

# Practical Study of a Defense Against Low-Rate TCP-Targeted DoS Attack

Petros Efstathopoulos  
Symantec Research Labs, Culver City, CA, USA  
Petros\_Efstathopoulos@symantec.com

## Abstract

*It has been proven in theory and through simulations [3, 9] that a low-rate TCP-targeted Denial-of-Service (DoS) attack is possible by exploiting the retransmission timeout (RTO) mechanism of TCP. In contrast to most DoS attacks, this exploit requires periodic, low average volume traffic in order to throttle TCP throughput. Consequently this attack is hard to detect and prevent, since most DoS detection systems are triggered by high-rate traffic.*

*For the attack to be successful, the attacker must inject a short burst of traffic, capable of filling up the bottleneck buffers, right before the expiration of the sender's RTO. This forces the sender's TCP connections to timeout with very low throughput. The effectiveness of the attack depends on the attacker's synchronization with the victim's RTO. Certain commercial systems follow the guidelines of RFC-2988 [4] (suggesting a minimum RTO of 1 sec), making this synchronization is far from impossible, while popular operating systems using lower minRTO values (e.g. Linux) are still vulnerable to an attacker using a low latency network.*

*RTO randomization was proposed by [9] as a defense against this attack, since it prevents the attacker from synchronizing attack traffic with RTO expiration intervals. In this paper, we study the results of the attack on a real system (Linux), and evaluate the effectiveness the of RTO randomization in defending against low-rate TCP targeted DoS attacks, showing that the method can prevent a TCP flow from being throttled from attack traffic.*

## 1 Introduction

Denial-of-Service (DoS) attacks aim to prevent legitimate users from making use of a service, by claiming enough of the service's resources to render it unavailable. In doing so, DoS attacks may aggressively consume renewable resources (such as CPU cycles or network bandwidth) thereby launching a "busy attack", or try to allocate and hold large portions of non-renewable, limited resources (e.g. memory, disk or buffer space) thus bringing down the system due to resource starvation through this "claim-and-

hold attack". In both of these cases, the DoS is performed by applying aggressive tactics, that obviously aim at limiting the victims access to the service, therefore making the reasons for the lack of service easier to observe and identify. The irregularities caused to system behavior and network traffic by many such DoS attacks, makes them easier to detect and defend against.

TCP's retransmission timeout (RTO) mechanism is intended to deal with cases of severe congestion and multiple losses. If a sender fails to receive an ACK after the timeout period, it reduces its congestion window to one and retransmits the packet.

Following the recommendation made by [4], some systems implement a minimum RTO (minRTO) of 1 sec. Values for minRTO greater than or equal to 1 sec make these systems vulnerable to a low-rate TCP-targeted DoS attack, like the one presented by [3]. This attack could be characterized as "asymmetric", since it uses limited resources to throttle the throughput of the victim's TCP connections. By doing so, this attack produces the negative effects of a typical DoS attack, except it targets only TCP flows and can often elude detection since it sends traffic at a relatively low average rate.

A defense against this type of DoS attack has been proposed by [9] involves the randomization of the minRTO. Instead of enforcing a fixed minRTO of 1 sec, the proposed defense chooses a random minRTO value between 1 and 1.2 uniformly. Simulation has shown that although this solution does not eliminate the vulnerability, it is able to achieve significantly better throughput when under attack, by making it harder for the attacker to synchronize with the victim and completely throttle TCP connection throughput. Although simulation results are promising, it is unclear how effective it can be when used in real systems that are vulnerable to TCP-targeted low-rate DoS attack.

Nowadays, the most popular systems implement TCP without using randomized minRTO values. Additionally, many systems adhere to the 1 sec minRTO value proposed by [4], while some systems use even higher values (e.g. some versions of OpenBSD and NetBSD use 1 sec, Solaris 8 uses 4sec while some versions of Microsoft Win-

dows probably use values of at least 1 sec). This paper investigates the effectiveness of the attack in Linux, for both its standard minRTO value and the 1 sec minRTO. Furthermore, the paper presents the implementation of the “randomized RTO” defense in the Linux kernel and study its effectiveness in a real system.

