Malicious Packet Dropping: How It Might Impact the TCP Performance and How We Can Detect It

Xiaobing Zhang
Ericsson
Raleigh, NC

S. Felix Wu
U. of California
Davis, CA

Zhi Fu
NC State University
Raleigh, NC

Tsung-Li Wu
CCIT
Taiwan

Abstract

Among various types of denial of service attacks, "dropping attack" is probably the most difficult one to handle. This paper explores the negative impacts of packet dropping attacks and a method to detect such attacks. First, three dropping patterns are classified and investigated. We demonstrate that attackers can choose different dropping patterns to degrade TCP service to different levels, and selectively dropping a very small number of packets can result in a severe damage to TCP performance. Second, we show that a hacker can utilize a DDoS attack tool to control a "uncompromised" router to emulate dropping attacks. This proves that dropping attacks are indeed practically very possible to happen in today's Internet environment. Third, we present a statistic analysis module for the detection of TCP packet dropping attacks. Three measures, session delay, the position and the number of packet reordering, have been implemented in the statistic module. This paper has evaluated and compared their detection performance.

1. Introduction

Today's Internet is basically an unreliable environment, meaning that data can be lost during the transmission or out-of-order at receiver sites. The data loss is mainly caused by traffic overloaded, and the out-of-order delivery is mainly due to route flutter [10]. As the common bearer service in the Internet, Transmission Control Protocol (TCP) deals with this problem and provides a reliable flow of data between senders and receivers. To realize this, it employs slow start and congestion avoidance mechanism, acknowledges received packets, sets timeout to retransmit lost packets, and so on [1, 8, 11]. Applications such as FTP, HTTP and Telnet building on top of TCP achieve reliable data transmission without concerning these details. Nevertheless, the applications still face a threat if an intruder on the data path maliciously drops the TCP data packets, which we call TCP dropping. The quality of service (QoS) of the applications can be degraded drastically under such an attack. Traditional security mechanisms, such as embodied in IPSEC [9], can provide encryption and authentication, but they are unable to effectively handle denial of service attacks such as TCP dropping.

Few studies have been done on packet dropping attacks. The WATCHERS [3] from UC Davis was proposed to detect and react to routers that maliciously drop or misroute packets. WATCHERS is based on the "principle of packet flow conservation" (i.e., the number of incoming packets for a router, excluding those destined to it, should be the same as the number of outgoing packets, excluding those generated by it). In order to validate the conservation law, multiple decentralized counters are periodically and synchronously exchanged among neighbors of the target suspected router. Subsequently, each neighboring router runs a validation algorithm to diagnose the health condition of the target router. Furthermore, WATCHERS is robust against Byzantine faults. While WATCHERS offers theoretically an interesting way to deal with malicious packet

This work has been supported by the Defense Advanced Research Projects Agency and the U.S. Air Force Rome Laboratory under contracts F30602-99-1-0540 (ArQos), F30602-98-C-0249 (Deciduous) and DABT63-97-C-0045 (Celestial).

dropping, at least in its current form, it can not handle the packet dropping problems in today's Internet effectively. First, the number of messages for counter value exchanges can be very large (N3, where N is the number of nodes). Second, the "principle of packet flow conservation" does not hold "deterministically" for today's Internet environment. For instance, an innocent router might drop packets for good reasons such as preventive congestion control or insufficient resources to keep all incoming packets. In summary, we are still far from an effective solution to protect our network users from denial of service attacks such as packet dropping, and we can not even distinguish whether the packet dropping we observed is statistically normal, due to a fault or a malicious attack, or both.

Detecting dropping attacks is an open and difficult, and in this paper, we merely started the effort by providing a solution to effectively distinguish the malicious dropping from natural/normal dropping. First, we have empirically investigated the impacts of TCP packet dropping attacks. As a result, we are able to show that the attacker can control the rate of dropping to degrade the service to different levels. In another case, the attacker can selectively drop a very small number of packets to degrade the service dramatically. Second, we show that a hacker can utilize the TFN (DDoS) attack tool to control an "uncompromised" router to drop a small amount of traffic, which shows that dropping attacks are indeed possible to happen today. For instance, unlike the web site shutdowns happened earlier this year, the hacker can controllably flood "a little" to "degrade" the response time of a particular web site (e.g., yahoo) such that its visitors will eventually switch to other service providers. Third, we have designed and implemented a statistic-based analysis module to detect TCP dropping attacks. In particular, we have applied the NIDES/STAT method on three different statistic measures related to TCP behavior: session delay, position of packet reordering, and the number of reordering per connection. For evaluation, we have implemented an intrusion detection system on the LINUX kernel with divert socket support, and conducted experiments across four FTP sites distributed internationally. As a result, each individual statistic measure has some weakness in identifying the malicious attacks. And, in some cases, we got high false positive. When we combine the results for three measures together, we can cover most of the attacks. We also conjecture that the undetected attacks are those causing very limited damages. Furthermore, our detection system runs on the client side (i.e., the subscriber's machine). Thus, our approach can be used to help *Service Level Agreement* (SLA) subscribers to determine statistically how close the SLA has been supported by the service provider.

