the timing of the attack pulses with respect to the TCP's congestion window movement: timeout-based and AIMD (additive-increase-multiplicative-decrease)-based. We show through an analysis that even a small number of attack pulses can cause significant throughput degradation. The second part of this paper is a novel two-stage scheme to detect PDoS attacks on a victim network. The first stage is based on a wavelet transform used to extract the desired frequency components of the data traffic and ACK traffic. The second stage is to detect change points in the extracted components. Through both simulation and testbed experiments, we verify the feasibility and effectiveness of the detection scheme.

1 Introduction

Starting from the most well-known denial-of-service (DoS) attacks in February 2000 against a number of very popular web sites, such as Yahoo, Amazon, and eTrade, DoS attacks continue to plague the Internet. The most recent one is the attack against Akami's DNS servers, which disrupted their services for hours. Unlike other system-specific attacks, DoS attacks are more generic in nature. Therefore, their impact can be very significant in scope and damage.

Conventional DoS attacks are flooding-based [8]. That is, an attacker sends out an unusually large number of packets to a victim via a single host or multiple infected hosts. These attack packets either exhaust the victim's bandwidth, e.g. DNS reply flooding, or exhaust the victim's system resources, e.g. SYN flooding. Based on an anomalous rise in the traffic rate, these flooding-based attacks can be easily detected at the victim's side. The response to the detection is usually to ask the upstream provider to drop the attack packets inscribed with the victim's address. Since the attack packets usually contain spoofed source IP addresses, various detection mechanisms based on other signatures have been proposed, such as the statistical distribution of source addresses [14], source and destination addresses [18], and the TTL values [6].

In this paper, we consider a new generation of DoS attacks, called *pulsing DoS attacks* (PDoS). PDoS attacks are not entirely new, having been reported by Asta Networks in 2001 [9]. Based on a six-month period of analyzing the traffic in the Internet2 Abilene backbone, they have discovered the presence of *pulsing zombies*. Instead of generating a flood of attack packets, these zombies sent out short bursts of attack packets to a victim. Kuzmanovic and Knightly have recently proposed a low-rate TCP-based attack that also involves sending pulses of packets [13].

In this paper, we propose and analyze a class of PDoS attacks against TCP-based applications. This class of new attacks can be further categorized into *timeout-based attacks* and *AIMD-based attacks* (AIMD stands for Additive Increase and Multiplicative Decrease), depending on the timing of the attack pulses with respect to the TCP's congestion window movement. As will be explained later, there are also variants within each category. It turns out that the low-rate attack proposed in [13] is a specific case of the timeout-based attacks.

The PDoS attacks are much more sophisticated and effective than the traditional flooding-based attacks. First of all, by adjusting the parameters in the attack tool, different levels of damage can be launched. On one extreme, the result can be the same as the flooding-based attacks. On the other, it can create a *degradation-of-service attack*, i.e., the victim's performance will be degraded but

not to the extent of being denied of service. Second, the amount of attack traffic required to achieve a DoS attack is also much lower than in flooding-based attacks. Consequently, the PDoS attack can elude the detection methods designed for flooding-based attacks. Finally, the number of attack sources can be very small in a PDoS attack, as compared with a conventional distributed DoS (DDoS) attack. Therefore, the packet fields in the attack packets can be set with correct values in order to escape the detection methods proposed in [14, 18, 6].

We believe that this work is the first to formally address and analyze PDoS attacks. Therefore, our contributions is two-fold. First, we identify and formally describe a new class of PDoS attacks against TCP-based applications. We also model them and analyze their properties and effectiveness. Second, we propose a novel two-stage scheme to detect the attacks on the victim's side. The methods of detection are designed based on two key observations: (1) the PDoS attack causes the rate of incoming traffic to fluctuate more severely than would normally be the case, and (2) the outgoing TCP ACKs will decline after an attack has been launched. We have employed a wavelet transform in the first stage to extract relevant information, which is then fed into a nonparametric CUSUM algorithm for detecting abrupt changes.

The rest of the paper is organized as follows. In section 2, we describe and analyze the new class of PDoS attacks. In section 3, we describe a two-stage detection scheme for the PDoS attacks. In section 4, we evaluate the performance of the detection system based on the `ns-2` simulation and a testbed. Section 5 concludes this paper.

2 Modelling and analyzing of the PDoS attacks

One can view the flooding-based attack as a brute-force attack, which exploits the finiteness of network and system resources. However, the PDoS attack is more sophisticated in the sense that it exploits a transport protocol's congestion control mechanism. There are two main mechanisms in a typical end-to-end congestion control algorithm. The first is the generation of a congestion signal that serves to notify the sender of possible congestion. The second is the sender's response to the receipt of such a congestion signal. Table 1 summarizes the mechanisms used in TCP.

In essence, a PDoS attacker generates a sequence of *false congestion signals* to a TCP sender using attack pulses, so that the sender's *cwnd* is constrained to a low value. Therefore, the magnitudes of the attack pulses must be significant enough to cause packet drops in a router. We formally model the sequence of attack pulses by using the following:

$$\mathbb{A}(T_{Extent}(n), S_{Extent}(n), T_{Space}(n), N),$$

where,

- N is the total number of pulses sent during an attack.
- $T_{Extent}(n), n = 0, 1, \ldots, N-1$, is the duration of the nth attack pulse.
- $S_{Extent}(n), n = 0, 1, \ldots, N-1$, determines the shape of the nth attack pulse. If $S_{Extent}(n)$ is a constant, the attack pulse is rectangular.
- $T_{Space}(n), n = 0, 1, \ldots, N-2$, measures the time between the end of the nth attack pulse and the beginning of the $(n+1)$th attack pulse. If $T_{Space}(n) = 0, \forall n$, the PDoS attack is the same as a flooding-based attack.

Back to Table 1, a TCP sender's response to the recipient of three duplicate ACKs is generally known as a *additive-increase, multiplicative-decrease (AIMD) algorithm*. Although TCP is the prime target for such PDoS attacks in the Internet today, it is useful to examine more general AIMD algorithms in a similar manner as in [24]. That is, we denote an AIMD algorithm by $AIMD(a, b)$, $a > 0$, $1 > b > 0$. In this general AIMD algorithm, a sender will decrease its *cwnd* from W to $b \times W$ whenever it enters the fast recovery state, and then it will increase its *cwnd* from W to $W + a$ per round-trip time (RTT) until receiving another congestion signal.

Depending on the type of the false congestion signal received, a PDoS attack can force a victim TCP sender to frequently enter the timeout state or to frequently enter the fast recovery state. The result is a persistently low value of *cwnd*, which translates into a very low throughput for the victim TCP connection. Accordingly we classify the PDoS attacks into *timeout-based attacks* and *AIMD-based attacks*, to be presented next.

2.1 Timeout-based attacks

In a timeout-based attack, the attack pulses are severe enough to cause a victim TCP sender to frequently enter the timeout state. Without receiving (or without a sufficient number of) ACKs, the sender eventually timeouts and the retransmission timeout value (RTO) is computed by Eq. (1) [19], as follows:

$$RTO = max\{RTO_{min}, SRTT + max(G, 4 \times VRTT)\}, \quad (1)$$

where RTO_{min}, the lower bound on RTO, is recommended to be 1 second for the purpose of avoiding spurious retransmissions [2], and G is the clock granularity.

Table 1. TCP's network congestion signals and responses

	Congestion Signals	Sender's Responses
1	Retransmission timer expired	Reduce the congestion window (*cwnd*) to one and perform a slow start (the sender is said to enter the *timeout state*).
2	Three duplicate ACKs received	Halve the *cwnd* and increase the *cwnd* by one per round-trip time (the sender is said to enter the *fast recovery state*).

$SRTT$ is the smoothed RTT and $VRTT$ is the RTT variation, which are updated according to Eq. (2) and Eq. (3) upon receiving a new RTT measurement rtt, respectively.

$$SRTT = 7/8 \times SRTT + 1/8 \times rtt. \quad (2)$$

$$VRTT = 3/4 \times VRTT + 1/4 \times |SRTT - rtt|. \quad (3)$$

2.1.1 Synchronous timeout-based attacks

The first type of timeout-based attacks is synchronous with the RTO value. That is, if the attacker knows the RTO value of the targeted TCP sender and is able to cause every retransmitted packet to drop, then the victim TCP sender's *cwnd* always stays at 1, and the throughput is equal to 0. This type of attack is shown in Fig. 1, which shows that the attack epoches coincide with the retransmission epoches.

Proposition 1 (Attack epoches for synchronous timeout-based attacks). *Let $t_n, n = 0, \ldots,$ be the nth attack epoch. The first attack starts at t_0, and for $n > 0$, the nth attack epoch in a synchronous timeout attack is given by*

$$t_n = \begin{cases} t_0 + (2^n - 1) \times RTO & \text{if} \quad 1 \le n \le N_{max} \\ t_0 + (2^{N_{max}} - 1) & \text{if} \quad N_{max} < n \le A_{max} \\ \times RTO + (n - N_{max}) & \\ \times RTO_{max}, & \end{cases} \quad (4)$$

where RTO_{max} is the maximum value of RTO, and A_{max} is the maximum number of attempts in retransmissions before the TCP sender gives up. Moreover, $N_{max} = \lfloor \log_2 \frac{RTO_{max}}{RTO} + 1 \rfloor$.

Proof. According to RFC 2988, a TCP sender will double the current RTO value when the retransmission timer expires [19]. If the new RTO is not more than RTO_{max}, then the sender will use the new RTO. Hence, $t_n = t_0 + \sum_{i=0}^{n-1} 2^i \times RTO = t_0 + (2^n - 1) \times RTO$ for $1 \le n \le N_{max}$, where $N_{max} = \max\{n : 2^{n-1} \times RTO \le RTO_{max}\}$. Otherwise, the RTO value is set to RTO_{max} and the sender continues to retransmit the lost packet until the total number of attempts reaches A_{max}. □

2.1.2 Asynchronous timeout-based attacks

The synchronous timeout-based attack is a perfectly timed attack that is obviously not quite feasible in practice. Therefore, the attack epoches in the second class of PDoS attacks are asynchronous with the retransmission epoches. A most straightforward realization of the asynchronous timeout-based attacks is to send pulses with *fixed periods*. Kuzmanovic and Knightly have recently showed that this kind of attack is indeed possible [13]. Their attack scheme works in the following way:

After a victim TCP sender enters the timeout state after the launch of the first attack pulse, the attacker lets the sender send data within a short period T_{Shift}. This period should be small but long enough for the sender to transmit some packets successfully. Therefore, T_{Shift} may be set to $2 \sim 3$ RTTs. Thus, the $SRTT$, $VRTT$, and RTO values are updated according to Eqs. (1-3) during T_{Shift}. Assume that the RTO value at the attack epoch is given by RTO_{min}, i.e., the value of second argument of the $\max$ function in Eq. (1) is smaller than RTO_{min}. If the new rtts do not deviate too much from the $SRTT$ value, it is very likely that the RTO value is still being given by RTO_{min} at the end of T_{Shift}. Thus, subsequent attack pulses can be launched with a fixed period of $RTO_{min} + T_{Shift}$, as depicted in Fig. 2.

