國　立　交　通　大　學

資　訊　工　程　學　系

碩　士　論　文

TCP 低速阻斷服務攻擊之預防

Preventing Low-Rate TCP Targeted Denial-of-Service Attack

研　究　生：陳志偉

指導教授：謝續平　博士

中華民國九十三年六月

TCP 低速阻斷服務攻擊之預防

Preventing Low-Rate TCP Targeted Denial-of-Service Attack

研 究 生：陳志偉　　　Student: Zhi-Way Chen

指導教授：謝續平 博士　　Advisor: Dr. Shiuh-Pyng Shieh

國 立 交 通 大 學

資 訊 工 程 學 系

碩 士 論 文

A Thesis
Submitted to
Department of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Chiao Tung University
In Partial Fulfillment of the Requirements
For the Degree of
Master
In

Computer Science and Information Engineering

June 2004

Hsinchu, Taiwan, Republic of China

中華民國九十三年六月

TCP 低速阻斷服務攻擊之預防

研究生：陳志偉　　　　　　　　　　　　　指導教授：謝續平

國立交通大學　資訊工程學系

摘　要

在 TCP 裡，錯誤的逾期會造成相當大的傷害，而已經存在有兩種類的方法來減輕這個問題，我們稱他們為保守措施與積進措施，在保守措施裡有用到一個保守重送逾期值(RTO, Retransmission Timeout) minRTO，當根據 RTT (Round Trip Time) 所估算出來的 RTO 小於 minRTO 時，RTO 就會被設為 minRTO，這樣的做法會讓 RTO 變成是可預測的，進而可能被攻擊者利用。TCP 低速阻斷服務攻擊(Low-Rate TCP Targeted Denial-of-Service Attack) 就是屬於這種攻擊，在本論文裡，我們提出了四種方法來處裡這個問題，在不改變保守 RTO 的前提下，我們讓 RTO 變得不可預測，我們也做了分析和實驗來顯示我們的方法在遭預這種攻擊時可以存活下來。在我們的方法裡，當遭遇攻擊時能達到的平均流量會比改善前多出許多，另一方面，在平常的狀況下，我們的方法幾會不會影響 TCP 的表現。我們也分析了 TCP 低速阻斷服務攻擊是如何影響 TCP 的運作即使在攻擊沒有成功時。

Improvement of TCP Conservative Retransmission Timeout

Student: Zhi-Way Chen Advisor: Shiuh-Pyng Shieh

Department of Computer Science and Information Engineering
National Chiao Tung University

Abstract

Spurious timeout is very harmful to TCP. Two kinds of approaches were proposed to mitigate this problem, conservative approach and aggressive approach. In conservative approach, a conservative RTO (Retransmission Timeout), minRTO, is used. When RTO estimate form RTT (Round Trip Time) is less than minRTO, it is set to minRTO. This makes RTO become predictable and may be exploited by attackers. Low-Rate TCP Targeted Denial of Service is such kind of attack. In this thesis, we propose four schemes to deal with this problem. We make RTO unpredictable and keep conservative property. Analysis and experiment will be made to show that our schemes survive the attack. In our scheme, much better throughput is gained under attack. Besides, in usual condition, our schemes affect TCP performance only slightly. We will also analysis how Low-Rate TCP Targeted Denial of Service damages TCP operation inherently and how to gain the best throughput even if it is not achieved successfully.

誌　謝

首先要感謝謝續平老師這兩年來不厭其煩的指導，給予許多寶貴的意見，使本論文得以完成；感謝實驗室學長的指導，在遭遇難題時提供了相當大的幫助；感謝實驗室的同學及學弟與我相互討論切磋，讓我獲益良多，也得到了多方面的思考方向。

另外也感謝我的父母，全心全力地支持我完成我的學業；感謝我所有的朋友，讓我在做論文中過程中產生的壓力得以舒解。

Table of Content

List of Figures

1. Introduction

In the Internet, TCP[1] is the most widely used protocol and hence dominate the Internet. In order to mitigate network congestion, congestion control in include in TCP. It will let sender to deduce his transmission rate when he get the information about network congestion. Conventionally, the information is gotten from whether the packets he transmitted are lost. This can be done bye monitor the transmitted packets and corresponding acknowledgement. If sender does not receive any acknowledgement to transmitted packet for Retransmission Timeout (RTO), he thinks that the packet is lost and retransmits it immediately. At the same time congestion window and slow start threshold id reduced and TCP goes in to slow start state. This is very harmful to TCP. Therefore, the value of RTO plays an important role in TCP. When it is too small, retransmission timer will expire even then packet is just delayed, not lost. The timeout will become unnecessary. When RTO is too large, sender will wait longer to retransmit even the transmitted is really lost. This will cause unnecessary waiting. Both will reduce the performance of TCP. Originally, RTO is estimated from the value of Round Trip Time (RTT) since RTO can be thought as the upper bound of RTT. But in some condition, there will be sudden delay in network and spurious timeout will occur. In this condition, retransmission and other TCP parameter reduction still take place. However, these will all be unnecessary since the transmitted packet is probably delayed and not lost. There are now two kinds of approaches to deal with this problem. One is to set a minimum value of RTO (minRTO), i.e. when RTO estimated from RTT is less than RTO, it is set to minRTO[2][3]. We refer this kind as conservative approach. The other one is to keep the original estimation but store TCP states and parameters before retransmission [4][5][6][7][8][9]. When unnecessary retransmission is detected, TCP states and parameters will be restored. We refer this as aggressive approach. In conservative mode, RTO will become predictable since RTO for some connections is set to minRTO. This weakness can be exploited by attackers for malicious behavior. In this paper we will propose a scheme to adjust RTO in conservative approach and make the RTO unpredictable.

1.1. Background

In this section we will describe some background to facilitate understanding our works. It includes TCP Retransmission Timeout and Spurious Timeout.

1.1.1. TCP Retransmission Timeout (RTO)

TCP, a reliable protocol, maintains connection state between two hosts. It will let sender know whether transmitted packets are received by receiver. This is achieved by positive acknowledgement transmitted by receiver to sender. That is, when sender receives a positive acknowledgement from receiver, he judges that receiver has received the transmitted packet. In contrast, when sender does not receive positive acknowledgement respect to the transmitted packet for a period of time, that packet will be thought lost. This period is referred as RTO. Therefore, in TCP, each time a packet a transmitted, a timer will be set and start for that packet. When the timer expires, lost packet will be retransmitted, and the same, a timer will be set for that packet, but RTO will back off. It is because TCP thinks that lost packet implies network congestion and everyone should slow down to mitigate the congestion. When continuous expiration occurs, it will keep backing off until a predefined upper bound is reached.

On the other hand, when the timer expires, congestion control will be triggered [10][11]. Sender’s congestion window will be reduced to one segment and slow-start threshold will also be reduced. Hence, timer expiration is treated as an important indicator of network congestion. The setting of the value of RTO becomes important consequently. If the value is too large, there will be unnecessary waiting at sender side when transmitted packets are lost in the network. If the value is too small, there will be unnecessary timer expiration even transmitted packets or acknowledgement packets are just on the flight, not lost. TCP performance decreases in both situations.

1.1.2. Spurious Timeout

Briefly speaking, spurious timeout out is unnecessary timeout. It usually happens when there is sudden delay in the network. This will cause retransmission timer to expire before the positive acknowledgement packet is received. Spurious timeout will cause unnecessary retransmission and reduce TCP performance seriously. Some algorithms have been proposed to solve this problem.

In the Internet, there are many cases in which the delay may vary, for example, path result from route flipping [2][12][13]. Dial-up connections may also result delay of many seconds due to link-layer error recovery by a modem [14]. The most obvious one, wireless network, will cause delays because of its inherent property. With the development of network technology and hardware device, many different devices can connect to the Internet through different media and communicate to each other. Hence, the number of wireless device connected to the Internet become larger and larger. Wireless links, however, are much slower than wired ones and wireless hosts may hand off from cell to cell. The network properties may differ from one cell to another. Besides, wireless link has higher packet loss rate and may have his owns link-layer retransmission mechanism. When communicating with wireless host, the connection will not be as stable as with wired host. Especially when host is handing off, the delay will not be avoided easily, even in the near future [15].