In Section 2, we describe three packet dropping patterns, and the potential ways to practice packet dropping attacks, and discuss their negative impacts according to the experimental results. Section 3 presents a statistic-based method for detecting TCP packet dropping attacks and reports on the experiment results demonstrating the feasibility of the scheme. Section 4 briefly summarizes our findings with a discussion of limitations and potential extension.

2. TCP Packet dropping attacks

Among many schemes that are used to implement denial of service attacks, a packet dropping attack is probably considered one of the hardest to handle, because neither senders nor receivers have the knowledge of where and why packets being dropped. An attacker (e.g., a bad router) can simply drop all packets it receives. In this case, it acts like an inoperable router, which can be detected by using existing tools such as *traceroute*. Other routers will avoid sending packets to the bad router and finally remove it from the routing path in the Internet. Tricky attackers only drop a portion of packets selectively. It is not trivial to recognize the impacts caused by TCP packet dropping attacks, because the reaction of TCP to packet loss is complicated and dynamic. TCP's sending rate will be restrained at some level during the period of packet loss, and the effects caused by the packet loss will depend on various factors such as traffic load, network configuration, and applications.

Generally, packet dropping attacks can impact a network service on the following several aspects: ***Delay***: e.g., dropping the retransmissions of packets in a FTP connection will drastically increase the total file transfer time; ***Response time***: e.g., if the DNS query packets are dropped, a user may feel waiting for a long time to get a web page; ***Quality***: e.g., dropping some packets of MPEG video stream or IP telephoning data flow can degrade the quality of the service; ***Bandwidth***: because dropping

packets usually introduces packet retransmissions, which waste network bandwidth.

In this paper, we study how packet dropping attacks affect FTP file transfer. Since file transfer services have comparatively low requirements on response time and quality, we focus on the dropping attacks' negative impacts on session delay.

2.1. Dropping patterns

We distinguish two types of dropping attacks, *persistent* and *intermittent* dropping attacks. The former performs attacks on every FTP connection, while the later only attacks a portion of the connections. For example, an intermittent attacker can attack 20% of all the connections (i.e., one in every five). An attacked connection is called a *victim connection* and packets dropped by attackers are called *victim packets*. In each victim connection, we further classify three dropping patterns as follows.

Periodic packet dropping (PerPD): Packets are periodically dropped in the victim connection according to 3 parameters, (K, I, S). *K* is the total number of victim packets in the connection. *I* is the interval between two consecutive victim packets, and *S* is the position of the first victim packet in the connection. For example, a pattern defined as (K=5, I=10, S=4) means that 5 packets will be dropped in each victim connection, once every 10 packets, starting from the 4th packet seen by the attacker.

Since the *congestion window size* (cwnd) of TCP connections will be reduced to one half of the current value once a packet is lost [1], malicious dropping slows down the transmission rate. The more the packets are dropped, the more the TCP performance is degraded. However, attackers who drop too many packets may be identified easily.

Retransmission packet dropping (RetPD): Attackers always drop the retransmissions of a specific packet. Two parameters, (K, S), are defined for this pattern. *S* denotes the victim packet. *K* is the number of times dropping the packet's retransmissions. For example, given a pattern, (5, 10), the attacker first drops the 10th packet and then its subsequent retransmissions 5 times.

When a retransmission packet is lost, TCP goes back to slow start phase and exponentially backoffs its retransmission timeout value (RTO) upon every packet loss, with an upper limit of 64 seconds. We can infer that after a few consecutive retransmissions being dropped, the sender has to wait for a long period of idle time before performing a new retransmission. Normally, no packets are sent out during this idle period. Thus, in [13], through NS2 simulation, we showed that retransmission packet dropping attacks can degrade the TCP's performance greatly by dropping only a few packets. In addition, since TCP connection gives up after sending retransmissions of a packet about 12 times, attackers can easily terminate the TCP service by dropping retransmissions.

Random packet dropping (RanPD): Attackers randomly choose up to *K* packets to drop in a connection. Since this kind of attacks behaves more or less like "natural dropping" by the Internet, *the fast retransmit and recovery* mechanisms [1, 11] may work effectively in response to packet loss. From this point of view, we expect that randomly dropping has limited impacts on the TCP's performance, compared to other dropping patterns.

2.2. Dropping methods

One of the ways for an attacker to achieve packet dropping attacks is compromising a router, where he can manipulate the victim's traffic at will. The argument may be that attackers can do more malicious things once a router is compromised rather than simply dropping some packets. However, since the packet dropping is limited understood and difficult to detect, there is still a motivation for attackers to perform such kind of attacks on a compromised router.

Instead of compromising a router, another (or easier)

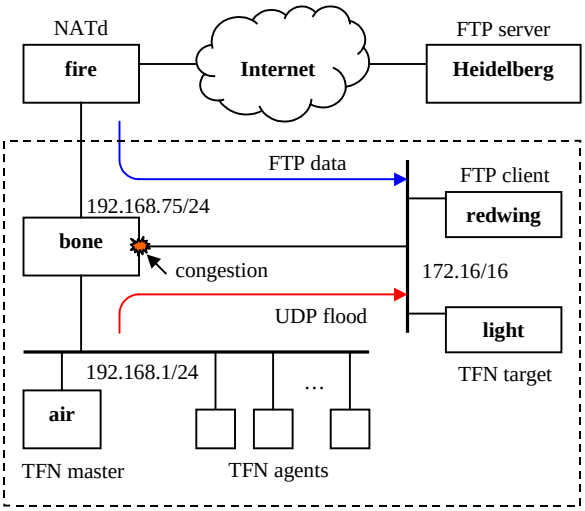


Figure 1: Network used in congestion experiment

way to achieve packet dropping is intentionally congesting a router. The following experiment shows that an attacker can cause packets loss and degrade the performance of other's FTP connection in this way. The experiment setting is shown in Figure 1.

