TCP SYN Flooding Defense*

Livio Ricciulli Patrick Lincoln
{livio,lincoln}@csl.sri.com
Computer Science Laboratory
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

Pankaj Kakkar
pankaj@csl.sri.com
Dept of Computer and Information Science
University of Pennsylvania
200 S 33rd St
Philadelphia, PA 19104

Keywords: SYN flooding, Random drop, Distributed simulation, TCP signaling, TCP denial-of-service

Abstract

The TCP SYN flooding denial-of-service attack pointed out a weakness of then-current Internet protocols. There have since been many proposals to defend against SYN flooding, some requiring significant changes to TCP. Several solutions attempting to resolve the TCP weakness are now generally available. We document these existing solutions and qualitatively compare them. We refine the analysis of the random drop approach and derive a simple and general way to improve its performance. Finally we show, through both analytical and packet-level simulations, the efficacy of the random drop approach in a variety of operating conditions.

1 Introduction

The TCP SYN flooding denial-of-service attack hinders the signaling mechanism that is used to establish TCP connections. HTTP and FTP are examples of two widely used TCP-based protocols that are becoming more important for the exchange of information over the Internet and that are affected by this type of denial-of-service. SYN flooding attacks are performed by the attacker submitting a stream of TCP SYN (connection request) packets to the target system, filling its connection request queue, and thus reducing (or eliminating) the target system's ability to respond to legitimate connection requests. The common TCP timeout for unsuccessful connections is several tens of seconds; thus an attacker can leisurely fill the TCP SYN queue before earlier SYNs time out. The SYN flooding denial-of-service attack, if not dealt with properly, requires very little computation and bandwidth commitment from malicious users. Although SYN flooding requires an attacker to continuously flood a target (otherwise within a few minutes the target will revert to normal operation), it is difficult to trace to the source of SYNs. Thus, SYN flooding remains a viable attack.

Potential loss of revenue caused by preempting reliable TCP communication is enormous, and therefore adequate mechanisms for dealing with SYN flooding should be sought. Current SYN flooding defense mechanisms seem to have greatly mitigated the problem by making it harder for an attacker to negatively affect service. The most popular approach [2] solves the problem by "brute" force. In this approach the TCP "connection pending" data structure is made so large that an average attacker, to be successful, would need to flood connection requests at a rate exceeding reasonable bandwidth capabilities. This solution, although sometimes very practical, requires large amounts of protected kernel memory and may slow down the server response time for looking up connections in the vast "connection pending" data structure. Other less popular techniques use one-way hash functions (cookies) to verify the authenticity of connection requests and therefore eliminate unnecessary memory allocation [3]. Some of these latter techniques can introduce changes in the TCP signaling behavior and are therefore less favored. Firewall approaches actively monitor the TCP signaling traffic to detect possible attacks and inject ad-hoc signaling messages in the network to mitigate the denial-of-service. These approaches are awkward because they introduce additional administrative complexity, may introduce significant delays for legitimate connection establishment, or may expose the system to different, though arguably less severe, kinds of vulnerabilities.

No one mechanism seems to provide an optimal solution, and thus an approach should be constructed by using a combination of techniques. In particular, both the brute force approach and some of the firewall

*This work was supported by the Defense Advanced Research Projects Agency, under contract number DABT63-97-C0040.

mechanisms could be greatly improved if they were to be coupled with some kind of admission control mechanism that can optimize, in a probabilistic sense, their resource utilization.

To this end, we revisit a well-known defense mechanism (random drop) and show its effectiveness. This mechanism, in its simplest form, always accommodates new SYNs and drops at random a connection-pending entry if the connection pending data structure is full. The rationale is that even though legitimate entries may be forced out by spoofed SYNs, one can probabilistically guarantee success. The performance of the random drop scheme has been incorrectly assessed through defective models, and its performance has been underestimated. The main contribution of this paper is to provide a detailed analytical model for characterization of the random drop scheme and to provide the specification of an "early drop" filter to improve its overall performance.