2.2 AIMD-based attacks

In a AIMD-based attack, the attack pulses cause a victim TCP sender to frequently enter the fast recovery state. Recall that we consider a general AIMD algorithm denoted by $AIMD(a, b)$, $a > 0$, $1 > b > 0$. If an attack pulse is able to cause some packet losses in a TCP connection, but a sufficient number of duplicate ACKs can still be received by the sender, the *cwnd* will drop by $(1 - b)\%$. After that, the *cwnd* will increase by an MSS every RTT. Moreover, many TCP implementations do not send an ACK for every received packet. Instead, they send a delayed ACK after receiving d consecutive full-size packets, where d is typically equal to 2 [15]. In this case the sender's *cwnd* is only increased by $\frac{a}{d}$ per RTT. The case of $d = 1$ corresponds to the ACK-every-packet strategy. TCP Tahoe, TCP Reno, and TCP new Reno use

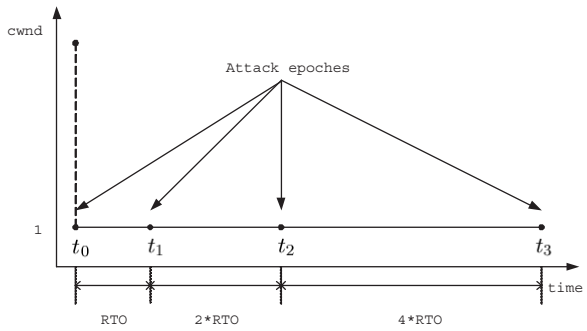


Figure 1. An example of the synchronous timeout-based PDoS attack.

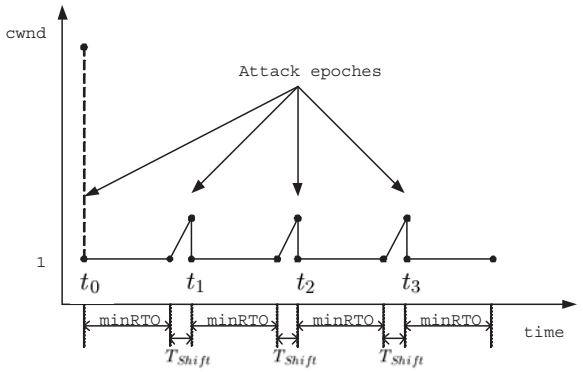


Figure 2. An example of a timeout-based PDoS attack with fixed periods.

$AIMD(1, 0.5)$.

Since it will take at least $\frac{(1-b)\times d}{a} \times W$ number of RTTs to restore the *cwnd* to W after a decrease from W to bW, the *cwnd* value could drop continuously if the attack pulses are launched frequently enough. Moreover, when the *cwnd* is dropped to a certain level, there may not be enough duplicate ACKs to trigger the fast recovery process. Thus, the AIMD-based attack can also achieve a similar effect as the timeout-based attack without causing timeouts at the beginning of the attack. On the other hand, the AIMD-based attack could also launch a degradation-of-service attack by lowering the attack frequency. Similar to the timeout-based attacks, there are two types of AIMD-based attacks.

2.2.1 Synchronous AIMD-based attacks

This type of AIMD-based attacks is referred to as synchronous in the sense that the attack epoches always coincide with a fixed set of *cwnd* values. For example, consider a general AIMD algorithm $AIMD(a, b)$,where $a > 0,\ 1 > b > 0$, and the sender's *cwnd* is increased by $\frac{a}{d}$ per RTT. Let $W_n, n = 0, 1, 2, \ldots,$ be the $cwnd$ value of a victim TCP connection *just before* the nth attack epoch. Therefore, W_0 is the *cwnd* value just before the attack. Suppose that an attack epoch always occurs at the instant when the *cwnd* rises from W_{n-1} to $f \times W_{n-1}, n \geq 1, 1 \geq f > b$ after a multiplicative decrease. The attack epoches for this type of synchronous AIMD-based attack are given in Proposition 2.

Proposition 2 (Attack epoches for synchronous AIMD-based attack). *For the synchronous AIMD-based attack just described, the number of attack pulses required to reduce the cwnd to 2 (the minimum value) is given by* $\frac{log(2/W_0)}{logf}$.

Assume that the RTT value is fixed. Let t_0 be the first attack epoch. For $n > 0$, the nth attack epoch is then given by

$$t_n = t_0 + \frac{1-f^n}{1-f} \times \frac{(f-b) \times d \times W_0}{a} \times RTT,\ \ n \geq 1. \tag{5}$$

Proof. Since the *cwnd* is decreased from W to $f \times W$ at each attack epoch, *cwnd* $= f^n \times W_0$ after the nth attack. Hence, the attack will bring down the *cwnd* to 2 by launching a sequence of $\frac{log(2/W_0)}{logf}$ attack pulses.

According to the attack strategy, before the arrival of the nth ($n \geq 1$) attack pulse the *cwnd* can only be increased to $(f-b) \times W_{n-1}$, which takes $\frac{(f-b)\times d\times W_{n-1}}{a} \times RTT$ amount of time, according to the AIMD algorithm. Therefore, the nth attack epoch should take place at

$$t_n = t_{n-1} + \frac{(f-b) \times d \times W_{n-1}}{a} \times RTT,\ \ n \geq 1. \tag{6}$$

Furthermore, by substituting $W_n = f \times W_{n-1}$ into Eq. (6),

$$t_n = t_{n-1} + \frac{(f-b) \times d \times f^{(n-1)} \times W_0}{a} \times RTT,\ \ n \geq 1. \tag{7}$$

We can therefore obtain Eq. (5) by a repeated substitution of t_{n-1}. □

Fig. 3 shows an example of a synchronous AIMD-based attack. The solid line depicts the trajectory of *cwnd* controlled by $AIMD(1, 0.5)$. The dashed line, on the other hand, depicts the trajectory of *cwnd* when the TCP sender is experiencing a synchronous AIMD-based attack.

2.2.2 Asynchronous AIMD-based attacks

Similar to the case of synchronous timeout-based attacks, it is difficult to launch a synchronous AIMD-based attack because of the difficulty in estimating the attack epoches.

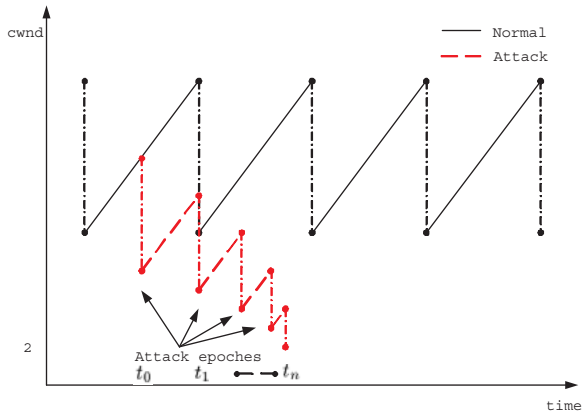


Figure 3. An example of a synchronous AIMD-based attack against $AIMD(1, 0.5)$.

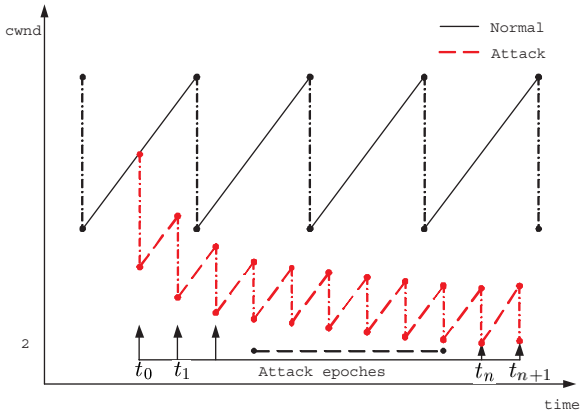


Figure 4. An example of a AIMD-based attack with fixed periods.

Therefore, we remove the synchronization requirement, and consider a AIMD-based attack with a fixed period of $T_{AIMD} = T_{Space} + T_{Extent}$. Proposition 3 presents the steady-state value of *cwnd* in the midst of such an attack. After that, Proposition 4 gives the minimum number of attack pulses for reducing the *cwnd* to the steady-state value.

Proposition 3 (Convergence of the *cwnd*). *Consider a AIMD-based attack with a fixed period of T_{AIMD} against a TCP connection using $AIMD(a, b)$. If the cwnd of the victim connection will converge during the attack, then the converged value is given by*

$$W_C = \frac{a}{(1-b) \times d} \times \frac{T_{AIMD}}{RTT}. \quad (8)$$

Proof. Just before the arrival of the $(n+1)$th attack pulse, the *cwnd* value is given by

$$\begin{aligned} W_{n+1} &= b \times W_n + \frac{a}{d} \times \frac{T_{AIMD}}{RTT} \\ &= b^{n+1} \times W_0 + \frac{a}{d} \times \frac{1-b^{n+1}}{1-b} \times \frac{T_{AIMD}}{RTT} \end{aligned} \quad (9)$$

If the *cwnd* converges, $W_{n+1} = W_n$ for some n. Therefore, by substituting $W_{n+1} = W_n$ into Eq. (9), we obtain the result. □

Proposition 4 (Minimum number of attack pulses). *Consider a AIMD-based attack with a fixed period of T_{AIMD} against $AIMD(a, b)$. Let $W_0 = W_C + \delta$, where $\delta > 0$. Moreover, if $W_n - W_C < \epsilon$, where ϵ is a small value, W_n is considered the same as W_C. Then, the minimum number of attack pulses required to reduce the cwnd from W_0 to W_C is given by*

$$N_{attack} < \frac{log\epsilon - log\delta}{logb}. \quad (10)$$

Proof. From Eq. (8) and Eq. (9), we have $W_n = b^n \times W_0 + (1 - b^n)W_C$. By substituting $W_0 = W_C + \delta$ into the equation and solving for n, we obtain $n = \frac{\log(W_n - W_C) - \log\delta}{\log b}$. Since W_n is considered to be the same as W_C if $W_n - W_C < \epsilon$, we obtain Eq. (10). □

In Fig. 5 we plot Eq. (10) for different values of b. The figure shows that the flow throughput of a typical TCP ($b = 1/2$) can be brought to the converged value using fewer than 10 attack pulses. With a higher value of b, more attack pulses will be required to achieve the same effect, because the *cwnd* drops with a slower rate in these cases. With a higher value of δ, it will also take a longer time for *cwnd* to converge.