The testbed network was located in the SHANG lab at North Carolina State University. It consisted of three subnets, 192.168.75/24, 192.168.1/24 and 172.16/16. It was connected to the Internet by a PC called *fire*, on which the Network Address Translation daemon (NATd) was running. Eight other PCs participated the experiment. Four of them were named as *air*, *bone*, *light* and *redwing*, respectively. All the PCs ran on Linux 2.0.36 except for the *fire*, which used FreeBSD 2.2.8 instead. The *bone* using a 500MHz CPU acted as an intermediate router to forward IP packets. The FTP server called *Heidelberg* was located in Europe (see Table 2), and the FTP client was operated on the *redwing*, which downloaded a file of 5.6M bytes from the *Heidelberg*.

A distributed denial of service attack tool, called *Tribe Flood Network 2000* (TFN2k) was used to congest the router, *bone*. TFN2K comprises two components: the *master* and *agent*. The master employs a program, which is referred to as *client*, to instruct its agents to attack a list of designated targets. On each agent, there is a *daemon* process, responsible for receiving and carrying out commands issued by a client. The agents can flood the targets with a barrage of packets [4, 6]. In the experiment, the *air* played the role of a TFN master and it has four TFN agents. The master and agents were all connected to the same Ethernet (192.168.1/24). The flooding target was the *light*, residing on the same subnet as the FTP client *redwing*. The *air* periodically instructed its agent to launch UDP packet flood. The flooding packets were forwarded by the *bone*. Since the FTP data traffic was also forwarded by the *bone* to the same subnet (172.16/16) as the UDP flood, they shared the outgoing interface on the *bone*. During the TFN attacks, the flood of UDP packets congested the interface and thus caused the *bone* to drop packets.

Figure 2 illustrates the FTP data packet loss under the UDP flood attacks. Only the first 50 seconds of the connection are shown. The *flood m, stop n* in the figure means that the TFN agents flood the victim for *m* seconds, stop for *n* seconds, and flood for another *m* seconds and so on. The *flood intensity* is defined as *m*/*n*. The *flood period* is equal to (*m* + *n* + *overhead*), where the overhead is the time for initiating and stopping the attacks. In the experiment, the overhead was about 4 seconds. From Figure 1, it is clear that there was a periodical burst of packet loss during the FTP connection, which is consistent with the flood period. The results

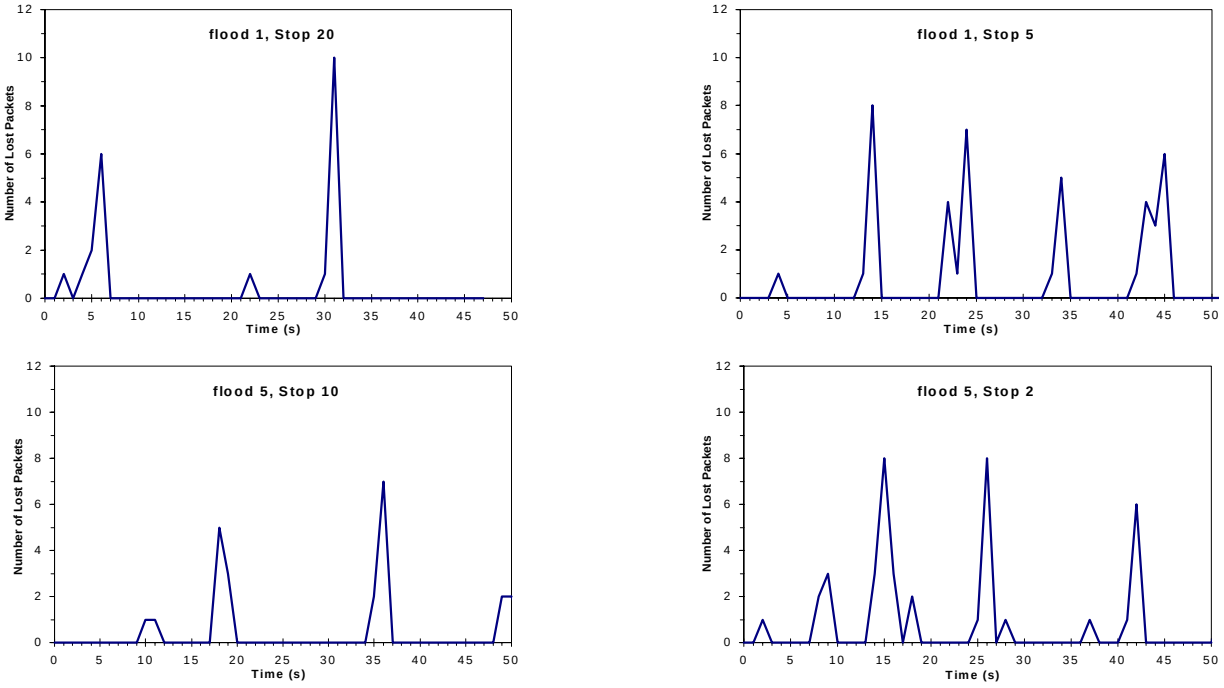


Figure 2: FTP data packet loss under UDP flood attacks

demonstrate that an attacker can periodically cause the router to discard packets by a periodical congestion.