Section 2 of this paper reviews some popular SYN flooding defense mechanisms and introduces our revised version of random drop. Section 3 qualitatively compares the defense mechanism outlined in Section 2 with respect to key properties that characterize their effectiveness. Section 4 details an analytical study of the random drop scheme applied to TCP signaling admission control. Section 5 reports on some high-fidelity simulation of the random drop scheme. We present our conclusions in Section 6.

2 SYN Flooding Defenses

A normal TCP connection is established by a three-step handshake protocol. The initiator (the client) sends a SYN message; the responder (the server) sends a SYNACK message back to the client and waits for the third and final message (ACK) from the client. In a SYN flooding attack the client is spoofed, and the SYNACK message is simply lost in the network. The server, therefore, keeps waiting in vain for an ACK and keeps a queue entry allocated for several seconds.

In a proactive approach, adequate monitoring of the network traffic can detect patterns that indicate a possible attempt to deny service through TCP flooding. Upon detection of the anomaly, the attacker can then be isolated from the network and prosecuted.

In some situations, the inter-domain cooperation necessary to trace an attacker may not be possible. Thus, it is necessary to adopt some local defense mechanisms. In this more reactive approach, the malicious requests are allowed to reach the target server, which can react to anomalous conditions and turn on specific mechanisms aimed at minimizing the impact of the denial-of-service attacks. Several such mechanisms have been proposed, each of which has tradeoffs with respect to effectiveness, robustness, and resource requirements.

Berkeley Cookie Berkeley Software Design Inc.'s solution [2] increased the capacity of storing outstanding connection pending entries, thus requiring that an attacker send a much greater number of SYNs. This mechanism offers limited protection, essentially just increasing the cost of an attack. The parameters chosen by the developers are well suited for today's network technology and, although requiring the server to devote more resources to prevent the SYN attacks, they seem to be adequate to discourage attacks carried out by individuals with limited bandwidth at their disposal. Coordinated attacks or attacks carried out from high-bandwidth links could still circumvent this defense mechanism. Although the Berkeley Cookie scheme does not semantically solve the SYN flooding problem, we believe it is the best current approach for defending large and heavyweight TCP servers (like WWW servers) with large amounts of kernel memory space.

Linux Cookies This idea was proposed initially by Bernstein and Bona [5] and later refined through a discussion in [1]. In this approach, the incoming SYN's sequence number, and the source and destination addresses are combined with a secret number (which is changed at regular intervals) and are run through a one-way hash function. The resulting cookie is used as the sequence number of the reply. The replay (SYNACK) is then sent to the source, using the cookie, but no record is kept locally of the connection request. If and when the ACK arrives from the source as the third step of the handshake, the sequence number of the received message is used to authenticate the source. If the source is properly authenticated, the connection is established; otherwise, the ACK is discarded.

The scheme exchanges memory for CPU time, which makes sense because CPU time is much cheaper than memory. The biggest problem with this approach is that it breaks TCP semantics by not letting the server retransmit SYNACKS in case of packet loss. Other minor problems include loss of the initial round trip time measurement and the incoming maximum segment size, but those can be circumvented.

Reset Cookie Shenk [11] has devised a mechanism that, while not requiring changes to TCP, allows a server under attack to establish security associations with clients before connection requests are processed. In this approach, when the server is under attack and it receives a SYN packet, it will first see if the client had previously established a security association. If the client has a security association, the SYN is processed normally; if not, the server triggers a mechanism to create a security association with the client and discards the received SYN. The creation of the security association is triggered by the server. The server sends to the client an illegal SYNACK message with its sequence number replaced by a cookie.

In this scenario, according to the standard TCP specifications, the client responds to the anomalous SYNACK with a TCP reset (RST) bearing the server's cookie. When the server receives the reset, it verifies the cookie and records a security association with that particular client. This mechanism is backward compatible, it does not permit the unwarranted allocation of resources on the server [2] but has the obvious drawback of significantly increasing the first connection setup time. Servers with a small turn-around time with millions of clients like popular WWW services may significantly reduce their response time.