2.2.3 A comparison of the two PDoS attacks

It is now useful to point out a major and important difference between the timeout-based attacks and AIMD-based attacks. Recall from section 2.1 that the timeout-based attack can effectively deny service to TCP flows whose $RTOs$ are less than the RTO_{min}. That is, an attacker can launch the attack epoches using Eq. (4) and $RTO = RTO_{min}$ in a synchronous attack and, as explained before, the attack period is deterministic in an asynchronous attack. On the other hand, other TCP flows may survive the timeout-based attack. The simulation results presented in [13] have indeed shown that those flows with an RTT higher than $180ms$ are less affected under the proposed low-rate attack. However, this is not the case

with AIMD-based attacks. In a periodic AIMD-based attack, Proposition 3 indicates that all flows, regardless of their RTTs, will be adversely affected by the AIMD-based attack by limiting their *cwnd* to a low value of W_C.

To drive the point further, in Fig. 6 we use Eq. (8) to show the relationship between W_C and $\frac{T_{AIMD}}{RTT}$ for a TCP flow and a TCP-friendly flow ($AIMD(0.31, 0.875)$) with $d = 2$. According to [24], a flow with $AIMD(a, b)$ is considered to be *TCP-friendly* if its parameters satisfy $a = \frac{4\times(1-b^2)}{3}$. Therefore, the converged *cwnd* value for a TCP-friendly flow is given by $W_C = \frac{4\times(1+b)}{3\times d} \times \frac{T_{AIMD}}{RTT}$. The figure also shows a lower bound and an upper bound on W_C.

The figure shows that if $\frac{T_{AIMD}}{RTT}$ is small, the flow's *cwnd* will be constrained to a very low value that will severely limit the flow's throughput. For example, consider those flows with RTT between 200ms and 500ms. In [13], the simulation results have shown that these flows will survive a periodic timeout-based attack. However, Fig. 6 shows that a periodic AIMD-based attack with a period of $1s$ is sufficient to degrade their throughput to the extent that the *cwnd* will be confined within $(4/3, 20/3)$. Note that the TCP fast recovery algorithm usually requires three duplicate ACKs. Therefore, even if the *cwnd* value is given by the upper bound, it is very likely that the fast recovery procedure cannot be started and that a timeout will therefore occur.

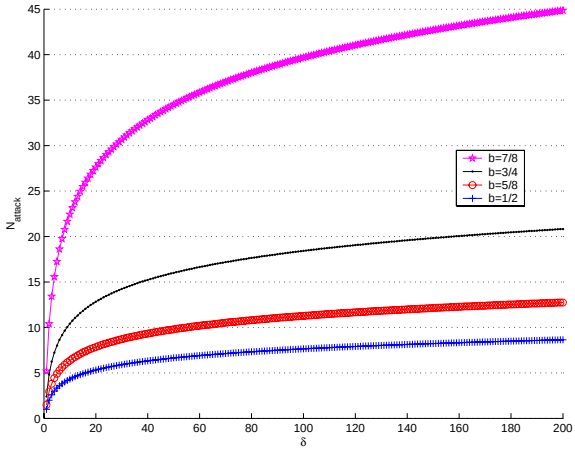


Figure 5. The relationship between N_{attack} and δ.

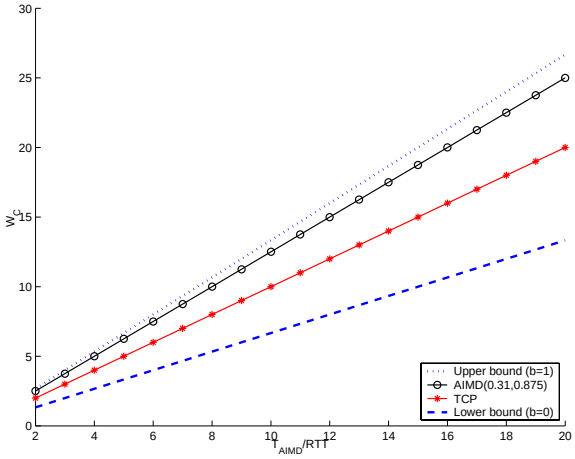


Figure 6. The relationship between W_C and $\frac{T_{AIMD}}{RTT}$.

3 A two-stage detection scheme for PDoS attacks

We propose in this section a novel two-stage scheme for detecting PDoS attacks. In so doing, we assume that both the timeout-based and AIMD-based attacks can be launched. Since a successful PDoS attack does not require a sustained high attack packet rate, the feasible location for detecting such an attack is at the victim network. The patterns of both incoming traffic and outgoing traffic are then under surveillance. Moreover, since the PDoS attack can be effectively launched even by a single source, our detection system is based on the detection of traffic pattern anomalies.

We have discovered a total of two anomalies that were incurred by a PDoS attack. The first is that the incoming data traffic will fluctuate in a more extreme manner during an attack. The abnormal fluctuation is a combined result of two different kinds of anomalous events caused by the attack. The first kind is obviously the introduction of the attack pulses, and the other is a fast decline in the traffic volume of the affected TCP flows. For AIMD-based attacks, the unusually high level of traffic fluctuations can immediately be observed at the beginning of the attack. The fluctuation may even continue after the *cwnd* converges to W_C, because it is very likely that the flows involved will time out because of a small value of W_C.

The second anomaly has to do with the outgoing TCP ACK traffic. As just mentioned, the incoming legitimate TCP traffic volume will decline because of the attack. However, the overall incoming TCP traffic volume may or may not decline during the attack, because the attack packets can also be TCP based. Hence, our detection system is also required to observe a possible decline in the outgoing TCP ACK traffic. It is important to emphasize that detecting both anomalies is necessary for confirming a PDoS. Doing one without the other will result in a high false positive rate.

3.1 The first stage: A wavelet analysis of the network traffic

Based on the above discussion, the first stage in the detection process is to monitor the variability in the incoming traffic and in the outgoing TCP ACK traffic. Here we employ a discrete wavelet transform (DWT) for this purpose. The DWT represents a signal $f(t) \in L^2(R)$ using scaling functions $\varphi_{j,k}(t)$, and a translated and dilated version of wavelet functions $\psi_{j,k}(t)$ [23]. Since the wavelet functions operate like high-pass filters that use narrow time windows to compute differences in signals, they can capture the variability of the incoming traffic volumes. On the other hand, the scaling functions perform like low-pass filters; therefore, they can be used to extract the trend of the outgoing TCP ACK traffic.

To realize an on-line detection, we use a moving window to group G continuous samples to compute the DWT. Moreover, we define a statistic based on the signal energy to quantify the variability in the incoming traffic for the nth window of samples as follows.

$$E_H(n) = \frac{1}{G} \sum_k |d_{1,k}^{In}|^2, \tag{11}$$

where $d_{1,k}^{In}$ is the wavelet coefficient at the finest scale ($j = 1$). Similarly, we define a statistic based on the signal energy to characterize the trend in the outgoing TCP ACK traffic for the nth window of samples as follows.

$$E_L(n) = \frac{1}{G} \sum_k |c_{L,k}^{Out}|^2, \tag{12}$$

where $c_{L,k}^{Out}$ is the scaling coefficient at the highest decomposed scale ($j = L$). Further details about the DWT can be found in Appendix A.

3.2 The second stage: A CUSUM detection for change points

The second stage is then to detect abrupt changes in $E_H(n)$ for the incoming traffic and in $E_L(n)$ for the ACK traffic. We employ a nonparametric CUSUM algorithm for this purpose. The CUSUM method assumes that the mean value of the variable under surveillance will change from negative to positive when a change occurs. Since both $E_H(n)$ and $E_L(n)$ are larger than zero, we transform them into two random sequences, $Z_H(n)$ and $Z_L(n)$, which have negative mean values under normal conditions.

$$Z_H(n) = E_H(n) - \beta_H \tag{13}$$

$$Z_L(n) = \beta_L - E_L(n), \tag{14}$$

where β_H and β_L are constants for determining the mean values of $Z_H(n)$ and $Z_L(n)$. Normally, we can set β_H to the upper bound of $E_H(n)$, and set β_L to $\overline{E_L(n)} - P_{tolerance} \times [\triangle(E_L(n))]$, where $\triangle(E_L(n))$ is the standard deviation of $E_L(n)$, and $P_{tolerance}$ controls the limit of the allowable decrease in $E_L(n)$.

Let T^{In} be the detection time for $Z_H(n)$ when

$$y_{Z_H}(n) > C_{cusum}^{In}, \tag{15}$$

where $y_{Z_H}(n)$ is the CUSUM value of $Z_H(n)$ and C_{cusum}^{In} is the corresponding threshold. Similarly, let T^{Out} be the detection time for $Z_L(n)$ when

$$y_{Z_L}(n) > C_{cusum}^{Out}, \tag{16}$$

where $y_{Z_L}(n)$ is the CUSUM value of $Z_L(n)$ and C_{cusum}^{Out} is the corresponding threshold. The detection system confirms the onset of a PDoS attack when both Eq. (15) and Eq. (16) hold. Hence, the final detection time is determined by

$$T^{Final} = max\{T^{In}, T^{Out}\}. \tag{17}$$

Let T_{Attack} be the start time of a PDoS attack, and τ^{In}, τ^{Out}, and τ^{Final} be the detection delays as defined below.

$$\tau^{In} = T^{In} - T_{Attack} \tag{18}$$

$$\tau^{Out} = T^{Out} - T_{Attack} \tag{19}$$

$$\tau^{Final} = max\{\tau^{In}, \tau^{Out}\} \tag{20}$$

Other details about the CUSUM algorithm can be found in Appendix B.

4 Performance evaluation

We have conducted experiments using both `ns-2` simulation [1] and a test-bed to evaluate the effectiveness of the PDoS attacks and the two-stage detection system.

4.1 Simulation experiments and results

We have conducted extensive `ns-2` simulation experiments based on the simulation scripts provided by [13]. The network topology is shown in Fig. 7(a). It consists of N pairs of TCP senders and TCP receivers. The links connecting the router S, the senders, and an attacker are 100 Mbps, as are the links between router R and the receivers. Both routers are connected through a bottleneck link of 10 Mbps with RED queue management [10]. The simulation period is 900 seconds, and the attack begins at 181 seconds ($T_{Attack} = 181$) and ends at 720 seconds. Since $S_{Extent}(n)$ is set to a constant in all experiments, we use $\mathbb{A}(T_{Extent}(n), R_{Attack}, T_{Space}(n), N)$ to describe the PDoS attack, where R_{Attack} is the sending rate in each attack pulse.

The settings for the detection system are selected as follows. The detection system aggregates the incoming data traffic and outgoing TCP ACK at a fixed time interval $T_s = 0.25s$. In order to achieve a small detection delay, the moving window size is chosen to be $G = 120$, which means that each observation period is half a minute ($0.25s \times 120 = 30s$). At the end of each observation period, the detector generates the statistics, such as $Z_H(n)$ and $Z_L(n)$, and then executes the CUSUM algorithm to search for abrupt changes.