Table 1 presents the average impacts of the UDP flood attacks on the FTP connection. As expected, increasing the flood intensity correlates with an increase in the number of packet loss per connection as well as the session delay. By controlling the flood intensity, attacker can degrade the performance of other's FTP connection to different levels.

Table 1: Impacts of UDP flood attacks on FTP

Attack mode (flood *m*, stop *n*)	Number of packet loss per connection	Session delay (sec.)	Damage
Normal	0.9	31.7	-
Flood 1, stop 20	18.5	470.5	27.8%
Flood 1, stop 5	57.4	58.4	84.5%
Flood 5, stop 10	62.1	67.3	112.6%
Flood 5, stop 2	124.4	164.5	418.9%

Damage = $(\text{delay}_{\text{flood}} - \text{delay}_{\text{normal}}) / \text{delay}_{\text{normal}}$

2.3. Empirical study of impacts of packet dropping attacks

2.3.1. Experiment Setting. In order to investigate the impacts of packet dropping attacks, we did various experiments using different dropping patterns. All experiments were based on FTP bulk data transmission. Because of the choice of FTP, our empirical results only reflect the bad effects to FTP traffic.

The session delay of FTP file transfer was studied. The file transfers were conducted between 4 client-server pairs in the Internet, of which 4 clients were located in the SHANG lab, and the 4 servers in the North America, Europe and Asia, respectively (see Table 2). The data file was a shareware (5.5M) being mirrored to server sites.

Table 2: FTP severs used in the experiment

Name	FTP Server / IP Address	Location
Heidelberg	ftp.uni-heidelberg.de 129.206.100.134	Europe
NCU	ftp.ncu.edu.tw 140.115.1.71	Asia
SingNet	ftp.singnet.com.sg 165.21.5.14	Asia
UIUC	ftp.cso.uiuc.edu 128.174.5.14	North America

All the clients ran on Linux 2.0.36 with IP firewall and divert socket support. In order to simulate an attacker, a program named *attack-agent* was executed on each client. For every arriving FTP data packet, the attack-agent captured it through a divert socket and then released or dropped the packet according to its dropping pattern.

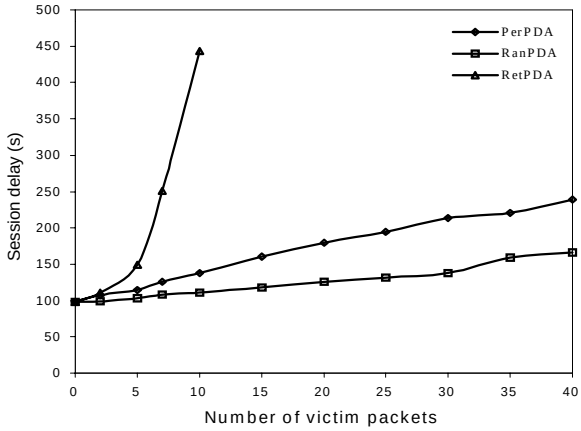


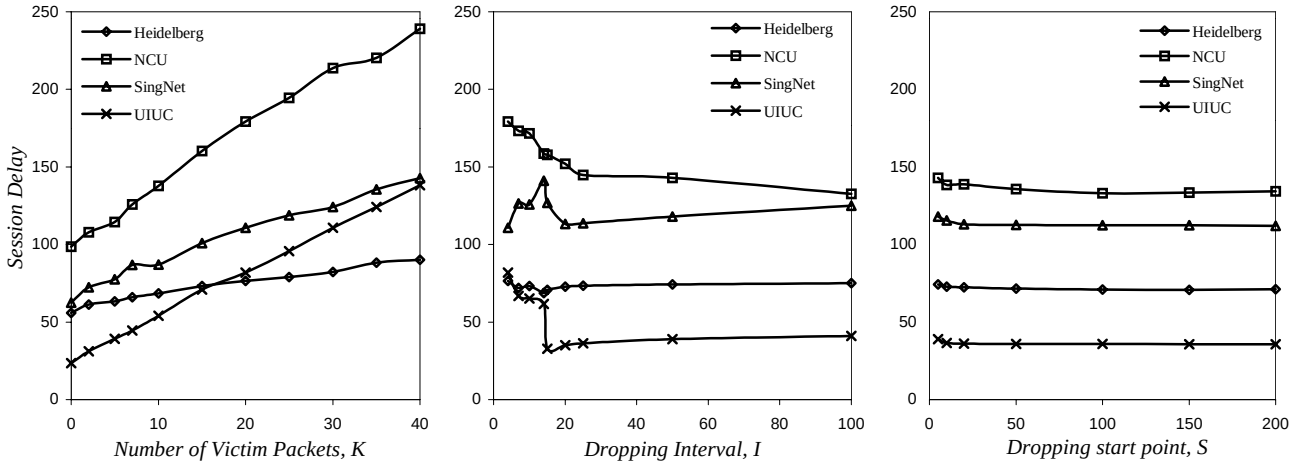
Figure 3: Session delay for the NCU site under three dropping patterns

I=4 and S=5 for PerPD, and S=5 for RanPD.