Random Drop Random drop can be seen as a generic, simple, and effective admission control mechanism for systems that support preemption and cannot support the storing of large amounts of state upon which to base admission decisions. We revisit random drop as a fourth SYN flooding defense solution. Although this idea has been already proposed and implemented [4, 7], it has not been correctly analyzed and assessed. Random drop maintains the TCP connection establishment protocol unchanged and allows the flexible tradeoff of defense effectiveness with resource requirements. In addition, this mechanism can be used in isolation or in conjunction with the Berkeley cookie scheme to dramatically reduce memory requirements.

In the random drop approach, when a SYN message reaches a server with a full connection pending queue, it replaces one of the pending requests chosen at random. The client whose connection entry is dropped is notified with a RST. If the replaced entry was previously generated by the attacker, the RST is simply lost in the network. If the replaced entry was from a legitimate client, the RST will cause the client's first attempt to communicate with the server to fail (returning end of file). The rationale for this approach is that by making the queue large enough, a server

[2] the number of security associations is proportional to the number of good clients only

Table 1: SYN Flooding Defense Mechanisms Comparison

	BSDI Cookies	Linux Cookies	Reset Cookies	Random Drop
Guarantee	NO	YES	YES	Prob.
Memory Immunity	NO	YES	YES	YES
Computing Immunity	NO	NO	NO	YES
Robustness	YES	NO	YES	YES
Good Performance	YES	YES	NO	YES

under attack can still offer an arbitrarily high probability of successful connection establishment. The obvious drawback is that the attacker can still occasionally deny connection establishment to a legitimate client. As we will see in the following Sections, both new analytical models and experimental evidence confirm that this scheme is extremely resilient to very high bandwidth attacks (or coordinated attacks) or attacks carried out against clients with relatively small connection pending queues.

3 Qualitative Comparison

The four approaches we described all have pros and cons and therefore should be carefully compared for an understanding of the tradeoffs. To this end, we summarize the key characteristic differences in the four approaches with respect to some main attributes. The first attribute, *Guarantee*, is the ability of the mechanism to provide availability to the clients in worst-case scenarios. This is perhaps the most important attribute because it directly impacts the usefulness of the mechanism. The *Memory* and *Computing Immunity* attributes signals when the server employing the mechanism cannot be forced to spend more memory or computing resources in proportion to the volume of SYNs sent by an attacker (for example, by computing cookies or allocating table entries as a result of malicious SYNs). The fourth attribute, *Robustness*, signals when a mechanism causes the TCP signaling semantics to be partially compromised. Finally, *Good Performance*, estimates whether a mechanism does not significantly affect TCP performance. Although these attributes are open to different interpretations that could result in completely different characterizations, in table 1 we report our intuitive analysis.

As shown in the table no mechanism is optimal. The BSDI approach fails to guarantee service and can be forced to compute cookies by an adversary. Although we listed this approach as providing robustness, we should point out that to reduce the size of the connection pending data structure this technique often also reduces the timeout that triggers the garbage collection of non-acknowledge SYN-ACKs. We do believe that the current TCP standard timeout of 75 seconds is a bit extreme but, for fairness, we should point out that lowering the garbage collection timeout could be interpreted as compromising the robustness of the approach.

The Linux cookie scheme guarantees service, but it is vulnerable to computing immunity compromises and, most importantly can compromise the TCP signaling semantics. Because no record is kept of received SYN messages, the server cannot retransmit SYN-ACKs to the client. If the last ACK message sent to the server is lost because of congestion, the client enters a state that is not directly addressed by the TCP standard and that can therefore compromise proper operation. The reset cookies scheme is the only one that provides both guarantee and robustness but fails in performance-related attributes. Finally, in our interpretation, the random drop scheme has the only drawback of guaranteeing service in a probabilistic, but not absolute, manner[3]