Two kinds of approaches have been proposed to deal with this problem. One is to adjust RTO. Originally, RTO is estimated from the estimated value of RTT. Later on, in RFC2988 [3], it is suggested that there should be a minimum value of RTO. This also prevents spurious timeout and is widely deployed today. This is referred as conservative RTO in this paper. The other one is to keep the original estimation and store TCP states and parameters before retransmission. When unnecessary retransmission is detected, TCP states and parameters will be restored. We call this aggressive RTO.

1.2. Contribution

In TCP with conservative RTO, there is a minimum value of RTO (minRTO). When RTO estimated from RTT is less than RTO it is set to minRTO. In this situation, RTO will become predictable and maybe exploited by attackers. Low-Rate TCP Targeted DoS takes advantage of this weakness and make throughput of victim approximate zero. In this paper we will propose a scheme to adjust RTO and make it unpredictable to improve the security of TCP in which conservative RTO is deployed. In our scheme, the damage of attack will be reduced and better throughput will be gained apparently. On the other hand, TCP performance will only be affected slightly in usual condition.

1.3. Synopsis

This paper is organized as follows. The related work of our scheme is given in Chapter 2. In Chapter 3, our scheme is proposed. Then, the experiment, evaluation and analysis are illustrated in Chapter 4. Finally, a conclusion presented in Chapter 5.

2. Related Work

In TCP with conservative RTO, when estimated RTO is small than minimum RTO (minRTO), it is set to minRTO. This means that for connections between two near host or with high bandwidth, RTO would be the same value, i.e. the value of minRTO. This may be a problem because many RTO are the same and predictable. There is already an attack, Low-Rate TCP Targeted DoS, that takes advantage of this weakness. In this chapter, we will describe the operation of conservative RTO and the attack.

2.1. Conservative RTO

The conservative RTO that we refer in the paper is the one proposed in RFC2988 [3]. The setting of RTO is illustrated below.

```
RTO: retransmission timeout
RTT: round-trip time
SRTT: smoothed round-trip time
RTTVAR: round-trip time variation
G: clock granularity (typically ≦ 100ms)

Before a round-trip-time (RTT) measurement for a connection
        RTO = 3 seconds

When the first RTT measurement R is make
        SRTT = R
        RTTVAR = R/2
        RTO = SRTT + max ( G , 4*RTTVAR)

When a subsequent RTT R’ is made
       RTTVAR = ( 1 - β ) * RTTVAR + β * |SRTT – R’ |
       SRTT = ( 1 - α ) * SRTT + α * R’
       RTO = SRTT + max ( G , k*RTTVAR)

If RTO < 1 second
       RTO = 1 second
```

Before RTT is measured, RTO is set to 3 second. After RTT is measured, RTO can be thought as RTT plus some network variation. When the network is more unstable, the value of RTTVAR (network variation) will be larger.

2.2. Low-Rate TCP Targeted DoS

Low-Rate TCP Targeted DoS is an attack that takes advantage of predictable RTO. When it is achieved, the throughput of victim will be down to approximate zero. The predictable RTO of victim makes attackers know when the victim will retransmit after packet loss occurs. Attacks then burst packets in the same direction with victim’s retransmission around this time. The burst lasts just a short period of time. Their purpose is to make the retransmission packet of victim to be dropped. When each RTO is predicted by attackers, victim will never transmit packet successfully and throughput will decrease to zero. In Figure 2-1, the basic Low-Rate TCP Targeted DOS attack is shown. The length of the burst is set to around RTT since it will be enough to make whole window of packets to be dropped. The period of the burst is set to minRTO (1 second in RFC2988) since RTO of connections of victim will always be multiple of minRTO if that estimate from RTT is less than minRTO. Thus, whenever victim makes a retransmission, the retransmitted packet will meet the burst packets of attackers. The burst rate is set at least the same with victim link. However, it should be much larger than victim link since the purpose of the attack is to make all retransmitted packets to be dropped by occupying the queue. Larger burst rate will cause better effect, but it will be detected more easily. Besides, legitimate packets of other user that access the same queue of target will help dropping the packets of victim. In this case, the burst rate of attacks may be not so high. This kind of attack is hard to be detected accurately with low false alarm since it just bursts for a short time and this behavior is somewhat like normal user. In the following of this paper, when mention “Low-Rate DoS”, it means “Low-Rate TCP Targeted DoS” described above.

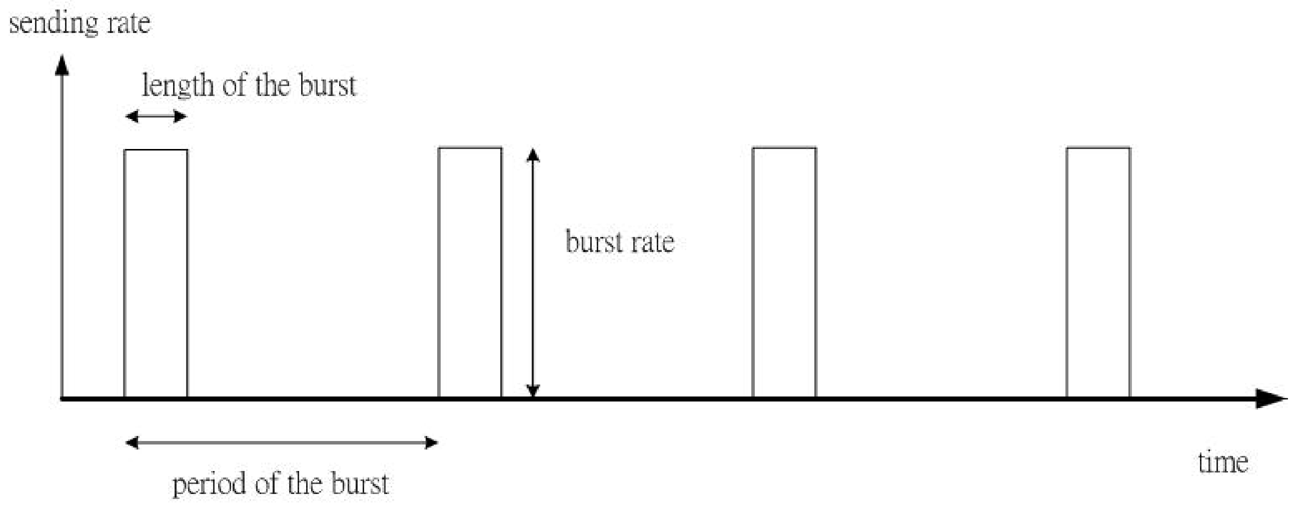

Figure 2-1 Low-Rate TCP Targeted DoS

3. Proposed Scheme

In this chapter, we will describe how we improve the security of TCP in which conservative RTO is deployed. We improve the security by adjusting RTO and make it become unpredictable .There some requirement we should meet:

security: the RTO should be adjusted to be unpredictable.

performance: the performance of TCP before the adjustment should be kept.

We adjust the RTO in four different modes and separate the time to reset RTO and the time to retransmit loss packet. We will define some variables first and then step into out scheme.

3.1. Definitions

Here we will define some variables and they will be used in the following sections.

timeout	the maximum time sender should wait until receipt of the ACK for the transmitted packet
estrto	the RTO estimated according to RTT and historical RTTs before make conservation
minrto	the minimum RTO that is set for transmitted packets
backoff	the backoff multiplier of RTO
t_timeout	the time at which timer expires
t_rtx	the time at which the loss packet is being retransmitted
r	the range with which we use to adjust *t_timeout* or *t_tx* with

With the definition above and RFC2988 we have the relation:

$$timeout = max(minrto,estrto) * backoff$$

Obviously, for connections in smooth network or between two near hosts, the RTT will not be too large and the timeout for transmitted packets will always be minRTO multiplied by an integer. In such case, the timeout for transmitted packets will become predictable, and will be exploited by attackers. Low-Rate TCP-Targeted Denial of Service Attack, for example, is a kind of attack achieved by taking advantage of this weakness. In the following sections, if we do not make any adjustment, we assume that the value of RTO is *minrto* and backoff coefficient is 2. Besides when we mentioned the term “conventional TCP’, it means the TCP version before our modification.