2.3.2. Results. Figure 3 demonstrates the impacts of three dropping patterns for the NCU site (National Central University in Taiwan). Other sites show similar behavior under these attacks. It is clear from the Figure 3 that given the same number of victim packets, the retransmission packet dropping attack causes the most. It is interesting though that periodic packet dropping causes significantly more damage than random packet dropping.

Periodic packet dropping (PerPD). Results show that the impacts of PerPD attacks vary with the setting of (K, I, S). Figure 4(a) displays the session delay of 4 sites with different values of *K* while *I* is set to 4 and *S* to 5. As expected, given a fixed *I* and *S*, increasing the number of victim packet correlates with an increase in session delay. For instance, for the NCU site, the delay was 137.8 seconds when 10 packets were periodically dropped. It moved up to 239.1 seconds when 40 packets were dropped. Furthermore, we see that session delay increase approximately linearly with the increase of *K*, which indicates that each PerPD's packet dropping causes about same amount of damage at a site. However, the different slopes of the curves demonstrate that packet dropping results in different amount of damage at different sites.

Figure 4(b) presents the session delay with fixed value of *K* and *S*. The points of the curves whose *I* value is less than 4 are removed from the plot, because many retransmissions were dropped at that time, which makes the dropping behave more like a retransmission dropping



(a) I=4, S=5 (b) K=20, S=5 (c) K=20, I=50

Figure 4: The impact of periodic dropping attacks with different values of dropping parameters

rather than a periodic one. The results show that with an increase in the dropping interval *I*, session delay experiences two phases, which are separated by a certain value, say I_{fr}. In the first phase, where *I* is not larger than I_{fr}, session delay tends to reduce with an increase in dropping interval. For instance, the session delay for the UIUC site was 81.8 seconds when *I* equals 4. It gradually reduced to 61.7 seconds with the *I* increasing up to 14, and sharply fell to 32.8 seconds when *I* reached the 15, the I_{fr}. In the second phase, where *I* is larger than I_{fr}, increasing the dropping interval could either decrease or increase session delay slightly. For example, while the delay moved up from 32.8 to 41 seconds for the UIUC site when *I* increased from 15 to 100, it decreased from 157.8 to 132.6 seconds for the NCU site. This two-phase phenomenon can be explained by examining the TCP congestion window size (cwnd).

In the first phase, small dropping interval *I* keeps the *cwnd* small, which gives the sender few chances to receive 3 duplicate ACKs when a packet is lost. Thus, packets are retransmitted due to timeout rather than fast retransmit. The closer the *I* to the I_{fr}, the more chances the lost packets can be repaired by fast retransmit. Thus we observe a decrease in the first phase. Once the dropping interval *I* reaches the I_{fr}, fast retransmit can be applied to all packet loss, leading to a sharp drop in session delay at that point. The value of I_{fr} is site specific, e.g., it is 15 for the UIUC site and 25 for SingNet site. For the Heidelberg site, however, the decrease is not clear. This is because that the first RTO value calculated at the Heidelberg was almost same as the time used for fast retransmit (This site seems very aggressive). For the SingNet site, the curve ascending at the beginning is due to that some redundant retransmissions happened to be dropped at that time.

In the second phase, increasing the dropping interval *I* results in two opposite impacts. On one hand, it makes the dropping affect more packet transmissions, which may have been delivered by using a large *cwnd* otherwise. On the other hand, a large dropping interval gives the sender more time to recover its *cwnd* from the latest packet loss. Thus, the *cwnd* could be big enough to enable the fast retransmit when the next dropping occurs. The results show that the former dominated at the Heidelberg, SingNet and UIUC sites, and the later dominated at the NCU site. It seems likely that a connection-pair with a high transmission rate favors a small dropping interval, while a slow connection-pair favors a large one. This is probably due to that a fast connection-pair is usually related to a small RTT and large *cwnd*. Therefore, the fast retransmit is usually applied to the repair of packet loss and the sender does not have to wait for the 3 duplicate ACKs for a long time. A slow connection-pair however is usually related to a large RTT and high packet loss rate. The natural packet loss will keep the *cwnd* small, which minimizes the advantage of a small dropping interval. Moreover, because of the large RTT, the sender may wait for a long period of idle time for the 3 duplicate ACKs.

The impacts of the parameter *S* on session delay are shown in Figure 4(c). The results demonstrate that early dropping tends to cause a little larger session delay. One possible reason for this is that early packet loss brings a connection from the slow start phase into the congestion

avoidance phase when the *cwnd* value is small. The additive increase makes the *cwnd* move up slowly.