4 Random Drop Revisited

Randomly dropping connection pending entries when the queue is filled by SYN requests was one of the first proposed defense mechanisms. In the original formulation, the server simply replaces at random one of the entries in the queue and lets the client time out and later retry the connection with another SYN. To maximize the overall response time of a server under attack, this mechanism was modified by having the server send a RST message to the client. With this addition, if a client's SYN entry happens to be dropped by the server, the client is notified immediately with an EOF signal at the application level. Subsequently, the client can retry the connection establishment until the connection goes through (we will show, both analytically and experimentally, that with a finely tuned dropping scheme, a client is guaranteed connection establishment under most conditions).

4.1 Revised Analytical Model

It has been proposed that the rate of successful connection establishment C_{good} in a random drop scheme is

$$C_{good} = R_{good}(1 - 1/q)^{(R_{good}+R_{bad})T} \qquad (1)$$

where R_{good} and T are the average rate of arrival and the average round-trip time of all clients attempting a connection to the server, q is the size of the connection pending queue, and R_{bad} is the constant rate at which spoofed SYNs arrive to the server. The expression $(1 - 1/q)$ is the probability that a new arrival will not cause an existing entry to be dropped. Because each arrival is statistically independent, by elevating $(1 - 1/q)$ to the power of the number of expected arrivals during the servicing of the requests $((R_{good} + R_{bad})T)$, one can find the probability that a legitimate request will succeed.

An initial objection to equation 1 is that for a queue of size 1 it would wrongly predict a 100% success rate for the legitimate users. For this reason q should be really defined as the size of the connection pending queue plus 1. Another minor inaccuracy of equation 1 is that it does not take into account the fact that spoofed TCP SYNs expire after 70 seconds thus freeing some queue space. In practice both of these facts have a very small impact on the estimate of C_{good} and, therefore, for the remainder of the paper we will ignore them.

It is important to note, though, that the above estimate is grossly pessimistic because it does not take into account the fact that when successful connections are completed, entries are removed from the queue to mitigate the replacement probability. This phenomenon, which we accurately reproduced by analytical simulation, can be modeled with the expression

$$C_{good} = R_{good}(1 - 1/q)^{(R_{good}+R_{bad}-C_{good})T} \qquad (2)$$

which reduces the frequency at which connections are randomly knocked out by the rate of success (C_{good}).

All current TCP SYN admission control schemes drop all incoming SYNs when the receive queue is full. Equation 2 models a scheme in which incoming SYNs that find a full queue always cause some other entry in the queue to be dropped. Intuitively, one could guess that the best solution is somewhere in between these two extremes.

Preempting existing entries is not always best, because in some situations the queue may contain a large number of legitimate clients. In fact, R_{bad} is 0 when there are no SYN flooding attacks or other network problems leading to similar behavior. Obviously, well-intentioned connection requests should not be evicted, even if the queue gets full. In other words, we should maximize the chance that legitimate users be given the possibility of completing a connection if possible. To model the above idea, we introduce a factor K that determines the probability that any incoming SYN is accepted in the queue. This modifies equation 2 to

[3] Most communication services today only offer probabilistic guarantees and therefore this drawback could be cast as a normal expected characteristic of the network.

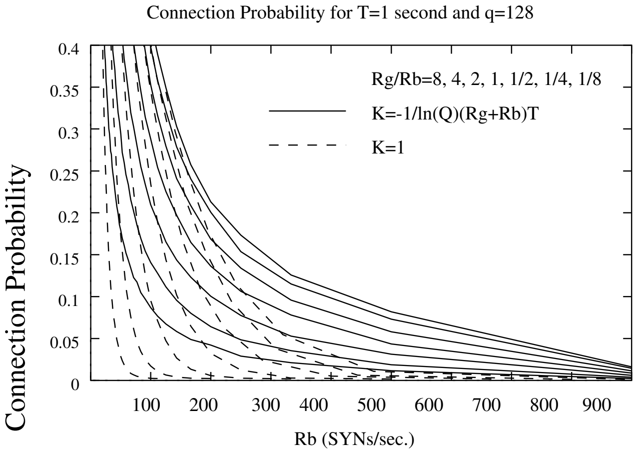

Figure 1: Connection probabilities for high ratios of spoofed SYNs

yield

$$C_{good} = K * R_{good} * (1 - 1/q)^{(K(R_{good}+R_{bad})-C_{good})T} \quad (3)$$