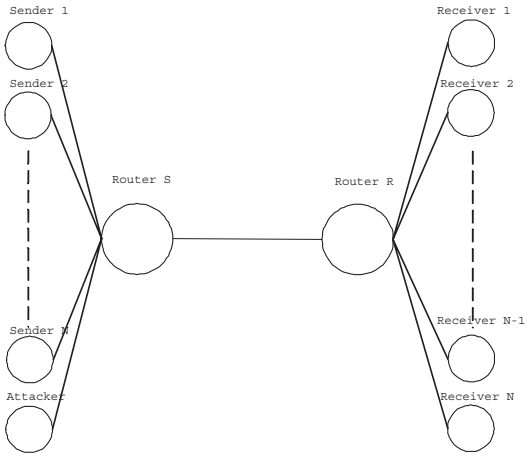
We employ the Daubechies family of wavelets in the first stage of the detection, which have been widely used to analyze network traffic. The Daubechies wavelets, represented by $DB(N)$, are orthonormal and compactly supported with N vanishing moments and $2N - 1$ support length. $DB(1)$, the Haar wavelet, is used to analyze the incoming data traffic and $DB(4)$ is applied to analyze the outgoing TCP ACK traffic.

We calculate C_{cusum} in Appendix B by selecting $\tau = m + 1$ and $h = 2\|a\|$ in Eq. (30). In principle, β_H can be set to the upper threshold of $E_H(n)$, which can be estimated by the value that corresponds to 95% of the cumulative distribution function of $E_H(n)$. To simplify the process, we set the value of β_H to the maximum value of $E_H(n)$, and set the tolerance parameter $P_{tolerance}$ to 1 in the simulation experiments. As a result of the parameter selection, the detection system will be more sensitive to changes in the ACK traffic, but will tolerate the normal oscillations of incoming traffic in order to keep a low false positive rate.

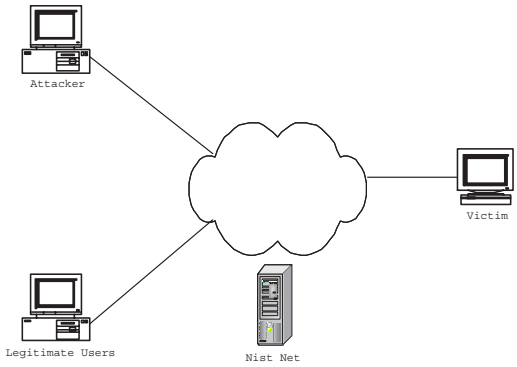
4.1.1 Timeout-based attacks

When we discuss the two types of PDoS attacks, we assume that a PDoS attack is either a timeout-based attack or a AIMD-based attack. However, a PDoS attack will generally cause an affected TCP to frequently enter *both* the timeout and fast recovery states. Therefore, in the rest of this paper, a PDoS attack is considered as a timeout-based attack if it will cause "many more" returns to the timeout state than to the fast recovery state. Similarly, a PDoS attack is regarded as a AIMD-based attack if it will cause many more returns to the fast recovery state than to the timeout state. In this section we first consider the timeout-based attack, and then examine the same set of issues for a AIMD-based attack.

We have conducted simulation experiments for three types of flows: TCP Reno, TCP New Reno, and TCP-friendly flows that are based on $AIMD(0.31, 0.875)$ and TCP SACK [22]. The simulation results for the incoming data traffic are shown in Figs. 8(a)-8(b), and the results for the outgoing ACK traffic in Figs. 9(a)-9(b). In each experiment, there are 30 flows with heterogeneous $RTTs$



(a) The topology of the simulation model.



(b) The topology of the test-bed.

Figure 7. Network topologies for simulation and test-bed experiments.

ranging from 20ms to 460 ms. The PDoS attack is parameterized as $\mathbb{A}(150ms,\ 10Mbps,\ 1050ms,\ \lceil\frac{540}{1.05+0.15}\rceil = 450)$. In order to compare the results, we adjust all of the horizontal axes with the same scale. However, this does not mean that each figure contains the same number of points. For example, Fig. 8(a) consists of 3600 points, each of which is a count during an interval of 0.25s, whereas Fig. 8(b) contains only 30 points that represent the values of $y_{Z_H(n)}$ computed at the end of each observation period.

As shown in Fig. 8(a), the wavelet coefficients clearly indicate that the incoming traffic exhibits more oscillations during the PDoS attack between 181s to 720s, as compared with periods without the attack, i.e., $[1s, 180s] \cup [721s, 900s]$. In the other direction, Fig. 9(a) shows that the outgoing TCP ACK traffic declines after the onset of the PDoS attack.

Fig. 8(b) displays the statistic $Z_H(n)$ of the incoming data traffic and the CUSUM results: $y_{Z_H}(n)$ and C^{In}_{cusum}. The detection times for $Z_H(n)$ for all three types of flows are all equal to $T^{In} = 210s$, which is $30s$ after the beginning of the attack. For the outgoing ACK traffic, Fig. 9(b) gives the result of $Z_L(n)$, $y_{Z_L}(n)$, and C^{In}_{cusum}. As with the data traffic, the detection times for $Z_L(n)$ for all three types of flows are all equal to $T^{Out} = 210s$. As a result, the final detection times for all three cases are equal to $T^{Final} = max\{T^{In}, T^{Out}\} = 210s$. The detection delay is $\tau^{Final} = max\{\tau^{In}, \tau^{Out}\} = 30s$, which means that the total detection delay is equal to the length of an observation period.

Table 2 summarizes other experimental results that are obtained by changing the value of S_{Extent}. The first column ($Mbps$) shows the different attack rates, ranging from 1 Mbps to 20 Mbps. τ^{Final}_{Reno}, $\tau^{Final}_{NewReno}$, and τ^{Final}_{AIMD} show the final detection delay (in seconds) for TCP Reno, New-Reno, and $AIMD(0.31, 0.875)$, respectively. On the other hand, $Loss_R$, $Loss_N$, and $Loss_A$ give the estimates on the percentage decrease in the legitimate TCP traffic. These estimates are computed by dividing the average number of ACKs observed during a PDoS attack by that during an attack-free environment. The results show that $AIMD(0.31, 0.875)$ is more robust than the other two versions of TCP under the PDoS attack and that the TCP Reno is most vulnerable, primarily because the *cwnd* is decreased only to 0.875 of its previous value (instead of 0.5).

Moreover, when the attack burst rate is at least the same as the bandwidth of the bottleneck, i.e., 10, 15, 20 Mbps, the $Loss_{R,N,A}$ columns show that the total throughput can be reduced by as least 50%. The results also show that our detection scheme will discover the attack after one observation period (30 seconds). When the attack burst rate is only half of the bandwidth of the bottleneck, i.e. 5 Mbps, the attack can deplete approximately 40% of total throughout. In this low-rate attack, our scheme can still detect the attack only after one observation period.

Table 2 shows other lower-rate attacks, i.e., 1, 2.5, and 4 Mbps. We use the symbol "*" to show that the corresponding attacks can sometimes be detected but not all the time, and the symbol "X" to show that the corresponding attacks cannot be detected in all experiment runs. The detection outcomes are largely dependent on the choice of detection parameters. For example, a relatively large β_H will fail to detect extremely low-rate attacks, because it cannot differentiate between the traffic fluctuations caused by the attack and the normal traffic fluctuations. Moreover, a relatively small $P_{tolerance}$ will increase the false positive rate, because it would be too sensitive to the changes in the ACK traffic and will therefore raise many false alarms.

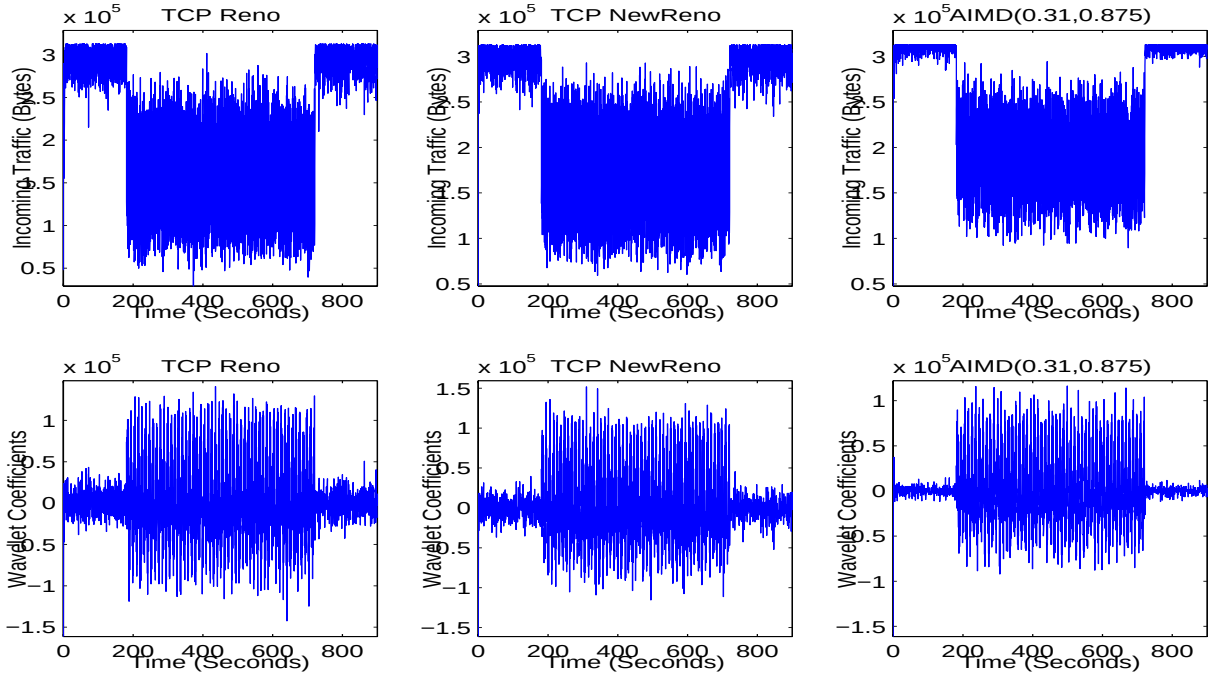
4.1.2 AIMD-based attacks

In this section, we re-examine the issues discussed in the last section for attacks that are dominated by the AIMD-based attack. Therefore, we have employed the same topology and parameter settings as before. But for the purpose of comparison, we consider only the New-Reno flows. The results are presented in Figs. 10-11. There are two different PDoS attacks in each figure. The graphs on the left are for $\mathbb{A}(10ms,\ 100Mbps,\ 800ms\ ,\ 667)$, while those on the right are for $\mathbb{A}(10ms,\ 50Mbps,\ 400ms,\ 1317)$.