The paper is organized as follows: Section 2 briefly presents related work, while Section 3 gives an overview of TCP’s timeout mechanism and insight to the vulnerability in question. Section 4 presents the low-rate TCP-targeted attack in more detail, and Section 5 describes the proposed solution and its implementation in the Linux kernel. In Section 6 we present the experimental results.

## 2 Related Work

The low-rate TCP-targeted attack investigated has been identified [3] and analyzed, showing in theory that it can throttle the throughput of a TCP connection. Router based methods and randomization where proposed by [3] as a defense.

The effectiveness of the attack and RTO randomization where studied in [9]. Simulation, showed that the attack can be very effective and randomization significantly improves, but can not solve the problem entirely. This study attempts to test the effectiveness of the attack on a real system, and also implement and measure the effectiveness of RTO randomization.

Recent studies [5] claim that the 1 sec (or higher) minRTO proposed by [4] is overly conservative for modern networks, since modern operating systems’ timer granularity and next-generation network design cancel the reasons that lead to the choice of such a high value. In fact, the 1 sec minRTO, is shown to cause performance degradation, especially for certain classes of wireless users.

Low-rate TCP-targeted attacks have been shown to be effective and hard to detect and defend against not only in the setting examined by this study, but also at the router level. It has been shown by [10] that routers with default settings are also vulnerable to such attacks, leading to significant BGP performance problems, such as session resets and increased convergence delays.

## 3 TCP Timeout Mechanism

TCP congestion control is essential to TCP’s performance and operates in two timescales depending on the degree of congestion the link is experiencing. On smaller timescales of round trip times (*RTT*) TCP performs *additive-increase multiplicative-decrease (AIMD)* control, aiming at forcing each flow to transmit at the fair rate of its bottleneck link. At times of severe congestion in which multiple losses occur, TCP operates on the longer timescales of

*retransmission time out (RTO)*, which the low-rate DoS attack tries to exploit.

The TCP congestion control mechanism uses the notion of *congestion window (cwnd)*. Each TCP sender uses the congestion window to calculate the transmission window based on the feedback it gets from the network. This mechanism helps avoid congestion since both the receiver’s capabilities and the networks characteristics are taken into account by the sender.

To address temporary congestion, when TCP receives three duplicate ACKs for a packet, it uses the *fast-retransmit* mechanism. In cases of heavy congestion though, when three duplicate ACKs do not reach the sender, the timeout mechanism is triggered, signifying that a packet has not been acknowledged although a specific period of time—the RTO—has elapsed. The calculation of the RTO according to [4] is based on the following formula:

$$RTO = \max(SRTT + 4 * RTTVAR, minRTO)$$

In this equation, *RTTVAR* is the variation of the round-trip time (*RTT*) and *SRTT* is the smoothened round-trip time, based on recent measurements.

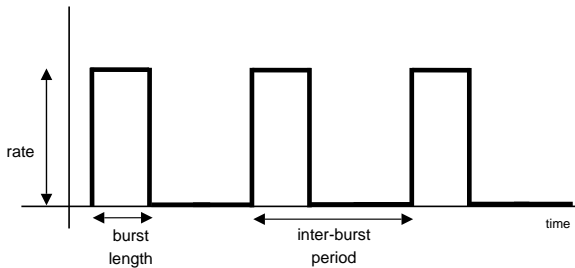
Under heavy congestion, TCP reduces its congestion window size to 1 segment and the the value of RTO is set to its minimum (minRTO). The recommended minimum value for the minRTO is 1 sec, as proposed by RFC-2988 [4]. If the RTO expires and the packet is lost again the exponential back-off continues and the sender doubles the value of RTO (2sec) and retransmits the packet. The sender then waits for the 2sec RTO to expire and if the packet still hasn’t been transmitted successfully the exponential back-off continues (RTO is set to 4sec and so on). This mechanism was chosen for dealing with cases of heavy congestion since it is the most conservative sender behavior.