This expression uses the parameter K (truncated to be in the range [0 1], to modulate the rate at which both legitimate users and spoofed packets are accepted. To find the optimal value for K, we differentiate equation 3 and set $\frac{dC_{good}}{dK}$ to zero. Solving then for K we get

$$K = \frac{-1}{ln(1-1/q)(R_{good}+R_{bad})T} \quad (4)$$

Equation 4 suggests that a simple filtering technique could be used to maximize (or minimize if expressing a minimum) the connection rate of legitimate users. Through both further analytical derivation and empirical evidence (See Figure 1) it can be shown that this indeed an expression to maximize (not minimize) the success rate.

It is interesting that the filter only requires knowledge of the total incoming SYN rate and can therefore easily be implemented by recording the inter-arrival periods and combining them with simple integer arithmetic.

To demonstrate the effectiveness of the random drop scheme augmented by early drop filtering, we conducted some analytical simulations that plot the probability of success of legitimate users as a function of R_{good} and R_{bad} for a fixed time T.

Figure 1 shows the client success rates for the two different random drop systems: (1) $K = 1$ (i.e., no filtering), (2) $K = \frac{-1}{ln(1-1/q)(R_{good}+R_{bad})T}$ (with filtering). The simulated system has a queue size of 128 and a round trip delay time T of 1 second. These experiments were designed to highlight the effectiveness of the filtering, and therefore the parameters were chosen to explore a region of operation where the spoofed SYNs can greatly reduce the connection probability.

Although these worst-case conditions would seldom be encountered in practice, these results show that, in these cases, random drop with filtering is strictly better than random drop with no filtering. In addition Figure 1 shows that even for extremely adverse conditions (very high T and R_{bad} frequencies up to 300 SYNs/sec.) random drop augmented with filtering would still guarantee some limited amount of service to legitimate connection requests.

4.2 High-fidelity Simulation

We measure the performance of the random drop scheme with a high-fidelity distributed simulation tool developed in the ANCORS project [6]. ANCORS's simulation and prototyping environment was obtained by modifying a Linux operating system to allow its execution in user mode. The modifications of the operating system substituted the lower-level hardware-dependent procedures and interfaces with user-level counterparts. We deleted the file system support and incorporated all necessary configuration procedures (like *ifconfig* and *route*) into the service itself. Memory management was completely deleted and replaced by user-level memory allocation functions (*malloc* and *free*). The scheduling was also completely replaced by nonpreemptive threading offered by the simulation package CSIM [10].

The resulting service executes on a virtual timescale, and offers networking behavior identical to that of a real Linux kernel, providing a vehicle to instantiate high-fidelity distributed simulations of virtual networks [8]. One of the model's configuration functions accepts several different timing configurations to approximate the protocol stack timing behavior of four different kernels (SunOS 4.13, SunOS 5.5, Linux 2.02, and FreeBSD 2.2).

The virtual kernel offers the network application programming interface (API) of the real Linux counterpart and therefore can be used to reproduce a wide range of loading conditions. ANCORS's ability to add and delete threads can be used in this application to dynamically change loading conditions (by adding or deleting user-defined loading threads) or by injecting user-defined monitoring probes into the kernel so that specific parameters can be observed.