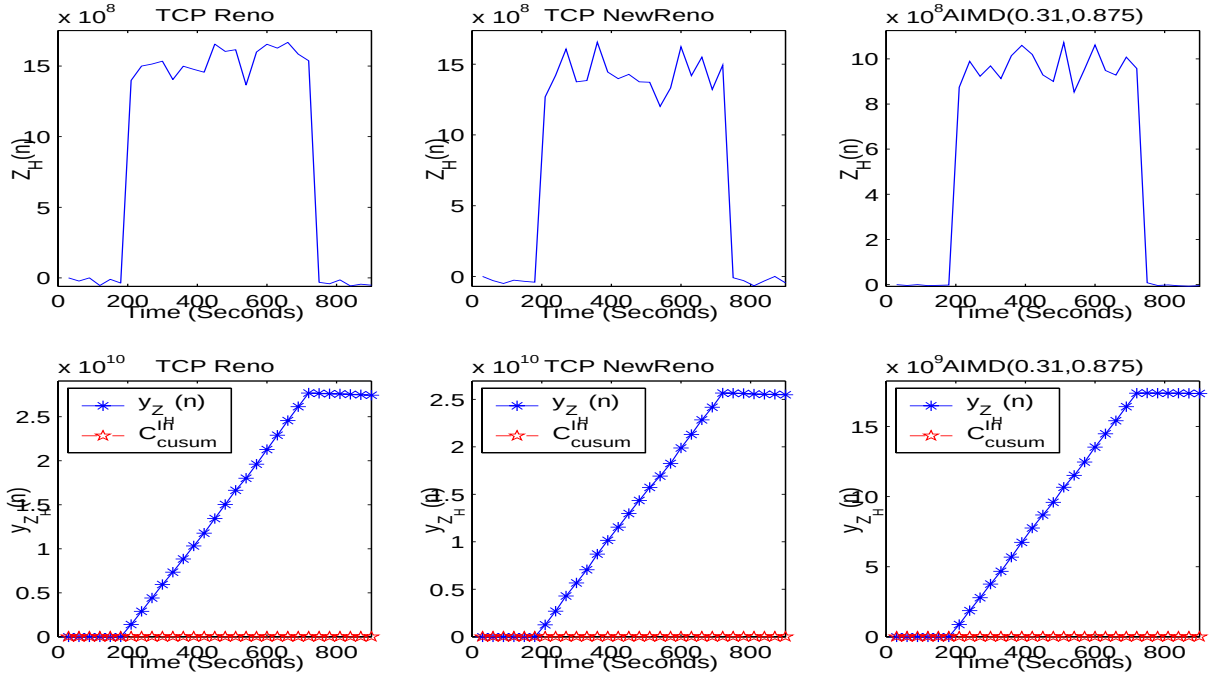
For the attack $\mathbb{A}(10ms,\ 100Mbps,\ 800ms,\ 667)$, Fig. 10 shows that the total throughput gradually decreases after an abrupt change at the beginning of the attack. The first high-rate pulse causes many flows to decrease their *cwnd*s. After that, the AIMD-based attack will force the senders' *cwnd* to gradually decrease to W_C. It may not be easy to observe in Fig. 10(a) the differences in the wavelet coefficients before and after the attack. However, our detection scheme can still quickly discover it by combining the detection results in both Fig. 10(b) and Fig. 11(b).

As for the attack $\mathbb{A}(10ms,\ 50Mbps,\ 400ms,\ 1317)$, Fig. 10(a) shows that this attack also forces most TCP flows to converge to W_C during the period between $600s$ and $720s$, i.e., the relationship between the AIMD-based attack and TCP flows is stable. Fig. 10(b) captures the phenomena in which the CUSUM value $Y_Z(n)$ drops from its maximum value. Since the throughput is very low during that period, the fluctuation also become weak. Otherwise, the severe fluctuation will continue because of the increase in the probability of timeout. In either case, our detection mechanism can detect it quickly.

Table 3 compares the three PDoS attacks (one from section 4.1.1). All three attacks have similar aver-

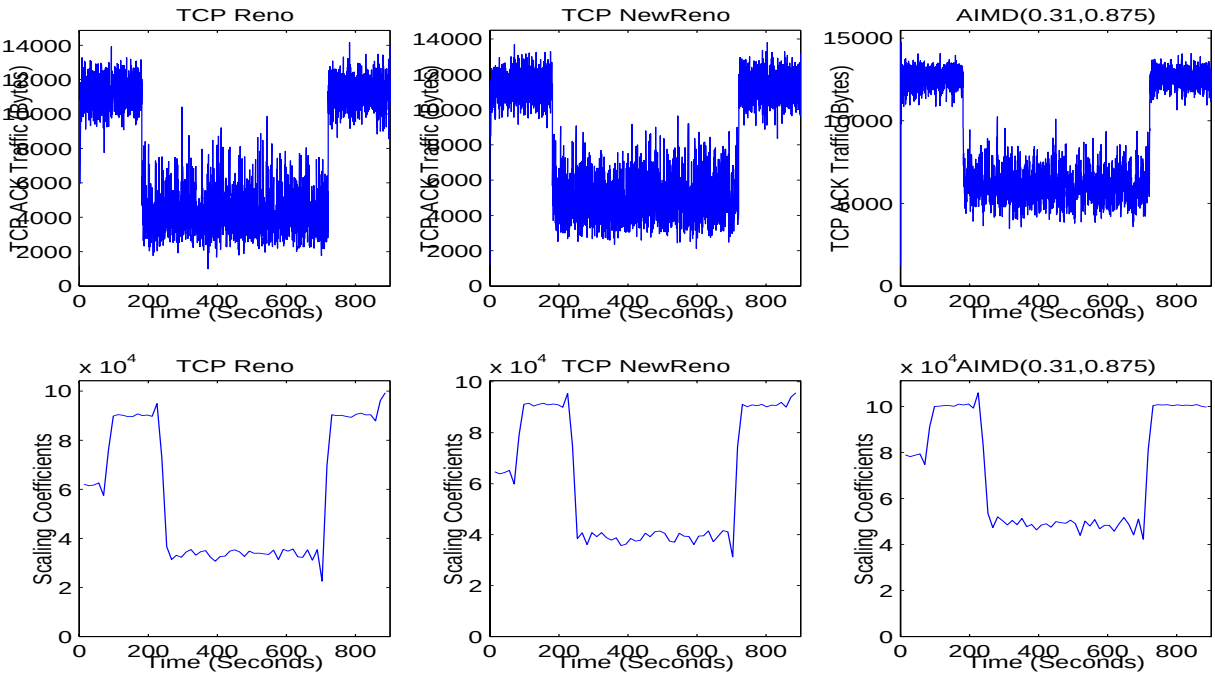


(a) The incoming traffic and their wavelet coefficients.

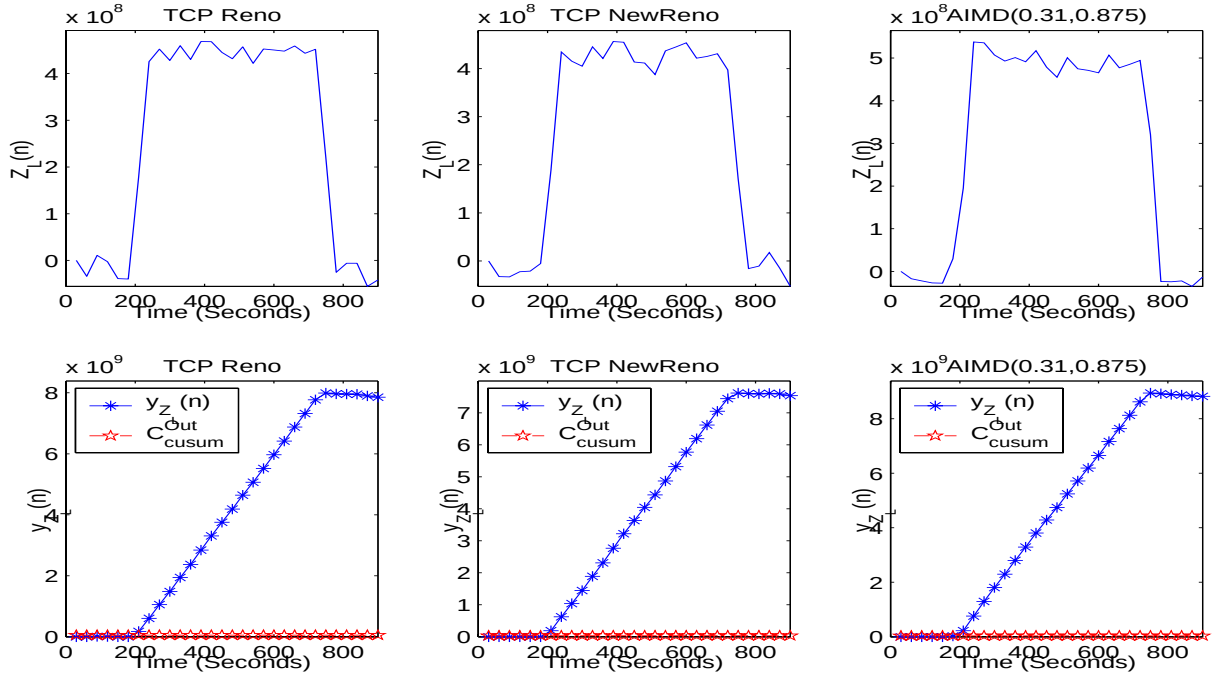


(b) $Z_H(n)$ and the CUSUM results.

Figure 8. The incoming traffic and the detection results for the timeout-based attacks.

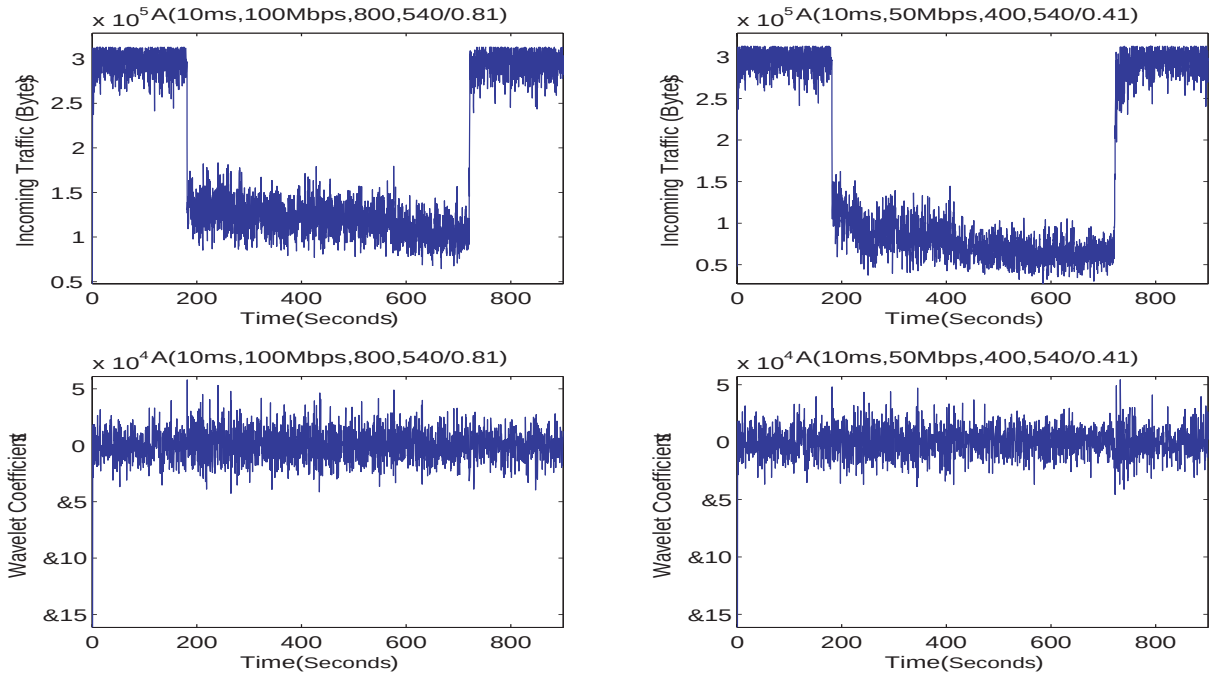


(a) The outgoing ACK traffic and their scaling coefficients.