Retransmission packet dropping (RetPD). From Figure 3, we see that the session delay corresponding to the RetPD increases exponentially with the increase of the *K*. This supports our simulation results [13] that retransmission dropping attacks can drastically degrade the TCP's performance by only dropping a few packets. For example, dropping a packet's retransmissions 5 times for the NCU site increased the average of session delay (98.6) by 50.5 seconds, more than 50%. As expected, all sites sent a reset and terminated the connection after around 12 retransmissions failed. This provides an easy way for attackers to terminate a TCP connection.

Random packet dropping (RanPD). In Figure 3, the curve corresponding to the RanPD has the smallest slope among the three patterns, which indicates that random dropping attacks do not increase session delay remarkably, compared to PerPD and RetPD attacks. For instance, when the number of victim packet was 10, a periodic dropping attack with interval 4 incremented the session delay by 39.2 seconds and a retransmission attack by 344.6 seconds. Randomly dropping 10 packets however only increased the delay by 12.7 seconds. This is due to that randomly dropped packets are often fast retransmitted in response to 3 duplicate ACKs, which does not impede the transmissions of subsequent packets.

3. Detection of malicious packet dropping

In Section 2, we have demonstrated empirically that packet dropping attacks with different parameters cause different levels of damages to the TCP performance. In this section, we discuss a "statistic-based" intrusion detection system running on the ftp client side to determine whether a particular TCP flow is under malicious dropping attack or not. Please note that we only consider the problem of detection in this paper. And, how to prevent/avoid the damages or how to identify the exact attack sources are out of the scope of this paper.

3.1. TCP-Dropping statistic analysis module

We propose an intrusion detection system approach, TCP-Dropping Statistic Analysis Module (TDSAM), to handle the TCP packet dropping attack problem. It is based on the NIDES/STAT algorithm developed by Stanford Research Institute International. This algorithm has also been successfully used in North Carolina State University's JiNao project [12].

NIDES/STAT describes subjects' behavior by means of profiles, which are separated into short-term and long-term components. Aspects of a profile are represented as measures in the way of probability distribution. NIDES defines four classes of measures: ***Activity Intensity Measures***, measuring whether the activity volume is normal; ***Categorical Measures***, whose values are by nature categorical, such as the TCP packet reordering; ***Continuous or Counting Measures***, whose values are numeric; ***Audit Record Distribution***, which monitors the distribution of all types of activities that have been generated in the recent past.

In terms of the probability distribution computation, each class of measures is done by the way of "binning" procedure for establishing histogram. Measures can have fixed bin end-points, e.g., the position of an out-of-order packet ranges from 1 to the last packet. On the other hand, for some measures, the bin end-points have to be determined first through other measures. For example, in our experiment, the lower bound of session delay was determined by the minimal observed value, while the upper bound was determined by mean value plus 4 times of the standard deviation value of session delay. After the end points have been determined, the bins can be scaled either in a linear or geometric fashion. In particular, the TDSAM uses linear scaling.

NIDES/STAT algorithm monitors a subject's (either a user or a software program) behavior on a computer system, and raises alarm flags when the subject's current behavior (short-term profile) deviates significantly from its expected behavior, which is described by its long-term profile. NIDES's algorithm is based on a χ^2-like test [5] for comparing the similarity between the short-term and long-term profiles. Three measures, position, delay and number of packet reordering (NPR), have been implemented in the TDSAM to reflect TCP behavior. The *position measure* builds the long-term profile based on the position of out-of-order packets in each connection, the *delay measure* on the session delay and the *NPR measure* on the number of occurrences of packet reordering in each connection. Due to the space limitation, we will skip the

detailed design and implementation of the TDSAM. Please refer to [14] for further information.

3.2. Intrusion detection results

In order to evaluate and compare the performance of the TDSAM measures, we conducted experiments over the Internet. The same FTP severs, file and attack-agent mentioned in Section 2.3.1 were used. The long-term profile was established based on 20000 FTP connections and updated daily when there is no attacks observed during the day.

In the following discussion, *nbin* means the number of bins used in the long-term distribution, and all attacks are *persistent attacks* unless specifically notified. Due to the limited space, we only show the *detection rate* (DR) in the following tables. The *false negative rate* can be derived from the detection rate by (1-DR). Two *intermittent attacks* are shown at the end of Table 3, 4 and 5. In particular, one used an attack interval of 5, which means that the attacker attacked one per 5 connections, and the other used an attack interval of 50. In every victim connection of the intermittent attacks, we chose a periodic dropping pattern (10, 4, 5).

Position Measure. Table 3 summarizes the intrusion detection results of the position measure where *nbin=5*. The results show that the TDSAM has a high detection rate for most periodic dropping attacks. This is due to that normal packet reordering are nearly uniformly distributed across a connection, while periodic dropping attacks usually generate an extremely different distribution. For instance, the long-term profile for the Heidelberg site was: p_1=0.194339, p_2=0.200759, p_3=0.197882, p_4= 0.204260, p_5=0.202760, where bin width was 800. The PerPD (20, 4, 5) dropped the first 85 packets of a connection, resulting in a large number of occurrences of packet reordering in the first bin. Under the attack, the corresponding distribution reordering became: p_1= 0.837264, p_2=0.039390, p_3=0.043192, p_4=0.041045, p_5= 0.039109. Comparing it with the long-term profile, the TDSAM can easily detect such deviation and raise alarm.