3.2. Random-Delay Scheme

In conventional TCP, *t_timeout* and *t_rtx* can be viewed the same in TCP layer regardless of protocols of link layer and physical layer, i.e. as soon as the timer expires, lost packet is retransmitted immediately. However, in our first two schemes, we randomly delay the retransmission when timer expires. Hence *t_timeout* and *t_rtx* are separated apart. The value *t_timeout* is kept unchanged as conventional TCP and *t_rtx* is randomly chosen so that the actual time lost packet is retransmitted will not be guessed easily.

3.2.1. Fixed Random-Delay Retransmission

In order to avoid the condition in which the retransmission time of lost packet is predictable, we delay the time to retransmit the lost packet. Once the retransmission timer expires at *t_timeout*, the timer is reset but the lost packet will not be transmitted then. After that, a time interval [0,t) is chosen and we pick a time spot randomly in the interval as the period we want to delay the retransmission packet.

Since we separate *t_timeout* and *t_rtx*, how much the timer timed and the elapsed time between when packet was retransmitted and the timer expired differs. The former one, obviously, is *timeout* and we let the latter one *tr* for following use. The value of *tr* can be treat as actual timeout for retransmitted pack. In common case, *tr* should be larger than *minrto* since *minrto* is the first *timeout* and *tr* is the n^{th} actual timeout for retransmitted packet. To meet this requirement, the time interval at which we chose random-delay from is at most [0,*minrto*). In the extreme case, the delayed time for retransmission is *minrto* and *tr* is also *minrto*, i.e. *tr* is half of *timeout*. This would only happen at the first time retransmission timer expires sine the value of the second *timeout* is twice of *minrto* due to backoff operation. If the timer expires again, *tr* will be at least three times of *minrto* because random interval is [0,*minrto*) and next *timeout* is *4minrto*. Therefore, the relation of *r_rtx* and *t_timeout* can be represented as:

$$t\_rtx = t\_timeout + minrto * random(r)$$

The value of *r* is between 0 and 1. The *random()* function will randomly return a real value located in the interval [0,r). Consider Low-Rate DoS attack, assume any packet sent by victim will be dropped when attacker bursts packets. Let ℓ be the lasted burst time, *n* be times of continuous timer expiration, and *R* be (*minrto * random(r))*, the probability that every retransmitted packet is dropped is:

$$\left[\min\left(1, \frac{\ell}{R}\right)\right]^n$$

When R is larger than ℓ, the attack will not always make the retransmission packets of victim dropped. In contrast, if ℓ is larger than R, retransmission packets will be always dropped. Of course attackers can enlarge ℓ to make the attack powerful, but the attack will not be "Low-Rate" and will be detected with more probability. So how to set the value of *r* is an important issue. We will discuss this in next chapter. Let *minrto* 1 second, this scheme can be illustrate as Figure 3-1.

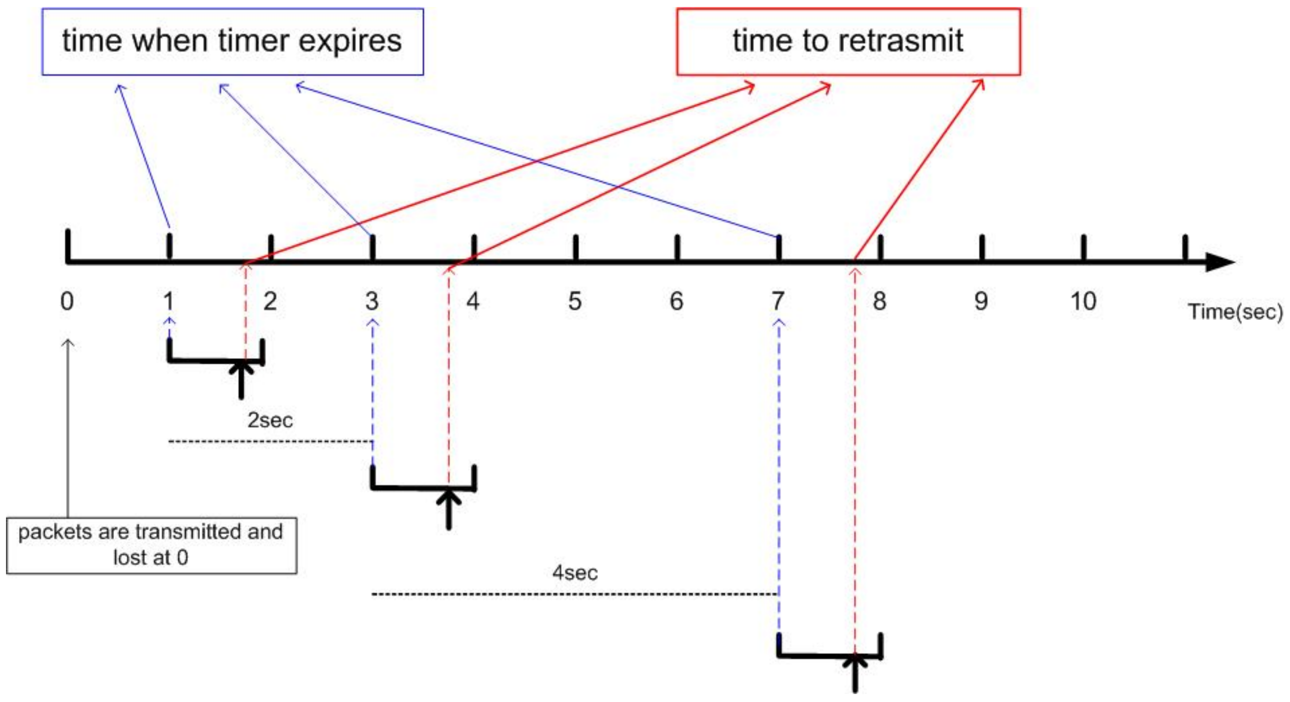

Figure 3-1 fixed random delay scheme

3.2.2. Geometric Random-Delay Retransmission

In the scheme mentioned above, the extreme case only happens at the time when first retransmission timer expires. We want to make the extreme case happen when each retransmission timer expires. This will cause the time interval [0,t) to become large and let the time to retransmit packet more unpredictable. The goal we want go achieve now is to make *tr* be always half of *timeout.* It is reasonable to do this since the nth *timeout* should be larger than (n-1)th one. And the effect of backoff will be reserved if we let *tr* always be half of *timeout*, i.e. the nth *tr* will be twice of the (n-1)th *tr*. Therefore, the relation of *r_rtx* and *t_timeout* can be represented as:

$$t\_rtx = t\_timeout + 1/2 * minrto * backoff * random(r)$$

The value of *r* is between 0 and 1. The *random()* function will randomly return a real value located in the interval [0,r). Let us onsider Low-Rate DoS attack, assume any packet sent by victim will be dropped when attacker bursts packets. Let ℓ be the lasted burst time, *n* be number of times of continuous timer expiration, and *R* be (*minrto* * *random(r))*, the probability that every retransmitted packet is dropped is at most:

$$\left(\frac{\ell}{R}\right)^n$$

This happens in the extreme case when ℓ is less than *R* and *R* is very closed to *minrto*. Note that when ℓ is larger than *R*(but still less than *minrto*), victim still have chance to retransmit packet successfully. This is because that the delay time for retransmission also have the "backoff" feature. The larger the number of time of continuous timer expiration is, the more probability with which the packet to be retransmitted successfully. Let *minrto* 1 second, this scheme can be illustrate as Figure 3-2.