## 4 Low-Rate TCP-Targeted Attack

TCP’s congestion control mechanism is well suited for dealing with high congestion, but it has an exploitable flaw: the values of RTO are predefined (and therefore predictable) under heavy congestion. The minimum suggested RTO is 1 sec and, therefore, the possible values during exponential back-off are multiples of 1 sec. This property of the algorithm makes the system vulnerable to attacks that use a short, properly timed high-rate burst of packets to fill the bottleneck buffers, right before the RTO expires.

An attacker that knows the timing of the sender can use a “square wave” attack traffic pattern (high rate, short duration bursts), in order to force the sender to repeatedly enter the retransmission timeout state. Since the bottleneck buffers will be filled with attack traffic, the throughput achieved by the sender will be near-zero. Such an attack traffic pattern—shown in Figure 1—transmits bursts of

packets at a set rate for a set time and then waits out the inter-burst period before bursting again. It does so in the hopes that subsequent bursts will occur just as victim flows begin retransmission. The attack has three properties: a send



**Figure 1. A low-rate attack can be approximated as a square wave.**

rate, a burst length, and an inter-burst period.

The send rate is the rate at which the attacker sends packets into the network. Since the intention of the attacker is to quickly fill the the bottleneck buffers, exactly before the victim attempts to retransmit, this rate must be higher than the bandwidth of the bottleneck link. If the rate is high enough to fill the bottleneck buffers, the victim’s packets will be dropped and this packet loss will cause TCP flows to timeout.

The burst length determines how long the attacker floods packets during each burst. This depends on the send rate, RTT of the flows, and bandwidth of the bottleneck link. If the burst length is too long, then the attacker risks being detected, while a short length may not be adequate to fill the bottleneck buffers (for a given rate). The inter-burst period (IBP) is the time that elapses between two consecutive bursts of attack traffic. Both send rate and burst length need to be chosen carefully by the attacker for the attack to be successful and remain undetected. An excessively high rate or an unnecessarily long burst length may trigger router-based DoS protection mechanisms. It was suggested by [3] that a burst length less than 300 msec would allow the low-rate attacker to go relatively undetected. The experiments show that a burst length of 200 msec is sufficient to produce zero throughput with the proper inter-burst period.

The inter-burst period dictates the frequency at which the attacker transmits bursts of packets. An attacker would want to synchronize with the victim by picking an inter-burst period that corresponds exactly to the victim’s RTO. This way, the attacker can throttle the throughput while transmitting the least amount of data possible to reduce the chances of being detected. Notice that the only network characteristic the attacker needs to know in order to launch the attack is the rate of the bottleneck link.

As discussed in [3], the attacker can further reduce the

chances of being detected by router-based detection mechanisms using a double-rate burst of traffic: first the attacker utilizes a very high rate (higher than the bottleneck link), for a very short period of time, and subsequently the attacker reduces the rate of the burst traffic to the bottleneck link’s rate. The initial, high rate is used to quickly fill the buffers of the bottleneck link and the secondary rate ensures that the buffers remain full for the duration of the burst length. This approach requires a very high rate only for the initial very small period of time and therefore is harder to detect. Without loss of generality, this study considers only the single-rate attack.

## 4.1 Linux TCP Implementation

Linux 2.4 kernels implements congestion control using the TCP New Reno [7] standard. Starting from 2.6.8 the Linux kernel implements Binary Increase Congestion control TCP [8] and versions 2.6.19 and above use CUBIC TCP [6].

What is interesting is the fact that Linux uses a minRTO of 200 msec, which deviates from the 1 sec minRTO standard, making the attack harder. Achieving fine synchronization of a 200 msec IBP square wave with the victim is hard, and filling the bottleneck buffers using such short IBP might require the use of a very high rate—potentially detectable by DoS defense mechanisms.