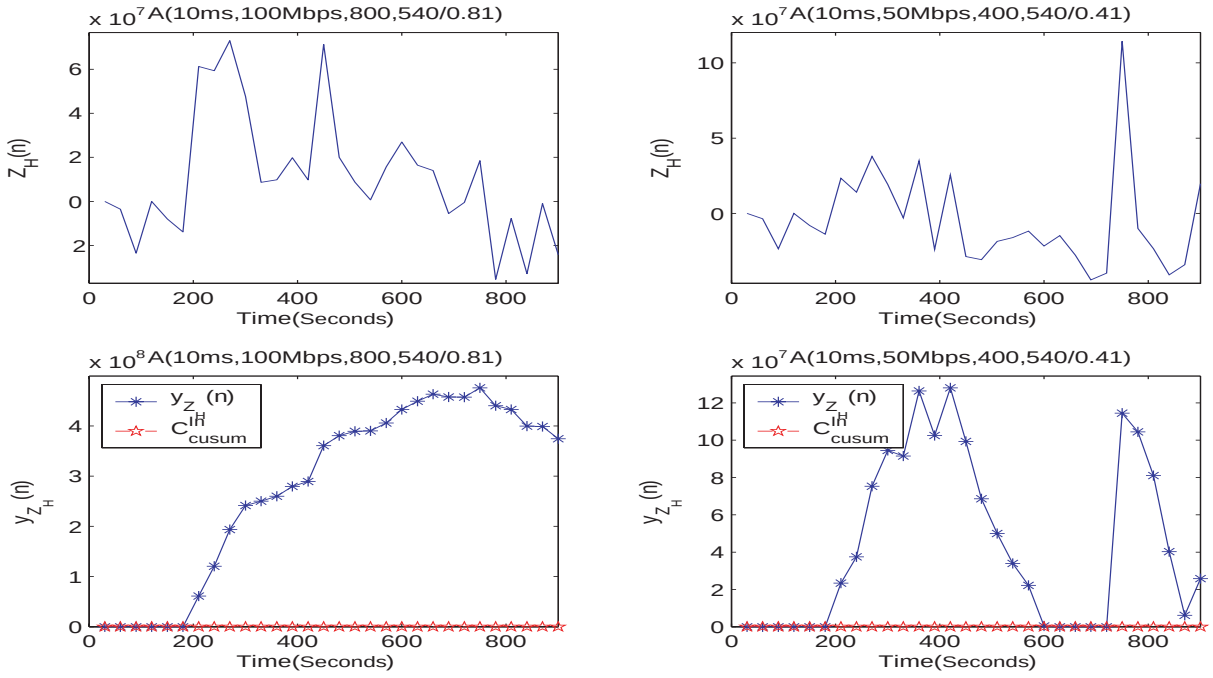


(b) $Z_L(n)$ and the CUSUM results.

Figure 9. The outgoing ACK traffic and the detection results for the timeout-based attacks.

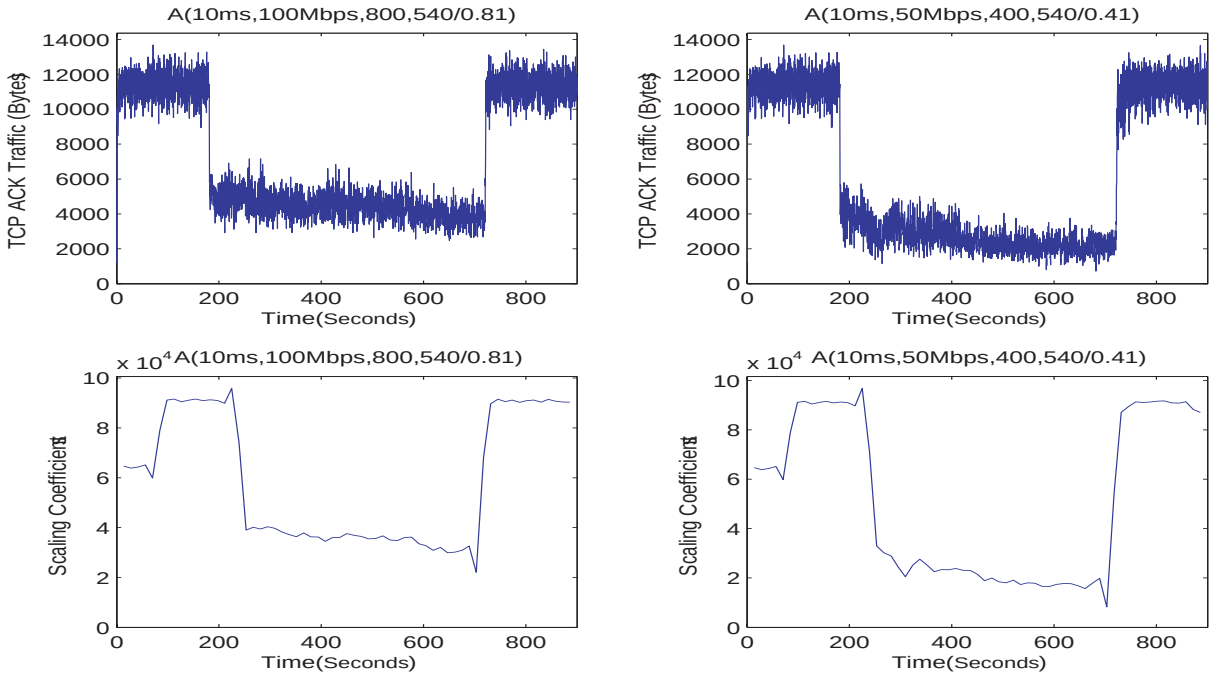


(a) The incoming traffic and their wavelet coefficients.

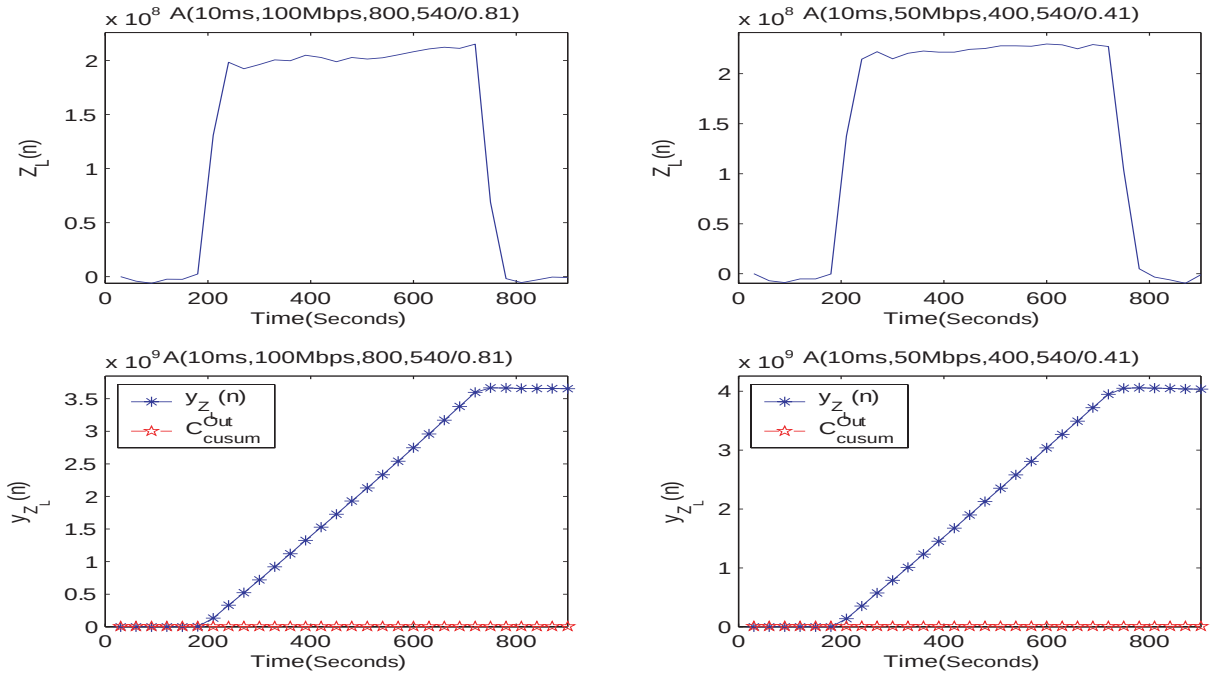


(b) $Z_H(n)$ and the CUSUM results.

Figure 10. The incoming traffic and the detection results for the AIMD-based attacks.



(a) The outgoing ACK traffic and their scaling coefficients.



(b) $Z_L(n)$ and the CUSUM results.

Figure 11. The outgoing ACK traffic and the detection results for the AIMD-based attacks.

Table 2. Detection time and loss rate caused by the timeout-based attacks.

R_{Attack}	τ_{Reno}^{Final}	$Loss_R$	$\tau_{NewReno}^{Final}$	$Loss_N$	τ_{AIMD}^{Final}	$Loss_A$
20	30	0.8039	30	0.7781	30	0.7757
15	30	0.7341	30	0.6962	30	0.6594
10	30	0.6234	30	0.5679	30	0.5098
5	30	0.4538	30	0.4024	30	0.3221
4	30	0.3969	30	0.353	60*	0.281
2.5	30*	0.2933	30*	0.2547	X	0.1877
1	X	0.1076	X	0.0805	X	0.0463

age attack rates, which is computed by $R_{Average} = \frac{T_{Extent} * R_{Attack}}{T_{Extent}+T_{Space}}$, but they differ in their values of R_{Attack}, T_{Extent}, and T_{Space}. It is interesting to note that, although it has the lowest attack rate, the attack $\mathbb{A}(10ms, 50Mbps, 400ms, 1317)$ causes the worst damage to the TCP throughput. The results suggest that the AIMD-based attack can achieve the same effect as the timeout-based attack, but with a lower attack rate. Moreover, our detection scheme can detect all of these pulsing attacks in a timely manner after one observation period.