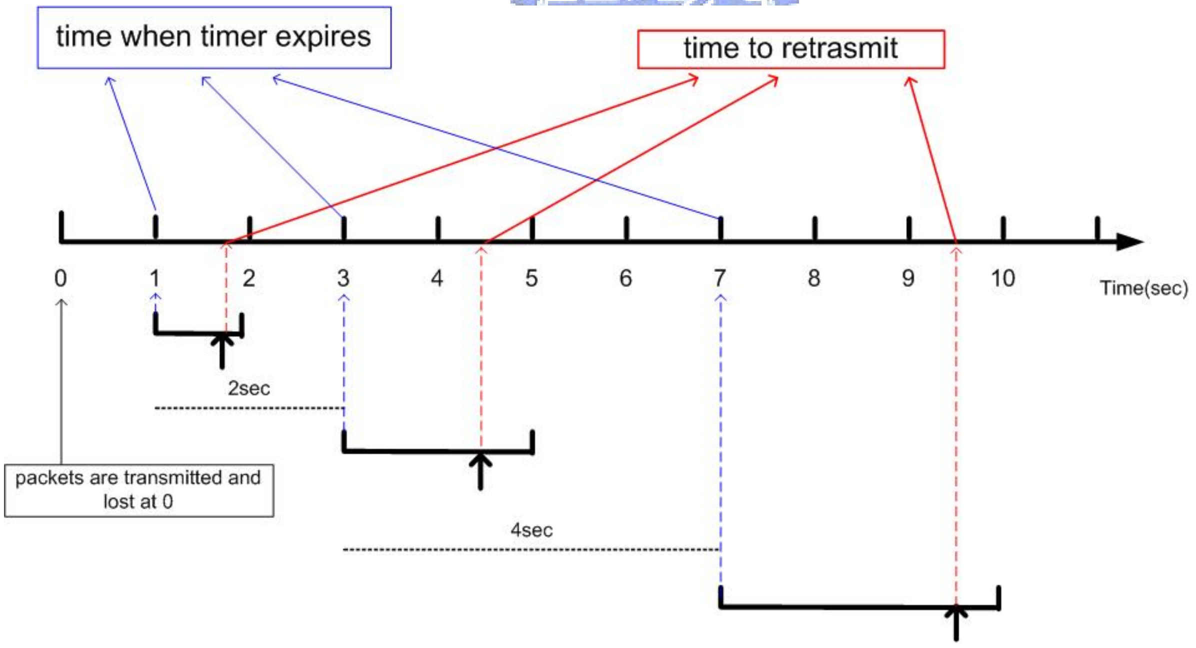

Figure 3-2 geometric random delay scheme

3.3. Random-Timeout Scheme

In random delay scheme, we delay a period randomly to retransmit packets when timer expires. However, the time when the timer expires remain unchanged. And this may be another exploitation for attackers although we have not have any information of such kind of attack. On the other hand, we may need one more timer for each packet. It is used to time the delayed retransmitted packet. In this section, we will introduce an enhanced scheme to improve the deficiency.

3.3.1. Fixed Random-Timeout

Although the random delay scheme make the actual packet retransmission time unpredictable, when the timer expires is stall predictable, and there should another timer for the delayed period. To improve these, we adjust *timeout* and let *t_rtx* the same with *t_timeout*. When the estimated RTO is less than *minrto*, we will not set RTO *minrto*. Instead, we assign a new value to RTO and this value should be unpredictable. Besides, we should keep the backoff property of conventional TCP, i.e. the second *timeout* should be about twice of the first one and the third one should be about four times of the first one. To meet these requirements, we add a random time to conventional *timeout* each time it is computed, and the random time is between 0 and *minrto*. We can represent the adjusted *timeout* as:

$$timeout = minrto * backoff \quad + minrto * random(r)$$

The value of *r* is between 0 and 1. the *random()* function will randomly return a real value located in the interval [0,r). Therefore, the largest range of the first *timeout* will be [*minrto*, *2minrto*), the largest range of the second timeout will be [2minrto,3minrto), and the largest range of the third timeout will be [4minrto,5minrto) … and so on. The backoff property is implicitly kept. Assume the first *t_timeout* and *t_rtx* is *t1*. The second *t_timeout* and *r_rtx* will locate in [*t1+2minrto, t1+3minrto*). Consequently, *t_timeo*ut and *t_rtx* will be unpredictable and *timeout* will keep the backoff property. Let *minrto* 1 second, this scheme can be illustrate as Figure 3-3.

In the random-delay scheme, *t_rtx* is always large then *t_timeout*. Consequently, the total time to wait for timer expiration and delayed retransmission is larger than *minrto*. In the random timeout scheme, since *timeout* is adjusted, it is possible for the *timeout* to be small than *minrto*. This can be represented as:

$$timeout = minrto * backoff - minrto * random(r)$$

In this situation, the value of *r* should be chosen carefully. It should be smaller than what we showed previously. But no matter how small it is, it will make *minrto* meaningless and the RTO will be less conservative. Besides, under the Low-Rate DoS, this will not help much. We will show the detail in next chapter. Therefore, we do not encourage adjusting *timeout* this way.

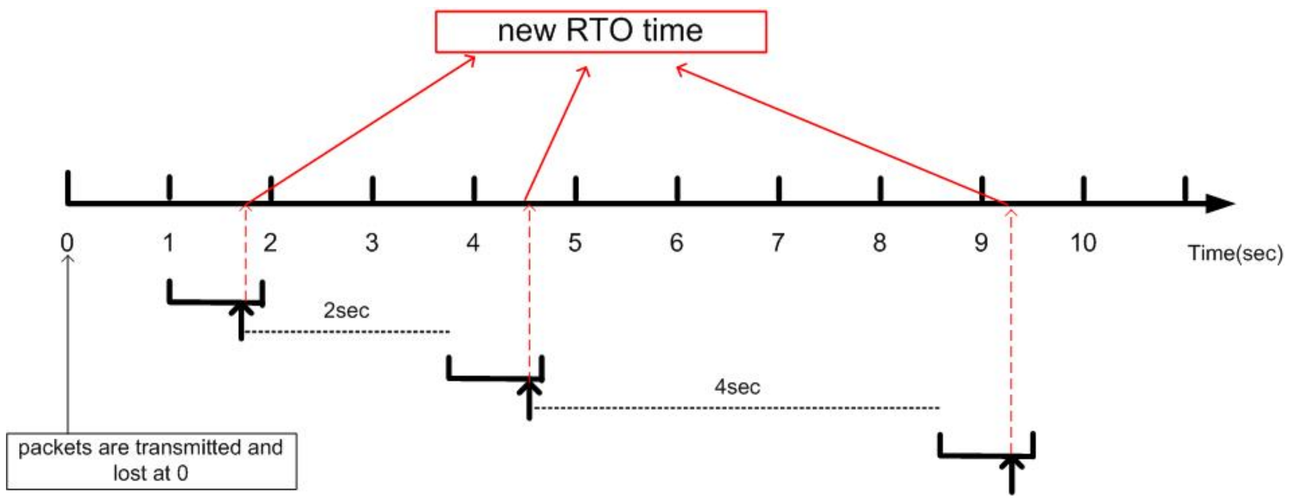

Figure 3-3 fixed random timeout scheme

3.3.2. Geometric Random-Timeout

There is another way to adjust *timeout* to become unpredictable and keep its backoff property. In random-delay scheme, we introduce random delay for retransmission when timer expires, and then the randomly delayed time is from fixedly adjusted to geometrically adjusted. We do similar works in random timeout scheme. That is, each time when timeout is computed, it is increased with a random value, and this value will increase geometrically when continuous timer expiration occurs. It can be represented as:

$$timeout = minrto * backoff + minrto * backoff * random(r)$$

The value of *r* is between 0 and 1. the *random()* function will randomly return a real value located in the interval [0,r). Therefore, the largest range of the first *timeout* will be [*minrto*, *2minrto*), the largest range of the second timeout will be [*2minrto*, *4minrto*), and the largest range of the third timeout will be [*4minrto*, *8minrto*) … and so on. The backoff property is implicitly kept. Assume the first *t_timeout* and *t_rtx* is *t1*. The second *t_timeout* and *r_rtx* will locate in [*t1+2minrto*, *t1+4minrto*). Consequently, *t_timeo*ut and *t_rtx* will be unpredictable and *timeout* will keep the backoff property. Let *minrto* 1 second, this scheme can be illustrate as Figure 3-4.