## 5 Randomized RTO Defense

Many protocols (e.g. link layer protocols such as Ethernet) use randomized timeouts during exponential back-off. Based on the observation that many protocols (e.g. Ethernet) use randomized timeouts during exponential back-off, [3] proposes a defense against low-rate TCP-targeted DoS attacks: instead of using a fixed value for the minRTO (e.g. 1 sec), minRTO is randomized around that value, making it hard for the attacker to synchronize with the RTO expiration intervals, and use attack traffic to flood the bottleneck buffers with a properly timed burst. Simulation results presented by [9] show that randomization of the RTO is indeed a possible solution.

There are three different ways one could choose to randomize minRTO. The most conservative approach, which overestimates the RTO, would be to pick a number in the range  $[t, t+1)$ . Alternatively, values within the ranges  $[t-0.5, t+0.5)$  and  $[0.5*t, 1.5*t)$  could also be used. Simulations and analysis done by [9] show that the range of randomization does not affect the results significantly. For the purposes of this study it is assumed that RTO randomization is done within the  $[t-0.5, t+0.5)$  range.

## 5.1 Implementation

Although Linux is less vulnerable because of its 200 msec minRTO, the vulnerability in the design of the TCP timeout mechanisms is existent and other systems using higher minRTO values (e.g. OpenBSD, Solaris 8 etc) may be susceptible to this attack. The purpose of this study is to test the effectiveness of the attack on real systems and try to reproduce the results of the simulations. This study is using the Linux kernel 2.4.22, and performed the necessary modifications to it<sup>1</sup>

Two versions of the Linux kernel were used: The first version was modified so that the minRTO was bounded to 1 sec in order to test the effectiveness of the attack. This required adjusting the RTO every time it gets calculated from the values of SRTT (smoothened RTT) and RTTVAR. When switching from a 200 msec minRTO to 1 sec, it was necessary to properly initialize RTTVAR and modify the *tcp\_bound\_rto()* Linux function to perform minRTO lower bound checking as well.

In order to measure the effectiveness of randomization in defending against the attack, a second Linux kernel version was created by modifying the “1 sec RTO” kernel so that the value of RTO is randomized within the range  $[t-0.5, t+0.5]$ . Initially, the minRTO is adjusted so that it is not below 1 sec, and afterwards a random number is generated to perform the desired RTO randomization.

## 6 Experiments and Evaluation

Experiments were conducted using the TCP test-bed, shown in Figure 2. The sender (victim) and attacker (supposedly on the same subnet) were connected to the receiver through an intermediate node running DummyNet [1], used to simulate the Internet. The sender and receiver are im-

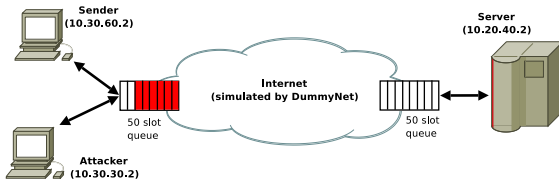


Figure 2. The experimental test-bed.

plemented using the Iperf [2] bandwidth measurement tool (client-server mode). The receiver has its own 50-slot queue, while the sender and the attacker share a 50-slot bottleneck queue, shown in Figure 2, that is filled by the attacker’s periodic bursts of traffic, forcing the sender’s TCP

<sup>1</sup>Note that 2.6 kernels are expected to behave similarly to 2.4 regarding the low-rate TCP-targeted attack, since they also use a 200 msec fixed value for minRTO.

connections to timeout. All hosts are connected to the test-bed through bi-directional 1.5 Mbps links, while the RTT is set to 40ms. A custom attacker was implemented to produce a low-rate attack, generating 3 Mbps of traffic in periodic bursts of specified lengths and inter-burst periods.

For each of the three Linux kernel versions we used (regular Linux kernel, “1 sec minRTO” and “1 sec randomized RTO”) we ran multiple 20 seconds attacks, allowing the sender to transmit for 2 seconds before starting the attacker. The results of previous studies have shown that the important inter-burst period values are at 0.5 and 1 second, because they coincide with the 1 sec RTO and are most vulnerable to the attack.