4.2 Test-bed experiments and results

The test-bed topology is shown in Fig. 7(b). We use `NIST Net` [7] to simulate the network and `iperf` to measure the TCP throughput. The link between `NIST Net` and the victim is 10 Mbps, whereas the links connecting the legitimate users and the attacker to the `NIST Net` are 100 Mbps. We also set the RTTs of both the legitimate users and the attacker to 100ms. In this setting, the legitimate user is running Linux 2.4.20-8 (Red Hat 9 (Shrike)), whose RTO_{min} is 200 ms, instead of the 1 second suggested in [19]. There are two reasons for using this setup. First, it is important to see what effect the PDoS attack will have on the TCP/IP stacks that may not follow the standard RFCs. Even though an attacker may not be able to determine the actual value of RTO_{min}, it is still possible for the attack to seriously degrade the victim's throughout. The second is to test the detection scheme under a "nonoptimal" PDoS attack. For this purpose, we launch a PDoS attack at 181 seconds with $R_{Attack} = 10Mbps$, $T_{Extent} = 200ms$, and $T_{Space} = 1000ms$.

As shown in Figs. 12-13, our scheme can detect the ongoing attack in a timely manner. Fig. 12(a) shows that the PDoS attack not only successfully degrades the TCP throughput, but also causes severe fluctuations in the incoming traffic as shown in Fig. 12(b). The detection time for $Z_H(n)$ is given by $T^{In} = 210s$; therefore, the delay time is $\tau^{In} = 30s$. The outgoing TCP ACK traffic begins to decrease after the attack is launched, as shown in Fig. 13(a). The detection time for $Z_L(n)$ is given by $T^{Out} = 240s$. Thus, the final detection time is given by $max\{T^{In}, T^{Out}\} = 240s$, and the detection delay is only $max\{\tau^{In}, \tau^{Out}\} = 60s$. Thus, the results obtained from the test-bed are quite consistent with those from the simulation experiments presented in the last section.

The `NIST NET` implements the Derivative Random Drop (DRD) algorithm instead of the Random Early Detection (RED) algorithm. Although both of them are superior to the drop tail method in terms of effectively controlling the average queue size, DRD is more sensitive to the traffic burst than RED[7]. Hence, we have conducted experiments on RED-based routers by replacing `NIST NET` with `dummynet`[21]. We set the parameters of RED with the following values: $min_{th} = 80$, $max_{th} = 160$, $w_q = 0.002$, $max_p = 0.1$, and the queue size is 250 packets. The `iperf` generates 15 TCP flows with $RTT = 150ms$. We have conducted a total of 15 PDoS attacks with different parameters as shown in Table 4.

The results from Table 4 have clearly shown that the PDoS attack can still seriously degrade TCP throughput even when the RTO_{min} is not equal to 1 second, as suggested by [19]. For Linux (2.4.20-8), which sets the RTO_{min} to 200ms and TCP flows with an RTT of 150ms, a periodic timeout-based attack may not be effective because it is very difficult to estimate the RTO. However, the AIMD-based attack can still reduce the throughput with a reasonable attack rate, e.g., the 5th PDoS in Table 4 can cause a 41.5% loss in throughput with a relatively small average attack rate (1.25 Mbps).

Given the same values of R_{Attack} and T_{Extent}, a PDoS attack with a smaller T_{Space} causes a bigger loss, e.g. 1st, 4th, 7th, 10th, and 13th attacks in Table 4. Another interesting observation is that those PDoS attacks, which have the same value of $R_{Attack} \times T_{Extent}(n)$, may have dif-

Table 3. Detection time and loss rate for three different PDoS attacks.

	$\mathbb{A}(150ms, 10Mbps, 1050ms, 450)$	$\mathbb{A}(10ms, 100Mbps, 800ms, 667)$	$\mathbb{A}(10ms, 50Mbps, 400ms, 1317)$
Average Rate	1.25Mbps	1.235Mbps	1.22Mbps
Loss Rate	0.5679	0.6073	0.7639
τ^{In}	30	30	30
τ^{Out}	30	30	30
τ^{Final}	30	30	30

Table 4. Different pulsing attacks and the detection time

Seq	$T_{Extent}(ms)$	R_{Attack} (Mbps)	T_{Space} (ms)	$R_{Average}$ (Mbps)	Loss	τ^{Final} (s)
1	100	10	900	1	0.239	60
2	200	10	1800	1	0.328	60
3	100	15	1400	1	0.258	60
4	100	10	700	1.25	0.272	60
5	200	10	1400	1.25	0.415	30
6	100	15	1100	1.25	0.371	30
7	100	10	566	1.5	0.293	60
8	200	10	1133	1.5	0.484	30
9	100	15	900	1.5	0.374	30
10	100	10	471	1.75	0.308	30
11	200	10	943	1.75	0.521	30
12	100	15	757	1.75	0.449	30
13	100	10	400	2	0.318	30
14	200	10	800	2	0.535	30
15	100	15	650	2	0.476	30

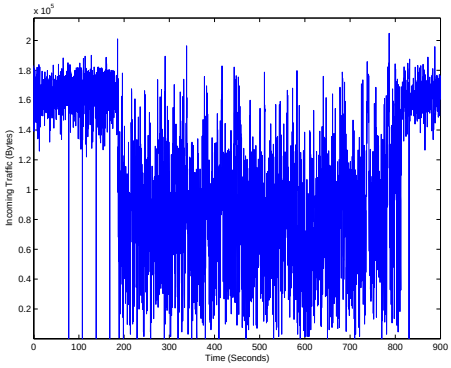
$R_{bottleneck}$ (bandwidth of the bottleneck) : 10 Mbps.

ferent effects on the TCP flows. For example, from our experiments, the PDoS attack with $T_{Extent} = 200ms$ and $R_{Attack} = 10Mbps$ always has a more severe effect than the other two types of PDoS attacks when all of them have the same average attack rate. This may be due to its large T_{Extent} value, which can clog the bottleneck link for such a long period that the packet-marking probability in RED [10] increases and the packets belonging to many flows are dropped. The experiment results also show that the proposed two-stage detection scheme can effectively discover the pulsing DoS attack. In most cases, the detection delay is only one observation period, i.e., 30s.

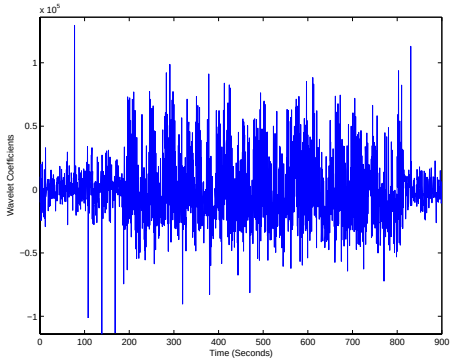
5 Conclusions

In this paper, we have identified a new class of PDoS attacks. Unlike the traditional DoS attack, the PDoS attack can effectively achieve the same purpose with a much lower attack rate. We have presented two specific attack methods: timeout-based and AIMD-based, and their variants. Our analysis has confirmed that the attacks can be very effective by forcing the affected TCP senders to continuously re-enter the fast recovery state or the timeout state.

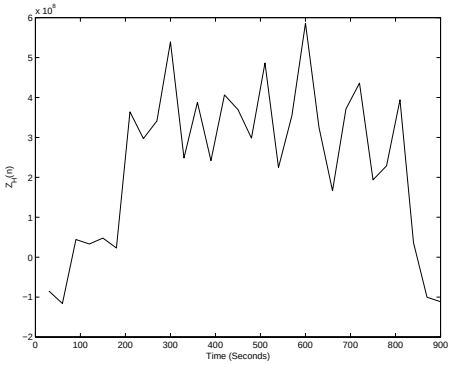
Another important contribution of this paper is a novel, two-stage detection scheme for the PDoS attacks. The detection is based on an unusually high variability in the incoming data traffic and a drastic decline in the outgoing ACK traffic observed in the midst of a PDoS attack.



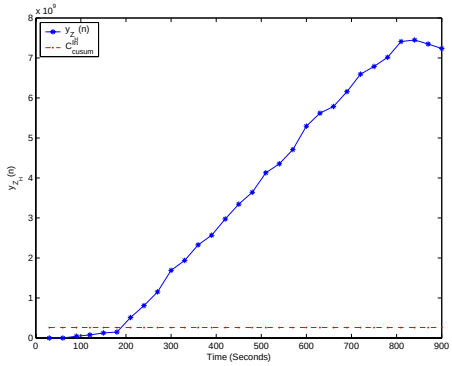
(a) Incoming traffic for the Test-bed.



(b) Wavelet coefficients of the incoming traffic.



(c) $Z_h(n)$ of the incoming traffic.



(d) The CUSUM results

Figure 12. Detection based on incoming traffic.

As a result, we have employed wavelet transform to observe the incoming data traffic and outgoing ACK traffic, and a nonparametric CUSUM algorithm to detect change points. The results from both the simulation and test-bed experiments show that the proposed scheme is effective at detecting a low-rate PDoS attack. Moreover, our scheme is feasible for on-line detection because of the low time complexity for both the computation of the discrete wavelet transform and the CUSUM method.

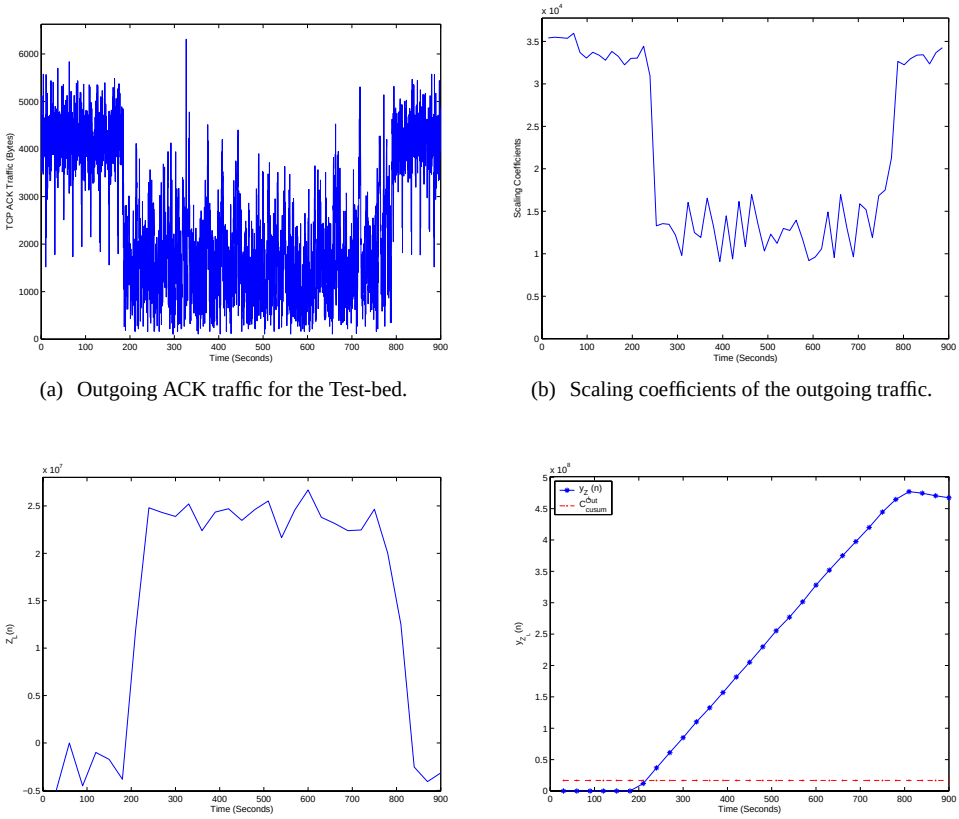
Acknowledgment

The work described in this paper was partially supported by a grant from the Research Grant Council of the Hong Kong Special Administrative Region, China (Project No. PolyU 5080/02E). We also thank the anonymous reviewers for their helpful comments.