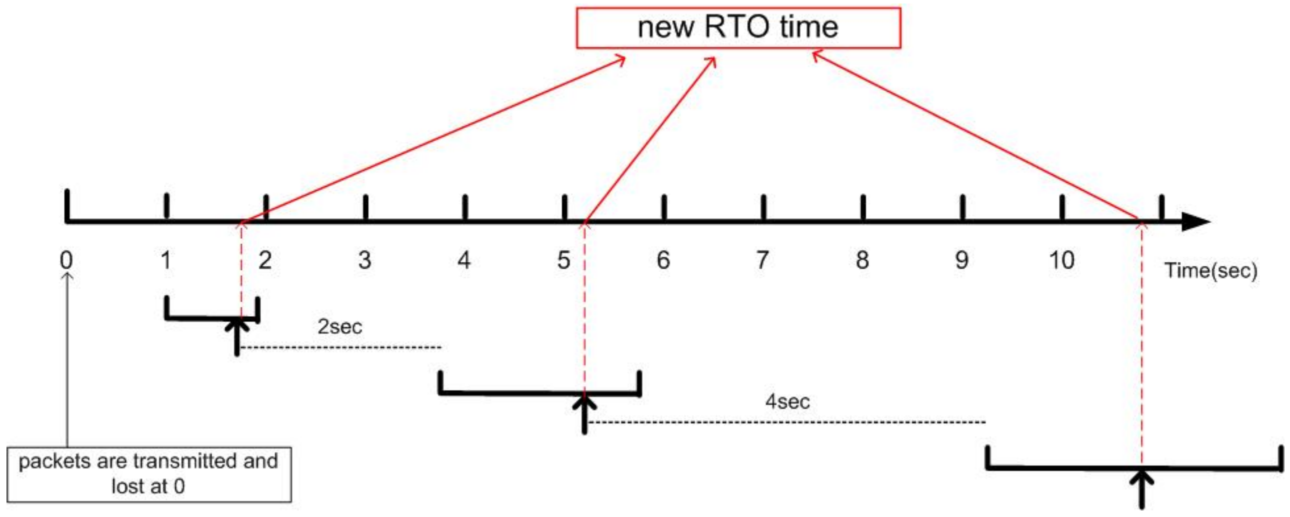

Figure 3-4 geometric random timeout scheme

4. Analysis

In this chapter, we will describe in detail how the proposed scheme improves the TCP conservative RTO. Performance and security issues are included. Besides, we made an experiment with Low-Rate DoS and our proposed scheme. It consists of random delay scheme and random timeout scheme. We also discuss how the value of *r* affects the performance of the scheme against Low-Rate DoS.

4.1. Security and Performance

The TCP conventional RTO suggested in RFC2988 will cause a predictable RTO. In the proposed scheme, no matter random delay scheme or random timeout scheme, the actual time to retransmit packet become less predictable. The security problem has been indeed improved. Up to now, we have only known one kind of attacks which takes advantage of this weakness, Low-Rate TCP-targeted DoS. In conventional TCP, Low-Rate DoS results in approximate zero throughput at victim. But after improvement, our proposed scheme survives this attack. This is because the retransmitted packet in our scheme will not always meet burst packets of attacks.

The version of TCP most popularly deployed today include fast retransmission feature. Recall that when TCP receiver receives an unexpected packet, he will send back an ACK to sender. The purpose of the ACK is to ask sender to retransmit the expected packet because that packet was probably lost. In TCP versions with fast retransmission feature, such as Tahoe, Reno and Vegas, sender will retransmit packet once he receive three duplicate packets. The time required for sender to receive three duplicate packets is often less than RTO at first time packet loss occurs, especially for conservative RTO. This is aim to not let TCP fall into timeout state, as described previously, this will be harmful to TCP performance. Hence, in usual case, TCP retransmission timer will not expire very often. In our proposed scheme, we make the first *timeout* from *minrto* to at most *2minrot*. This will make RTO more conservative but the delay retransmission time will increase. However, under the protection of fast retransmission, we believe this will affect average TCP performance less. On the other hand, attacker can cause network congestion or packet loss to let the retransmission timers expire. In this situation, a following unpredictable timeout will avoid the retransmitted packet being dropped in the trap arranged by attackers.

4.2. Experiment

In this section, we will describe how we make experiment. The result will also be presented accompanied by some analysis and discussion.

4.2.1. Environment

Our experiment is made with NS simulator. The topology is show in Figure 4-1. Node A is a FTP server and node D is a FTP client. Node B is an attacker who uses Low-Rate DoS attack to reduce throughput of A. And node D can be viewed as a router. The bandwidth of link AC is 2Mbps, BC is 10Mbps, CD is 1.5Mbps. We set the bandwidth of link BC larger than AC because we want to let the attacker have more power to achieve his attack. Propagation delay on each links is 10ms. The size of queue on node C for link CD is 10 slots. The whole experiment lasts for 50 seconds. In the beginning, FTP connection is setup between node A and node D, D retrieve files from A. After 1 second, Low-Rate DoS is started from node B. Attack is started late because we want to let TCP window of FTP connection grow to a suitable degree.

In the experiment, we deployed our schemes in the FTP connection and *minrto* is set 1 second. We also adjust the burst duration of Low-Rate DoS for each of the deployed scheme. Once a value is changed, one hundred experiments are made continuously. We take the average as our result.

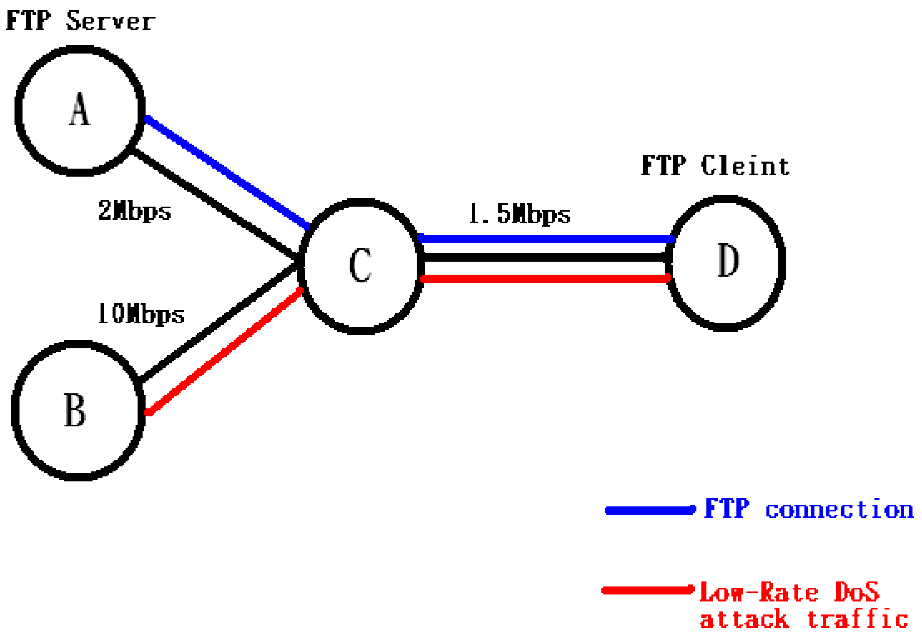

Figure 4-1 experiment topology

4.2.2. Experiment Result

Before the experiment result is showed, we give some analysis of our proposed scheme and Low-Rate DoS. This will be helpful for us to explain the experiment result.

At first, we define some variables, and variable defined in chapter 3 is also used.

B:	the bandwidth of the link connected to victim
ℓ :	burst duration of Low-Rate DoS
T:	burst period of Low-Rate DoS

Consider our scheme and Low-Rate DoS, each time t_rtx dose no conflict with the burst duration of attack, the connection will last at most (*minrto* - ℓ). After that, the connection will be interrupted by next burst packets of attack and all packets transmitted will be dropped. This will cause the TCP to go into timeout state and victim will keep silent and wait for at least *minrto* to retransmit lost packets. Hence, we assume ℓ is always less than *minrto*, the best throughput victim can achieve become

$$\frac{B}{2} - \frac{\ell B}{minrto}$$

Half of the total time is occupied by waiting, and in the rest time, burst duration must be eliminated. To achieve the best throughput, the right time to retransmit packet is at when the busrt just finishes. We name this time point as “sweet point”. The TCP slow-start is not taken into consideration, so the value of the throughput should be decrease some. The case we described above is that T equals to *minrto* and *timeout* ≧ *minrto*. However, there is another case. In [16], it described that when minRTO of victim is uniformly distributed in range [a, b], null time scales will shift to b. It means that when *T* is set to b, throughput of victim will be approximate zero. Actually, the throughput will not be zero since some packets can be retransmitted successfully before the burst occurs. But this will be make throughput of victim to be minimum compared to other attacks with different values of *T*. The best throughput victim can reach become:

$$\frac{T-\ell}{T}B$$

When the value of T is large, better throughput will be gained. This implies that the larger the range [a, b] is the better throughput there will be.