4.3 Simulated Network

Twelve workstations were configured, each with an ANCORS virtual host and were arranged in the topol-

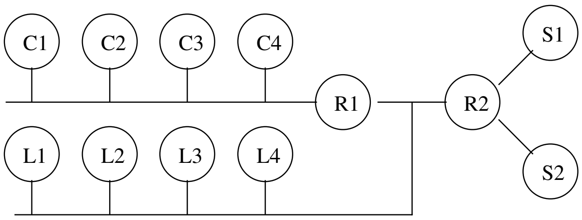

Figure 2: Simulated network

ogy depicted in Figure 2. Four clients C_1, C_2, C_3 and C_4 generate TCP connection requests to the server S_1, send data requests to the server, receive data from the server, and then close the connection. This behavior reproduces a typical scenario where an HTTP server replies to HTTP GET requests from a number of clients. The client and server replies are routed through nodes R_1 and R_2, which simply forward the data back and forth. Four additional hosts L_1, L_2, L_3 and L_4 and another server S_2 communicate through router R_2, thus causing congestion and routing delay. By varying the amount of data exchanged between L_1, L_2, L_3 and L_4 and server S_2, one can adjust the level of congestion in the experiments. L_1 also serves as the host from which the SYN attack is launched against S_1.

4.4 Simulation Parameters

An important parameter for the experiments is the determination of a reasonable rate at which an attacker can send SYN packets to the server. We vary this rate from 500 to 25 SYNs per second to capture typical rates that could be achieved on the Internet. Note that although these rates only translate to quite small effective bandwidths (a 60-byte SYN packet would consume 30,000 and 600 Kbyte/s, respectively) they capture the overheads that the packets may encounter in a realistic routing environment.

The routing delay encountered by the clients in all SYN flooding defense mechanisms, except the BSD cookie scheme, plays an important role in the effectiveness of defense. All approaches suffer in different ways from high routing delays. In the random drop mechanism this delay determines the likelihood that a malicious SYN may preempt a legal connection request while it is being acknowledged by the client (large T values). As the delay increases the probability of a preemption also increases. In the Linux cookie scheme, high routing delay and therefore high congestion can cause the loss of the last ACK packet of the TCP signaling handshake. If the last ACK packet is lost, the client enters in a half-open state and may pay a high penalty by having to time out on a reply from

Table 2: Random Drop Packet-level Simulation

Load	R_{bad}	Model	Actual	T	Total	Loss due
		Drop Rate		(ms)	Loss	to drop
high	25	3%	1.25%	109	11.16%	0.48%
high	50	5%	2.25%	105	6%	0.88%
high	100	10%	5.5%	130	22.29%	2.21%
high	200	25%	13.75%	179	40.86%	5.77%
high	250	31%	14%	185	44.11%	5.73%
high	333	43%	15.25%	201	49.29%	6.18%
high	500	78%	16.25%	336	63.37%	7.93%
low	25	0.2	0%	8	-1.02%	0%
low	50	0.5	0%	11.6	1.5%	0%
low	100	1%	0.5%	12.4	7.3%	0.24%
low	200	3	2.75	14	21.4%	1.3%
low	250	3%	3.25%	16.25	28.3%	1.57%
low	333	5%	5.25%	18	36.6%	1.91%
low	500	10%	8.25%	24	52.7%	3.91%

the server. In the reset cookie scheme, routing delay proportionally deteriorates performance by a factor of 66% because two more messages are necessary for each new client's connection.

In assessing the performance of the random drop scheme we chose to load our virtual network in such a way that clients experience average delays of 50 to 200 ms. Although these delays may seem optimistic and do not represent worst-case conditions, the queue size could be increased in proportion to the expected higher routing delays and thus outweigh higher values of T.

4.5 Results

In these experiments the virtual clients C_1, C_2, C_3 and C_4 each connect to server S_1 100 times. For each connection the clients send a small packet to the server and receive a reply. This behavior tries to model the downloading of a large HTML page containing numerous HTTP objects. As the page is loaded, the client opens a number of connections with the server to download all the different parts of the page. The experiments performed through our virtual network are very revealing and further strengthen our belief that random drop is an adequate defense mechanism.