A Discrete wavelet transform

Wavelet transform is very suitable for analyzing irregular signals, such as network traffic, because it gives a more accurate local description of signal characteristics in both time and frequency domains. Indeed, wavelet transform has been applied to analyze network traffic and identify traffic anomalies. For example, wavelet analysis has been employed to identify traffic anomalies caused by flooding-based DoS and flash crowds through a deviation score [16]. Compared with the work in [16], there are two main differences in our wavelet analysis. First, the wavelet analysis there is used to perform a postmortem analysis of trace data, whereas ours concentrates on a real-time analysis of incoming data. Second, the analysis there only considers the signal variations in the high and medium-frequency bands that are not sufficient to detect the PDoS attack. Our analysis requires both high and low-frequency bands.

The discrete wavelet transform (DWT) represents a signal $f(t) \in L^2(R)$ using scaling functions $\varphi_{j,k}(t)$, and a translated and dilated version of wavelet functions



(a) Outgoing ACK traffic for the Test-bed.

(b) Scaling coefficients of the outgoing traffic.

(c) $Z_L(n)$ of the outgoing ACK traffic.

(d) The CUSUM results.

Figure 13. Detection based on outgoing ACK traffic.

$\psi_{j,k}(t)$:

$$f(t) = \sum_k c_{j_0}(k)\varphi_{j_0,k}(t) + \sum_k \sum_{j=j_0} d_j(k)\psi_{j,k}(t), \quad (21)$$

where $\{\varphi_{j,k}(t) = 2^{-j/2}\varphi(2^{-j}t - k),\ j,k \in Z\}$ and $\{\psi_{j,k}(t) = 2^{-j/2}\psi(2^{-j}t - k),\ j,k \in Z\}$. In this expansion, the first summation describes a coarse approximation of $f(t)$, and the second summation depicts the details of $f(t)$. In practice, the coefficients $c_j(k)$ and $d_j(k)$ are calculated via the Mallat's pyramid algorithm:

$$c_j(k) = \sum_m h_0(m - 2k)c_{j-1}(m), \quad (22)$$

$$d_j(k) = \sum_m h_1(m - 2k)c_{j-1}(m), \quad (23)$$

where h_0 and h_1 are the coefficients of low-pass and high-pass filters, respectively. If the scaling functions and wavelet functions form an orthonormal basis, Parseval's theorem states that $f(t)$'s energy is equal to the energy in its scaling coefficients and wavelet coefficients [5]. That is,

$$\int |f(t)|^2 dt = \sum_k |c_{j_0}(k)|^2 + \sum_k \sum_{j=j_0} |d_j(k)|^2. \quad (24)$$

Since the wavelet functions operate like high-pass filters that use narrow time windows to compute differences in signals [23], they can capture the variability of the incoming traffic volumes. On the other hand, the scaling functions perform like low-pass filters; therefore, they can be used to extract the trend of the outgoing TCP ACK traffic.

In order to realize an on-line detection, we use a moving window to group W continuous samples for the computation of DWT. Let $S = s(t), t \geq 1$, be the traffic samples, and $S_W(n) = \{s(t)\}_{t=(n-1)\times W+1}^{n\times W}, n \geq 1$, be the sequential windows of the samples. We use S^{In} and S^{Out} to denote the traffic samples for the incoming data traffic and outgoing ACK traffic, respectively. We also use $S_W^{In}(n)$ and $S_W^{Out}(n)$ to refer to the observation periods for the two respective cases.

Since the fluctuation of the incoming traffic can be captured by its high-frequency part, we continuously process $S_W^{In}(n)$ through the DWT and obtain their wavelet coeffi-

cients $d_{j,k}^{In}$. In order to quantify the degree of variability, we define a statistic based on the signal energy as follows.

$$E_H(n) = \frac{1}{W} \sum_k |d_{1,k}^{In}|^2, \quad (25)$$

where $d_{1,k}^{In}$ is the wavelet coefficient at the finest scale ($j = 1$). A similar approach was used in [17] to investigate the scaling properties of the network traffic.

On the other hand, we process $S_W^{Out}(n)$ to obtain the trend of the outgoing TCP ACK traffic. We also define a statistic based on the signal energy to represent the trend of the outgoing TCP ACK traffic as follows.

$$E_L(n) = \frac{1}{W} \sum_k |c_{L,k}^{Out}|^2, \quad (26)$$

where $c_{L,k}^{Out}$ is the scaling coefficient at the highest decomposed scale ($j = L$).

B The nonparametric CUSUM algorithm for change-point detection

In order to automatically locate the change point in the statistics E_H and E_L as soon as possible, we apply the nonparametric sequential detection algorithm at the end of every observation period. Here, we employ the nonparametric CUSUM algorithm for this purpose. This algorithm has also been used in other detection methods for D/DoS attacks [20, 12, 11].

The formal definition of the nonparametric CUSUM algorithm is summarized as follows [3, 4]:

$$y(n) = (y(n-1) + x(n))^+, \ \ y(0) \equiv 0, \ n = 1, 2, \ldots, \quad (27)$$

where $(y(n))^+$ is equal to $y(n)$ if $y(n) > 0$, and 0, otherwise. Its decision rule is:

$$d_N(\cdot) = d_N(y(n)) = I(y(n) > C_{cusum}), \quad (28)$$

where C_{cusum} is the threshold. $x(n)$ is defined on the probability space $(\Omega, \mathcal{F}, P)$ by the model

$$x(n) = a + h(n)I(n \geq m) + \xi(n), \quad (29)$$

where $\xi = \{\xi(n)\}_{n=1}^{\infty}$ is the random sequence such that its mathematical expectation $\overline{\xi(n)} \equiv 0$, and $\{h(n)\}$ is the deterministic sequence representing the profile of changes that take place at the moment m [4]. As suggested in [3, 11], we calculate the threshold C_{cusum} by the following equation:

$$C_{cusum} = (\tau - m)^+(h - \|a\|), \quad if \ m \geq 1, \quad (30)$$

where τ is the preferred detection time.

The CUSUM method assumes that $a < 0$ and $h + a > 0$, which together implies that the mean value of $x(n)$ will change from negative to positive when a change occurs. Therefore, it may necessary to first transform the statistics under the change-point detection to new random sequences which have negative mean values under normal conditions.

References

[1] The network simulator - ns-2. http://www.isi.edu/nsnam/ns/.

[2] M. Allman and V. Paxson. On estimating end-to-end network path properties. In *Proc. ACM SIGCOMM Conf.*, 1999.

[3] B. Brodsky and B. Darkhovsky. *Nonparametric Methods in Change-Point Problems*. Kluwer Academic Publishers, The Netherlands, 1993.

[4] B. Brodsky and B. Darkhovsky. *Non-Parametric Statistical Diagnosis Problems and Methods*. Kluwer Academic Publishers, The Netherlands, 2000.

[5] C. Burrus, R. Gopinath, and H. Guo. *Introduction to Wavelets and Wavelet Transforms: A Primer*. Prentice Hall, Upper Saddle River, NJ, 1998.

[6] C. Jin, H. Wang, and K. Shin. Hop-count filtering: an effective defense against spoofed DDoS traffic. In *Proc. ACM Conf. Computer and Communications Security (CCS)*, 2003.

[7] M. Carson and D. Santay. NIST Net: a Linux-based network emulation tool. *ACM Computer Communication Review*, 33(3), July 2003.

[8] R. Chang. Defending against flooding-based, distributed denial-of-service attacks: a tutorial. *IEEE Communications Magazine*, 40(10), 2002.

[9] M. Delio. New breed of attack zombies lurk. http://www.acm.org/technews/articles/2001-3/0514m.html, May 2001.

[10] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Networking*, 1(4):397–413, August 1993.

[11] H. Wang, D. Zhang, and K. Shin. Detecting SYN flooding attacks. In *Proc. IEEE INFOCOM Conf.*, 2002.

[12] J. Baras, A. Cardenas, and V. Ramezani. On-line detection of distributed attacks from space-time network flow patterns. In *Proc. 23rd Army Science Conf.*, 2002.

[13] A. Kuzmanovic and E. Knightly. Low-rate TCP-targeted denial of service attacks. In *Proc. ACM SIGCOMM Conf.*, August 2003.

[14] L. Feinstein, D. Schnackenberg, R. Balupari, and D. Kindred. Statistical approaches to DDOS attack detection and response. In *Proc. DARPA Information Survivability Conf. and Exposition*, 2003.

[15] M. Allman, V. Paxson, and W. Stevens. TCP congestion control. RFC 2581, April 1999.

[16] P. Barford, J. Kline, D. Plonka, and A. Ron. A signal analysis of network traffic anomalies. In *Proc. ACM Internet Measurement Workshop*, 2002.

[17] P. Huang, A. Feldmann, and W. Willinger. A non-intrusive, wavelet based approach to detecting network performance problems. In *Proc. ACM Internet Measurement Workshop*, 2001.

[18] K. Park and H. Lee. On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law Internets. In *Proc. ACM SIGCOMM Conf.*, August 2001.

[19] V. Paxson and M. Allman. Computing TCP's retransmission timer. RFC 2988, November 2000.

[20] R. Blazek, H. Kim, B. Rozovskii, and A. Tartakovsky. A novel approach to detection of denial of service attacks via adaptive sequential and batch-sequential change-point detection methods. In *Proc. IEEE Workshop on Information Assurance and Security*, June 2001.

[21] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1), January 1997.

[22] S. Floyd, M. Handley, and J. Padhye. A comparison of equation-based and AIMD congestion control. http://www.icir.org/tfrc/, May 2000.

[23] G. Strang and T. Nguyen. *Wavelets and Filter Banks*. Wellesley-Cambridge Press, Wellesley, MA, 1996.

[24] Y. Yang and S. Lam. General AIMD congestion control. In *Proc. IEEE Intl. Network Protocols*, 2000.