The result of fixed random-delay scheme is illustrated in Figure 4-2. The x-axis represents the burst duration of Low-Rate DoS and y-axis represent the average throughput of the victim. Each curve in the figure represents a value of *r*. The curve with lowest throughput is what *r*=0 presents. Note that the least throughput is not zero because we have unattacked connection for 1 second in the beginning. But we will call this throughput zero regardless of that it is not really zero. When the burst duration is short, it seems that the attack do not affect the victim much. This is because that the queue in router buffs the burst packets. While the throughput of conventional TCP fell into zero, our proposed scheme survived. When burst period is short, curves with small value of *r* will get larger throughput because they retransmit at the sweet point with more probability. However, the throughput of them will decrease when burst duration increase. Smaller the value of r is, more rapidly the throughput decreases. It is because that less delay time will not help TCP to escape the burst duration which is long. The best-performed curve is *r*=0.5 since it perform averagely at head and best in half-tail.

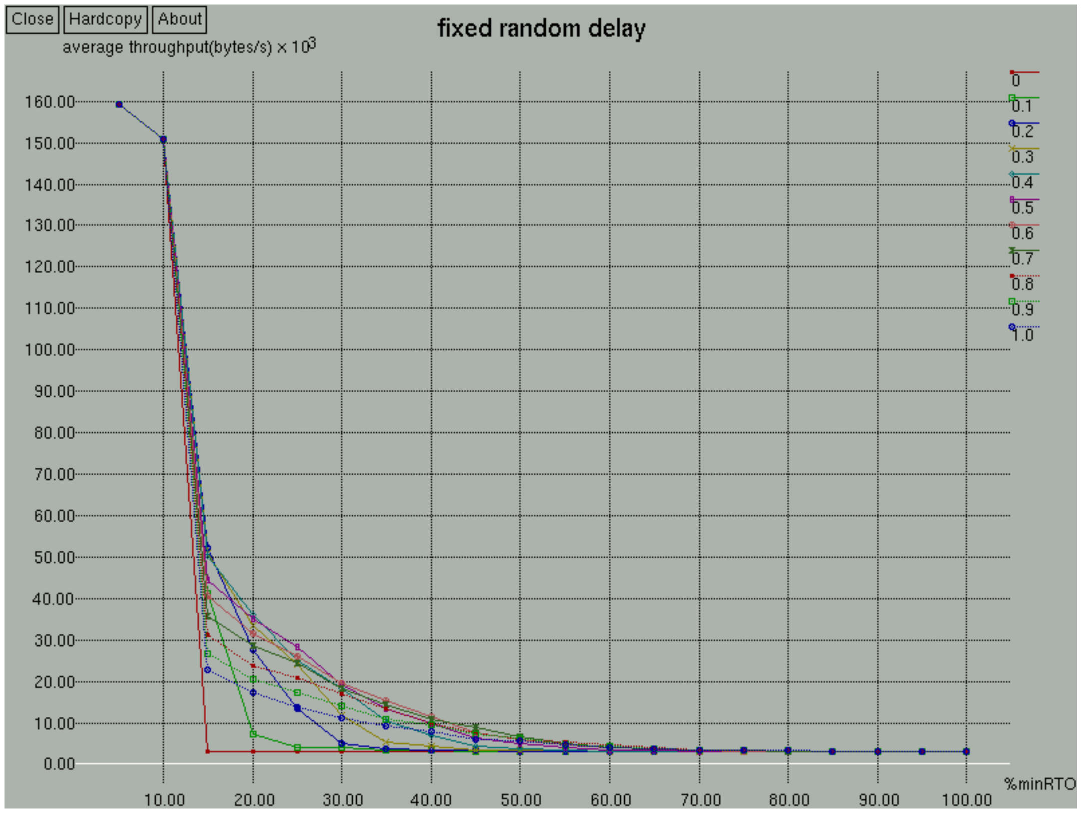

Figure 4-2 result of fixed-delay scheme

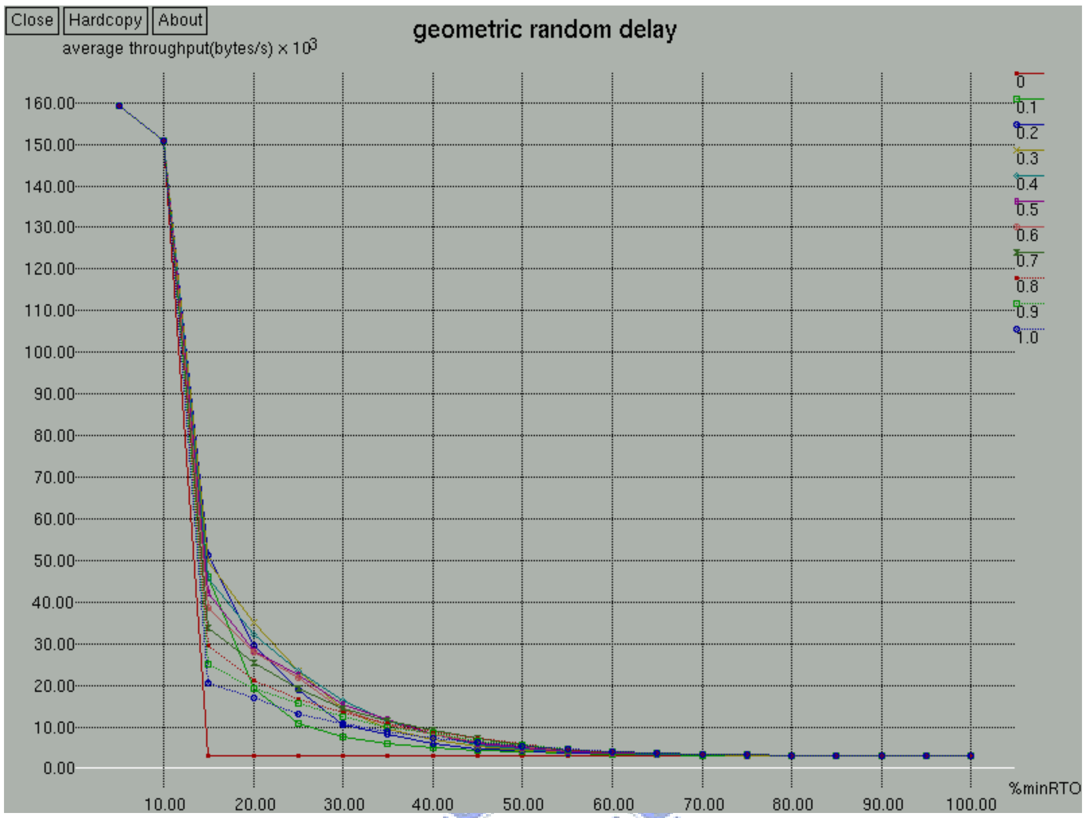

Figure 4-3 result of geometric-delay scheme

The result of geometric random-delay scheme is shown in Figure 4-3. It is similar with fixed random-delay scheme. When the value of *r* is small, the throughput also decreases more rapidly than that with large values of *r*. However, the throughput with small value of *r* does not decrease so rapidly in geometric delay scheme. This is because that TCP will probably escape from the trap in geometric scheme after several continuous timer expirations since the delayed time grows geometrically. When the value of *r* is around 0.4, TCP has larger average throughput than others.

We can observe that, in the geometric random delay scheme, when the value of *r* is larger than 0.4, the average throughput is less than that in fixed random delay scheme. This is because geometrically increased delay will probably decrease the probability of *t_rtx* locating at sweet point.