However, it is hard for the position measure to detect an attack that generates a distribution of packet reordering similar to the long-term profile. In this case, we say that the short-term distribution is *long-term-profile-like*. Note that for the Heidelberg, NCU and SingNet sites, the number of total data packets transmitted in a connection is about 4000. The dropping patterns, (20, 200, 5) and (100, 40, 5), evenly distributed the victim packets over the connection and thus generated a long-term-profile-like distribution. From Table 3, we see that the detection rate for these attacks can be as low as 0%. For the UIUC site, since the number of data packets per connection is about 11300, the two patterns, (20,200, 5) and (100, 40, 5), did not generate a uniform distribution in the bins. Instead, all victim packets fell into the first 2 bins. Therefore, the corresponding detection rates are high.

The results also demonstrate that the TDSAM performs poorly on discovering random packet dropping attacks. It is because that RanPD attacks usually generate a nearly even distribution of packet reordering, which is long-term-profile-like as well. The TDSAM using position measure may be totally out of function in this case.

The TDSAM has a comparatively high detection rate for retransmission dropping attacks. Although one attack may impact the distribution of packet reordering little, the aggregated impacts of many attacks can result in abnormally high occurrences of packet reordering at some position, because RetPD attacks always drop packets at a fixed position in a connection.

Table 3: Detection rate for position measure

Position nbin=5	Dropping pattern	Heidelberg	NCU	Sing-Net	UIUC
Normal*	-	4.0%	5.4%	3.5%	6.5%
PerPD	(10,4,5)	99.7%	100%	100%	100%
	(20,4,5)	100%	98.1%	99.2%	100%
	(40,4,5)	96.6%	100%	100%	98.5%
	(20,20,5)	100%	100%	100%	100%
	(20,100,5)	98.9%	99.2%	99.6%	99.1%
	(20,200,5)	0%	76.5%	1.5%	98.3%
	(100,40,5)	0.2%	0%	0%	100%
RetPD	(5,5)	84.9%	81.1%	94.3%	97.4%
RanPD	10	0%	42.3%	0%	0%
	40	0%	0%	0%	0%
Intermittent (10, 4, 5)	5	98.6%	100%	98.2%	100%
	50	34.1%	11.8%	89.4%	94.9%

* False positive rate for normal cases

For intermittent attacks, the performance of the TDSAM depends on the attack interval, binning mechanisms and specific sites. Generally, attacks using a small attack interval can be easily detected and increasing *nbin* correlates with an increase in detection rate. For the sites having a low packet reordering rate, such as the SingNet and UIUC, the detection rate remains high when

the attack interval becomes large. It is due to that it is hard for attacks to hide in the normal dropping behavior.

Delay Measure. The intrusion detection results are shown in Table 4. The results indicate that attacks that significantly delay a connection are easily to be identified. For example, the retransmission dropping attack with parameter (5, 5), incrementing by around 40 seconds session delay, corresponds to a high detection rate. While randomly dropping 10 packets increases session delay a little and thus is not easy to detect, randomly dropping 40 packets corresponds to a large session delay and thus a high detection rate.

Table 4: Detection rate for delay measure

Delay nbin=3	Dropping pattern	Heidelberg	NCU	Sing-Net	UIUC
Normal*	-	1.6%	7.5%	2.1%	7.9%
PerPD	(10,4,5)	97.4%	95.2%	94.5%	99.2%
	(20,4,5)	99.2%	98.5%	100%	100%
	(40,4,5)	100%	100%	100%	100%
	(20,20,5)	96.3%	100%	92.6%	98.9%
	(20,100,5)	100%	95.3%	98.7%	100%
	(20,200,5)	98.6%	99%	97.1%	100%
	(100,40,5)	100%	100%	100%	100%
RetPD	(5,5)	100%	100%	100%	100%
RanPD	10	74.5%	26.8%	67.9%	99.5%
	40	100%	100%	100%	100%
Intermittent (10, 4, 5)	5	25.6%	0%	0%	97.3%
	50	0%	24.9%	0%	3.7%

* False positive rate for normal cases

We notice that the position measure has much better performance than the delay measure on the detection of the intermittent attacks with the periodic dropping pattern (10, 4, 5). This is because that periodically dropping 10 packets impacted session delay slightly (except for the UIUC site), but deviated the normal distribution of the position of packet reordering greatly.

The results also show that a site that has stable and low session delay is related to a high detection rate. Clearly, the TDSAM for the UIUC site outperformed others, because the session delay for the site was only 23.6 seconds in average, far less than the others'. Maliciously dropping a few packets resulted in a comparatively long delay for the UIUC site. For example, dropping 10 packets periodically incremented by 30.5 seconds the session delay, 129% larger than the normal one and thus was easy to be detected. Additionally, the session delay for the UIUC site was stable. We observed that for the session delay of the 20000 connections in the experiment, its standard deviation was only 1.4. In other words, a site having such kinds of characteristics is sensitive to slight change in session delay.

Number of Packet Reordering (NPR) Measure. Table 5 demonstrates the intrusion detection results using the NPR measure. As expected, the NPR measure has a high detection rate for those attacks dropping large number of packets. For RetPD and another dropping patterns with a few victim packets, however, it works poorly. The comparatively high detection rate for RetPD (5, 5) for the NCU site is due to a significantly high occurrences of packet reordering during the detection period, which might be caused by route flutter.