## 6.1 Results and Evaluation

The first test attempts to evaluate how the regular Linux kernel reacts to a low-rate attack. The attack is expected to be less effective since the attacker is specifically aimed at systems with an RTO of (multiples of) 1 sec, while Linux has an RTO of 200 msec. The throughput graph is presented in Figure 3. Note that the throughput is similar to what is

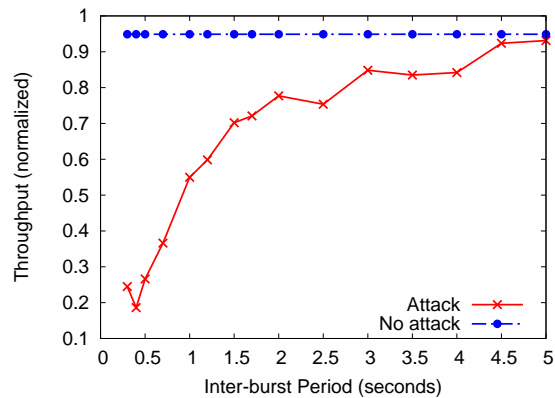
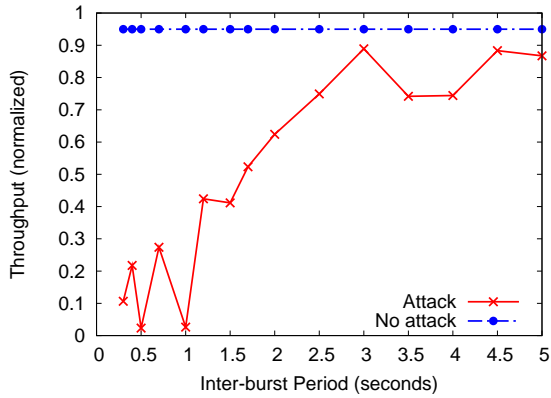


Figure 3. Throughput of the unmodified 2.4.22 kernel with and without the DoS attack.

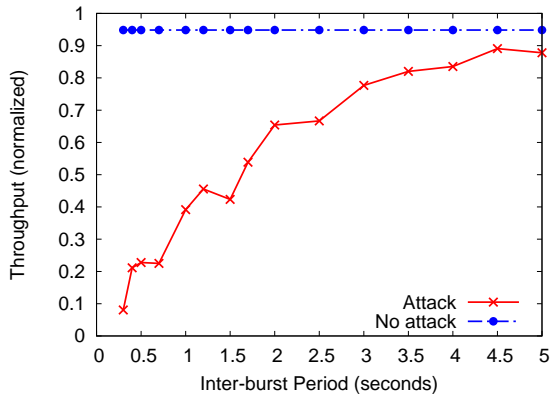
expected for a “1 sec minRTO” vulnerable system, only slightly less effective. As the burst frequency gets closer to 200 msec the throughput drops, but the throughput never drops to zero because the attacker process is geared toward a system with a minRTO of 1 sec. An attack with an inter-burst period of 200 msec could be generated, but it would require a much shorter burst length and much higher burst rate, making it much less stealthy. Additionally, synchronization at such high frequency may be unrealistic in practice.

The next test uses the modified “1 sec minRTO” kernel. As shown in Figure 4, there is a major drop in throughput for inter-burst periods of 0.5 and 1 sec—which is in



**Figure 4. Throughput of the “1 sec minRTO” kernel with and without the DoS attack.**

agreement with the simulation results reported by [3, 9]. In this case the attacker is able to synchronize with the 1 sec minRTO and each retransmission encounters severe packet loss. However, if the inter-burst period is not synchronized with the RTO, higher throughput is achieved: throughput is not affected significantly for inter-burst periods that are not even divisors of 1 sec. Note that the existence of a very small throughput at the 0.5 and 1 sec inter-burst periods is due to the fact that the sender was allowed to transmit for 2 seconds before engaging the attacker. Once the attack began, throughput was throttled to zero for both of these cases.