Figure 4-4 shows the result of fixed random timeout scheme. It is similar with Figure 4-2, but produce larger throughput, especial when *r* is small. It is because that fixed random timeout scheme has implicit ability to “shift away” from the burst pried. Note that *t_timeout* does not locate at *minrto* or multiple of *minrto* on time axis (assume packets are send and lost at 0). When *t_rtx* is conflict with burst period at t_c from the beginning of burst period, the next *t_rtx* will locate at the time t_c plus a random time. It will perform well as *r* is between 0.3 and 0.5. Actually, this mode is the best one among our proposed schemes in the experiment.

The result of geometric random timeout is displayed in Figure 4-5. Averagely, it does not perform better than fixed random timeouts. The reason is the same as in random delay scheme. Geometrically increased timeout will probably decrease the probability with which *t-rtx* hits sweet point. But when the value of *r* is small, this mode will help TCP to shift away from burst period more quickly due to geometrically increased *timeout*. Thus, the throughput will be larger than that in fixed random timeout scheme when the value of *r* is small. This mode will perform better as *r* is between 0.3 and 0.4

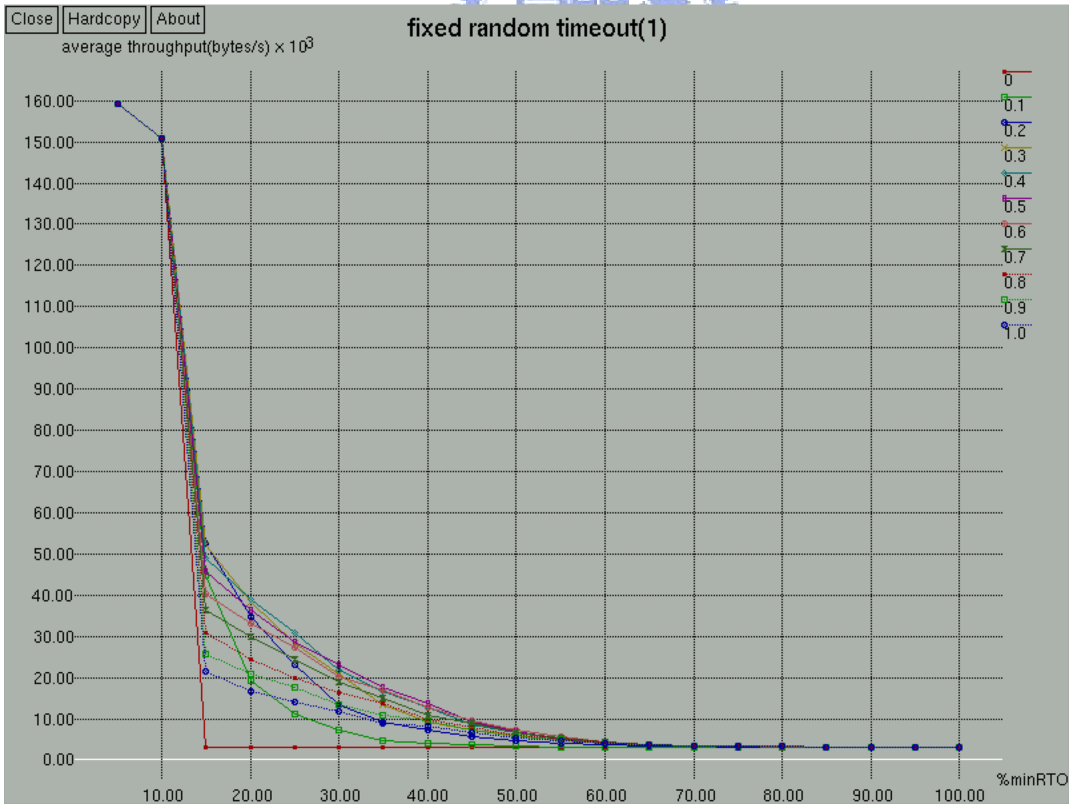

Figure 4-4 result of fixed random timeout scheme(1)

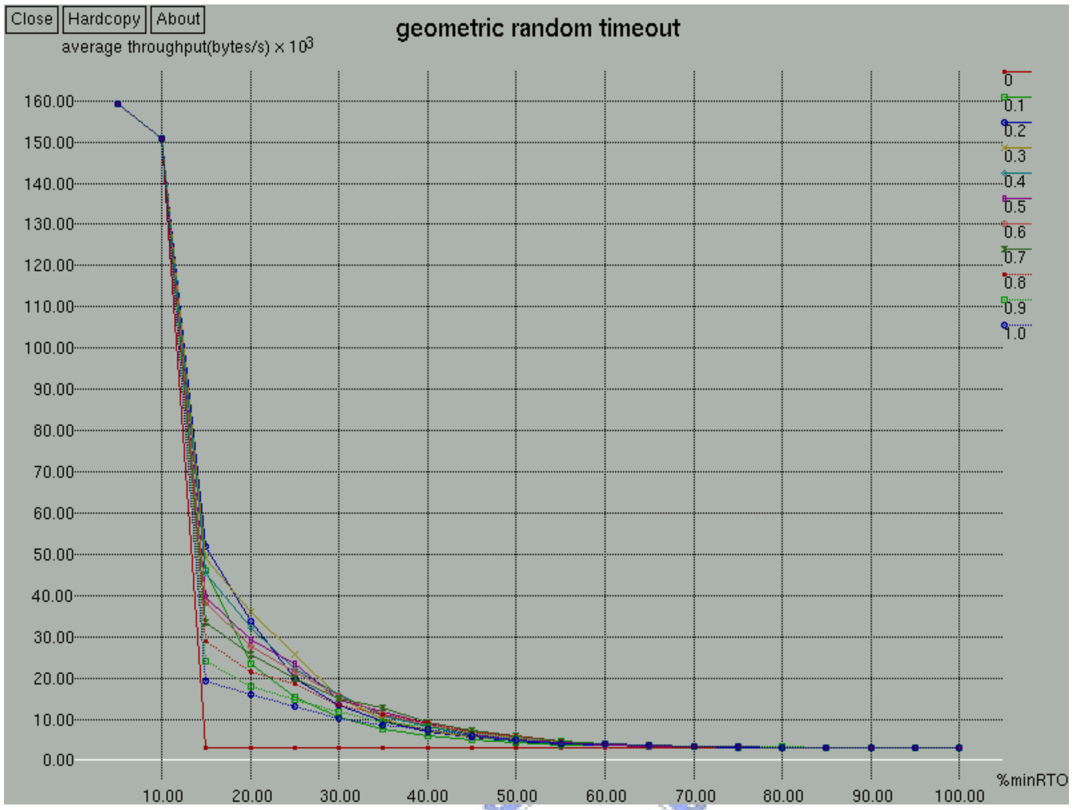

Figure 4-5 result geometrical random timeout scheme

The result of anther mode where *timeout* is probably less than *minrto* is shown in Figure 4-6. The poor performance is due to not only less probability for *t-rtx* to hit sweet point but also it has more chance for TCP to be conflict with burst duration. When continuous timer expiration occurs, it "shifts away" from the burst very slowly. The left-half of the random interval which originally doest not conflict with burst period will have more chance to conflict after repeated expiration.