The results show that the NPR measure did not work very well on detecting periodic dropping attacks (10, 4, 5), especially for those sites that usually experience many packet reordering in a connection, such as the Heidelberg and NCU sites. For these sites, 10 dropped packets were only a small portion of their total packet loss. Since the SingNet and UIUC sites generally experienced little packet reordering, the TDSAM could more easily detect the change in the number of out-of-order packets for these sites. Consequently, in most cases, they were related to a higher detection rate than the other two sites.

Table 5: Detection rate for NPR measure

NPR nbin=2	Dropping pattern	Heidelberg	NCU	Sing-Net	UIUC
Normal*	-	4.5%	5.8%	8.2%	2.9%
PerPD	(10,4,5)	0%	14.4%	29.1%	100%
	(20,4,5)	83.1%	94.2%	95.2%	100%
	(40,4,5)	100%	97.4%	100%	100%
	(20,20,5)	91.6%	92%	93.5%	100%
	(20,100,5)	94.3%	92.2%	96.4%	100%
	(20,200,5)	0%	96.5%	94.8%	100%
	(100,40,5)	100%	100%	100%	100%
RetPD	(5,5)	0%	84.7%	23.9%	46.5%
RanPD	10	0%	0%	100%	100%
	40	100%	100%	100%	100%
Intermittent (10, 4, 5)	5	0%	0%	82.2%	100%
	50	0%	1%	40%	64.8%

* False positive rate for normal cases

4. Discussion

In this paper, we have investigated the impacts of a set of TCP Packet dropping attack patterns on FTP file transfer and proposed a statistic-based approach to detect the attack. Our results indicate that the impacts of a periodic dropping attack depend on its parameters (*K, I, S)*. A retransmission packet dropping attack severely degrades TCP's performance by only dropping a few

packets. Among the three dropping patterns, random packet dropping attacks cause the least damage, given the same number of victim packets.

The TDSAM approach is a feasible way to detect some kinds of TCP packet dropping attacks. The position measure works well for most attacks whose victim packets are position-related. The delay and NPR measures have a good performance on discovering the attack that significantly changes delay or the number packet reordering per connection. However, each individual statistic measure has some weakness. When combining the 3 measures, the TDSAM can detect most kinds of attacks except for the persistent RanPD attacks dropping a small number of packets and some intermittent attacks that use a large attack interval or does not affecting delay, the position and the number of packet reordering significantly. Fortunately, all these attacks generally do not cause much damage as others.

We believe that malicious packet dropping is a very difficult problem to handle. For instance, we have not yet been able to prove analytically that our attack modes capture all classes of serious TCP dropping attacks. Also, at this moment, we only demonstrated that our system works for FTP traffic. Furthermore, in the near future, we plan to devise a countermeasure to defend the QoS against TCP packet dropping attacks. For instance, if a connection is identified to be under a particular type of packet dropping attacks, then how we can adjust the TCP behavior to minimize the damage?

5. Acknowledgment

Dr. Frank Jou from MCNC provided many valuable comments about NIDES/STAT. Chien-Long Wu and Ravindar Narayan implemented the LINUX divert sockets for our experiments.

6. References

[1] M. Allman and V. Paxson, TCP Congestion Control, RFC 2581, April 1999.

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

[3] K. A. Bradley, S. Cheung, N. Puketza, B. Mukherjee and R. A. Olsson, Detecting Disruptive Routers: A Distributed Network Monitoring Approach, In the 1998 IEEE Symposium on Security and Privacy, May 1998.

[4] J. Barlow and W. Thrower, TFN2K – An Analysis, Axent Security Team, February, 2000.

[5] Jay L. Devore, Probability and Statistics for Engineering and the Science, Brooks/Cole Pub. Co., 1991.

[6] D. Dittrich, The "Tribe Flood Network" distributed denial of service attack tool, University of Washington, October, 1999.

[7] Sally Floyd and Van Jacobson, Random Early Detection Gateways for Congestion Avoidance, IEEE/ACM Transactions on Networking, Vol. 1, No. 4, page 397-413, August, 1993.

[8] Van Jacobson, Congestion Avoidance and Control, SIGCOMM'88, August 1988.

[9] S. Kent and R. Atkinson, Security Architecture for the Internet Protocol, RFC 2401, November 1998.

[10] Vern Paxson, End-to-End Internet Packet Dynamics, Proc. SIGCOMM'97.

[11] W. Richard Stevens, TCP/IP Illustrated, Volume 1: the Protocols, Addison Wesley, 1994.

[12] S. F. Wu, H. C. Chang, F. Jou, F. Wang, F. Gong, C. Sargor, D. Qu and R. Cleaveland, JiNao: Design and Implementation of a Scalable Intrusion Detection System for the OSPF Routing Protocol, February, 1999.

[13] Tsung-Li Wu, Securing Internet QoS: Threats and Countermeasures, Ph.D. Thesis, North Carolina State University, 1999.

[14] Xiaobing Zhang, TCP Packet Dropping Attacks and Intrusion Detection, M.S. Thesis, North Carolina State University, June, 2000.