**Figure 5. Throughput of the “randomized RTO” kernel with and without the DoS attack.**

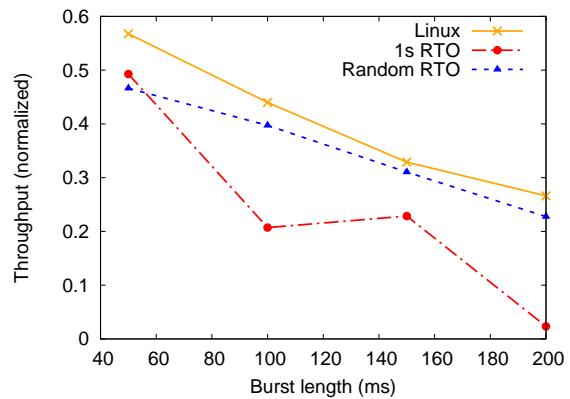
The next step was to test the modified Linux kernel that randomizes the RTO between 0.5 and 1.5sec. This produces RTOs that average to 1 sec yet do not allow a low-rate attacker to predict when the sender will attempt retransmission. The results are shown in Figure 5.

The throughput for the “randomized RTO” kernel is dra-

matically better than the “1 sec minRTO” kernel for attack inter-burst periods of 0.5 and 1 sec. In fact, for the 1 sec inter-burst period, the “randomized RTO” kernel produced a throughput of nearly 40%, which is a significant improvement—considering that the attack is flooding the network 20% of the time. In addition, it is a significant improvement from the “1 sec minRTO” kernel which had zero throughput at this point.

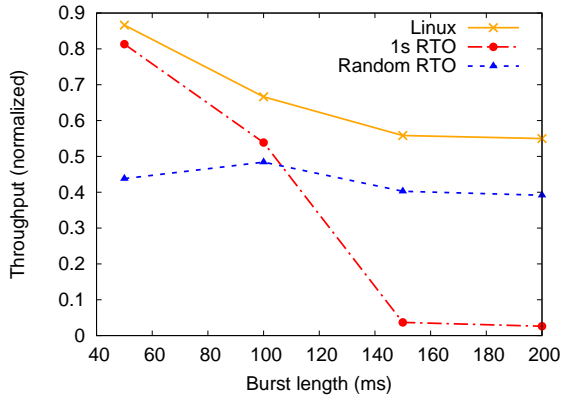
Certain interesting observations can be made about the results of Figures 3,4 and 5. First, notice that the unmodified Linux kernel suffers significant throughput degradation even from the attack targeted to “1 sec minRTO” systems. Furthermore, although the “1 sec minRTO” kernel is completely throttled by the attack, the “randomized RTO” kernel, does not suffer throughput throttling. In order to throttle the throughput of the “randomized RTO” kernel, an attacker would simply resort to a standard DoS attack that transmits almost constantly, making it relatively easy to detect. Therefore, randomization is indeed an effective defense against low-rate DoS attacks (although it can not provide a complete solution).

The results demonstrate the effects of different inter-burst period lengths. It is also interesting to see how the burst length affects the effectiveness of the attack: the attacker must adjust the burst length so as to maximize the effects of the attack (larger burst length) and avoid detection (shorter burst length). The burst rate was kept at 3 Mbps and measured the throughput for inter-burst periods of 0.5 and 1 sec while varying the burst lengths from 50 to 200 msec. The results can be seen in Figures 6, 7 and 8.

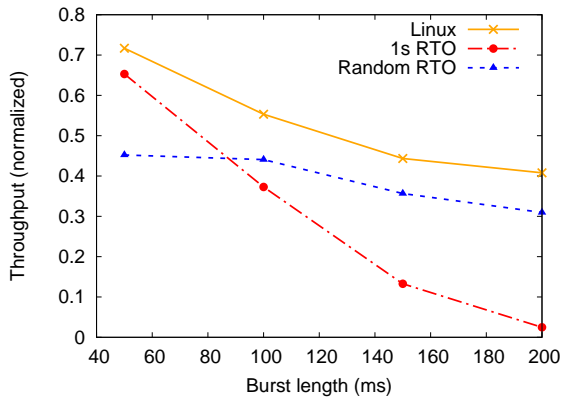


**Figure 6. Throughput of the three kernel versions using a 0.5 sec IBP.**

Measurements show that using a burst length of 200 msec is adequate to throttle throughput in the “1 sec minRTO” kernel. Also, note that the “1 sec minRTO” kernel’s throughput can be throttled with an even shorter burst length (around 150 msec) when the inter-burst period is 1 sec (completely