Table 2 shows the predicted and actual rates of connection drop for different loading conditions and spoofed SYN rates. In the low congested network the model agrees fairly accurately with the behavior. For high SYN rates and high congestion the actual behavior is much better than analytically predicted. In fact, in this case, as the SYN rates increase the actual behavior diverges more and more from the analytical model. This behavior is due, in large part, to the fact that as the SYN rate increases in a highly congested network, many spoofed messages are lost, thus allowing higher-than-expected numbers of legitimate connections to go through. We have to perform more detailed experiments to further investigate this important phenomenon because, if confirmed under a wide range of loading conditions, may definitively argue that brute force defense approaches like the BSDI cookie scheme or probabilistic approaches like the random drop may be the best solutions.

Table 2 also shows the average performance degradation of the four clients when the server is under attack. We report the total loss in performance (due to both failed connection retries and the congestion caused by the SYN stream) and the performance loss due to connection retries only. As shown in the table, because we send RST messages upon a drop, the drop rates translate to relatively low losses in performance due to connection retries. Most of the performance loss is due to the higher congestion introduced by high SYN rates. Notice, though, that as the congestion increases, both because of more legitimate user traffic and more spoofed SYN traffic, the relatively low drop rates translate into relatively high performance losses. This occurs because in highly congested networks each connection retry takes longer and therefore increasingly impacts overall performance.

Two important conclusions may be drawn from these experiments. One result indicates that a brute force defense approach or a more efficient probabilistic approach like the random drop may be the best solution because very high SYN rates are not possible in a realistic environment. Another result is that random drop works well in both low congestion as well as high congestion by keeping the clients performance losses below 10% even for very high spoofed SYN rates.

5 Conclusion

We have qualitatively compared several mechanisms to defend against the SYN flooding denial-of-service attack and have shown that no solution is optimal. We revisited the adoption of the random drop approach by producing a better analytical model of its behavior in the context of defending against the TCP SYN flooding attack. With the new model, we derived a simple filter that can improve random drop performance in worst-case scenarios. High-fidelity distributed packet-level simulations partially agree with our analytical model and illustrate that in a real environment with high congestion the random drop approach would behave much better than expected. We have also shown that performance loss due to connection retries stays well below 10% under a wide range of loading conditions.

References

[1] Syncookies mailing list. *ftp://koobera.math.uic.edu/pub/docs/syncookies-archive*, 1996.

[2] Inc. Berkeley Software Design. Bsdi releases defense for internet denial-of-service attacks. *http://www.bsdi.com/press/19961002.html*, October 1996.

[3] D.J. Bernstein. Syn floods – a solution. *http://www.op.net/~jaw/syn-fix.html*, 1996.

[4] Alan Cox. Linux tcp changes for protection against the syn attack. *http://www.wcug.wwu.edu/lists/netdev/199609/msg00091.html*, September 1996.

[5] Rex di Bona. Tcp syn attacks - a simple solution. *http://www.cctec.com/maillists/nanog/historical/9610/msg00155.html*, Oct 1996.

[6] P. Porras L. Ricciulli, N. Shacham. Ancors: Adaptable network control and reporting system. *SRI technical report SRI-CSL-9801*, 1998.

[7] Sun Microsystems. Sun's tcp syn flooding solutions. *http://ciac.llnl.gov/ciac/bulletins/h-02.shtml*, October 1996.

[8] L. Ricciulli. High-fidelity distributed simulation of local area networks. *Proceedings of the 31st Annual Simulation Symposium*, Boston, April 1998.

[9] C. L. Schuba, I. V. Krsul, M. G. Khun, E.H. Spafford, A. Sundram, and D. Zamboni. Analysis of a denial of service attack on tcp. *1997 IEEE Symposium on Security and Privacy*, 1997.

[10] H. Schwetman. CSIM: a C-based, process-oriented simulation language. Technical report, MCC, 1989.

[11] E. Shenk. Another new thought on dealing with syn flooding. *http://www.wcug.wwu.edu/lists/netdev/199609/msg00171.html*, Sept 1996.