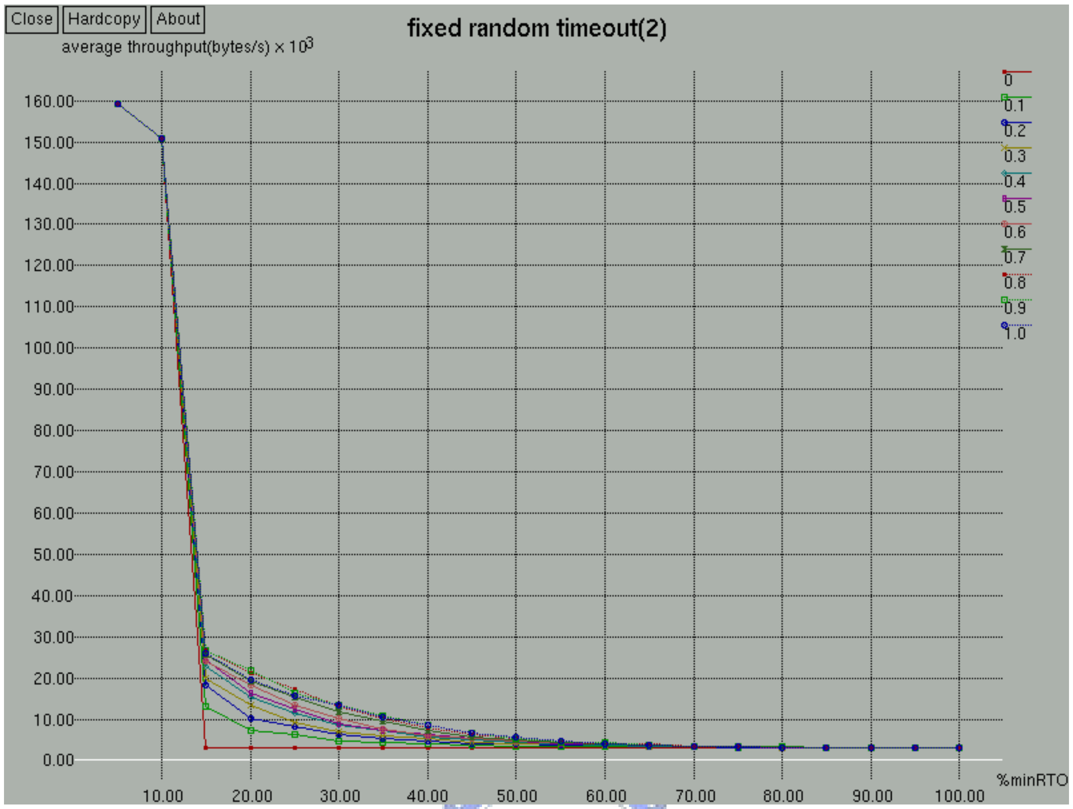

Figure 4-6 result of fix random timeout scheme(2)

4.3. Discussion

Apparently, in all of the proposed schemes, TCP can escape from the trap of attackers. The throughput also increases instead of being zero. Now let us consider the ideally most throughput in the experiment. Assume ℓ is 0.15, the most throughput is $\frac{1.5Mbps}{2} - \frac{0.15 \bullet 1.5Mbps}{1} = 0.525Mbps$. It equals to 67.2Kbytes/s. When ℓ is 0.3, the most throughput is 38.4Kbytes/s and when ℓ is 0.45 that is 9.6Kbytes/s. From the result of the experiment, we can observe that our schemes not only escape form the trap but also get good through under Low-Rate DoS attack. No matter in which scheme we proposed, it is clear that the value of *r* affect the scheme in similar way. As the value of *r* is small, TCP will perform better when burst period is short, but the throughput will decrease rapidly when bust period becomes longer. As the value of *r* is about 0.3 ~ 0.5, the performance is the best. As the value of *r* is larger, the performance will become worse. But actually, it will perform better when burst period is large. However, the influence is small in this situation and the attack is not “low rate” any more. Among all of our proposed schemes, we think fixed random timeout scheme is the best one since it can “shift away” suitably when timer expiration occurs.

Consider that why conservative RTO may fall into the trap of Low-Rate DoS and aggressive RTO won’t. One reason is that conservative RTO gives attackers opportunity that they can burst just for a short time and rest for a longer time. The other reason is that conservative RTO is predictable and aggressive RTO is not because it is adaptively set according to the value of RTT and RTT will not always the same. Although we want to keep conservative property of RTO, we can still make RTO adaptive to RTT. It would be:

$$timeout = r * rstrto + minrto * backoff$$

or

$$timeout = r * rstrto * backoff + minrto * backoff$$

This seems like a subset of random timeout scheme addressed in chapter 3. It does not involve randomness but still has some random property from the RTT (or network status). Hence, the routine of generating random number can be skipped compared to previous described schemes. But the RTO in this situation will be more predictable than random scheme sine RTT follows network state and not really random.

Low-Rate DoS is the only one attack we found that take advantage of the weakness of predictable RTO. In our experiment, we observe that it is not easy to let TCP go into timeout state.

One reason is that queues in routers can buff the burst. Another reason is that it is hard to make all packets in a window to be dropped with just short burst duration, especially when then the link bandwidth of attack and victim is similar. In TCP with fast retransmission feature, if one packet is dropped first and three packets in window are successfully received later by receiver, three ACKs that request for the same lost packet will be transmitted. When sender receives the ACKs(about a RTT after the beginning of burst), the burst is about stopped, and the retransmission will probably successful. This is why we enlarge the bandwidth of attack link in our experiment. However, our purpose scheme is to provide a more secure protocol by make RTO unpredictable and the threat of Low-Rate of is what we should avoid.

5. Conclusion

In this thesis, we have addressed that spurious timeout is very harmful to TCP. Two kinds of approaches are proposed to mitigate this problem, conservative approach and aggressive approach. In conservative approach, a conservative RTO, minRTO, is used. When RTO estimate form RTT is less than minRTO, it is set to minRTO. However, this makes RTO become predictable and may be exploit by attackers. Low-Rate TCP Targeted DoS is such kind of attack. It burst for just a short time periodically to make every retransmission packet of victim to be dropped and hence the throughput of victim will decrease to approximate zero. We also propose four schemes to deal with this problem. We make RTO unpredictable and keep conservative property. The analysis and experiment also showed that our scheme survive the attack. Besides, in usual condition, our schemes affect TCP performance only slightly. In the experiment, it is illustrated that how the random parameter *r* affects our schemes. To gain the best throughput under Low-Rate DoS, we explained that it can achieve by retransmitting at "sweet point".

Reference

[1] J. Postel, “Transmission Control Protocol,” RFC793, September 1981

[2] M. Allman, V. Paxson, “On Estimating End-to-End Network Path Properties,” ACM SigCOMM, September 1999

[3] V. Paxson and M. Allman, “Computing TCP’s retransmission timer,” November 2000. Internet RFC 2988

[4] R. Ludwing, and R. H. Katz, “The Eifel Algorithm: Making TCP Robust Against Spurious Retrasmissions, ACM Computer Communication Review,” vol. 30(1), January 2000.

[5] R. Ludwig, K. Sklower, “The Eifel Retrasmission Timer,” ACM Computer Communication Review, vol. 30(1), July 2000.

[6] P. Sarolahti, M. Kojo, and K. Raatikainen, “F-RTO: An Enhanced Recovery Algorithm for TCP Retransmission Timeouts,” ACM Computer Communication Review, vol. 33, April 2003

[7] A. Gurtov and R. Ludwig, “Response to Spurious Retransmission Timeouts,” in Proceedings of IEEE Infocom 2003

[8] P. Sarolahti, “Congestion Control on Spurious TCP Retransmission Timeouts,” In Proceedings of IEEE Globecom 2003,December 2003.

[9] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky, “An Extension to the Selective Acknowledgment (SACK) Option for TCP,” RFC 2883, July 2000

[10] M. Allman, V. Paxson, “W. Stevens, TCP Congestion Control,” RFC2581, April 1999

[11] V. Jacobson, M. J. Karels, “Congestion Avoidance and Control, ACM SIGCOMM,” August 1988

[12] M. Allman, “A Web Server’s View of the Transport Layer,” ACM Computer Communication Review, vol. 30(5), October 2000

[13] J. C. Bolot, “Characterizing End-to-End Packet Delay and Loss in the Internet,” Journal of High Speed Networks, vol. 2(2), September 1993

[14] D. Loguinov and H. Radha, “Measurement Study of Low-bitrate Internet Video Streaming,” In Proceedings of ACM SIGCOMM Internet Measurement Workshop, November 2001

[15] J. J. Wang, R. H. Katz, J. Giese, “Policy-Enabled Handoffs Across Heterogeneous Wireless Networks,” In Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, February 1999

[16] A. Kuzmanovic and E. W. Knightly, “Low-Rate TCP-Targeted Denial of Service Attacks (The Shrew vs. the Mice and Elephants),” In Proceedings of ACM DIGCOMM 2003, Karlsruhe, Germany, Aug. 2003