**Figure 7. Throughput of the three kernel versions using a 1 sec IBP.**



**Figure 8. Averaged throughput results for the 0.5 and 1 sec IBP cases.**

synchronized with the minRTO). In addition, the “randomized RTO” kernel is shown to perform better than the “1 sec minRTO” kernel in these “zero throughput” choking points. In any case, the results show that the threat is higher than initially estimated: the 300 msec burst length proposed by [3] is an over-estimation, since a 200 msec burst is adequate.

## 7 Conclusion

Many protocols randomize retransmission timeout values, but TCP uses multiples of a fixed minimum RTO value (e.g. 1 sec) during its exponential back-off stage. These statically defined, predictable RTO values make the protocol vulnerable to a low-rate attack, that can not be easily detected by current DoS attack defense mechanisms due to its very low average traffic.

This work tests the effectiveness of the attack and the defense against using the standard Linux kernel, the modified “1 sec minRTO” kernel and the “randomized RTO” kernel. Our results confirm simulation results and underline an important threat: systems that currently use minRTO values of 1 sec (or higher) are particularly vulnerable to this attack, since their throughput is throttled when the attacker achieves synchronization with the sender.

Experiments with the “randomized RTO” kernel show randomization can greatly improve performance, by preventing throughput throttling, especially in the case of 1 sec IBP, where the “randomized RTO” kernel achieves 40% of the link’s maximum throughput under attack.

Finally, it is shown that a 200 msec burst length is capable of maximizing the effects of the attack in all cases.

## 7.1 Acknowledgments

The author would like to thank Johnny Tsao for helping with the setup and experiments, as well as Guang Yang and professor Mario Gerla, for their advice.

## References

- [1] DummyNet homepage [Online]. <http://bit.ly/1nMKZR>, [Accessed, 29 May 2009].
- [2] Iperf homepage [Online]. <http://bit.ly/oFTmv>, [Accessed, 29 May 2009].
- [3] A. Kuzmanovic and E. W. Knightly. Low-rate tcp-targeted denial of service attacks and counter strategies. *IEEE/ACM Trans. Netw.*, 14(4):683–696, 2006.
- [4] V. Paxson and M. Allman. Computing TCP’s Retransmission Timer. RFC 2988 (Proposed Standard), Nov. 2000.
- [5] I. Psaras and V. Tsaoussidis. The tcp minimum rto revisited. In *Networking 2007, Atlanta, GA, USA*, May 2007.
- [6] I. Rhee and L. Xu. Cubic: A new tcp-friendly high-speed tcp variants. In *3rd International Workshop on Protocols for Fast Long-Distance Networks*, 2003.
- [7] P. Sarolahti and A. Kuznetsov. Congestion control in linux tcp. In *Proc. 2002 USENIX Annual Technical Conference—FREENIX Track*, pages 49–62, June 2002.
- [8] L. Xu, K. Harfoush, and I. Rhee. Binary increase congestion control (bic) for fast long-distance networks. *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, 4:2514–2524 vol.4, 7-11 March 2004.
- [9] G. Yang, M. Gerla, and M. Y. Sanadidi. Defense against low-rate tcp-targeted denial-of-service attacks. In *ISCC '04: Proceedings of the Ninth International Symposium on Computers and Communications 2004 Volume 2 (ISCC'04)*, pages 345–350, Washington, DC, USA, 2004. IEEE Computer Society.
- [10] Y. Zhang, Z. M. Mao, and J. Wang. Low-rate tcp-targeted dos attacks disrupts internet routing. In *Proc. 14th Annual Network & Distributed System Security Symposium (NDSS)*, Feb 2007.