Implementing and testing dynamic timeout adjustment as a DoS counter-measure

Daniel Boteanu Édouard Reich
José M. Fernandez
École Polytechnique de Montréal
C.P. 6079, succ. centre-ville
Montréal, Québec, Canada, H3C 3A7
{daniel.boteanu|edouard.reich|jose.fernandez}@polymtl.ca

John McHugh
Dalhousie University
6050 University Avenue
Halifax, Nova Scotia, Canada, B3H 1W5
mchugh@cs.dal.ca

ABSTRACT

In this paper we experimentally analyse various dynamic timeout adjustment strategies in server queues as potential counter-measures against degradation of service attacks. Previous theoretical work studied the relative performance of both coarse-grained threshold-based timeout and fine-grained adjustment strategies where the timeout value is as the number of connections in the queue varies. In addition, two methods for removing timed-out connections were explored: the deterministic method where the expiry time is determined at connection arrival depending on the timeout value at that moment, and the deferred method where connections are continuously polled and flushed when the time-in-queue is larger than the current timeout value.

We report on experiments performed on a laboratory network where these strategies were tested against various configuration and attack parameters. The experimental results confirm the conclusions previously obtained from mathematical modelling and simulation, i.e. that a) finer-grained dynamic adjustment performs better than coarse-grained or no adjustment, and b) that the deferred method performs better than the deterministic one. Furthermore, our implementation of these counter-measures is very efficient and transparent with the respect to the servers and applications it tries to protect. It could therefore be easily integrated into existing OS and applications or implemented in separate network devices, either on dedicated machines or network appliances.

1. INTRODUCTION

Denial-of-service attacks have been plaguing the Internet for more than a decade. They have been a topic of much research for almost as long, and much has been done and written about modelling them and potential counter-measures against them (see [1, 2] for complete surveys on the topic). Some might even say that they have fallen “out of fashion” as a security research topic. However, recent events in Estonia (see [3] for a very informative and quantitative technical summary) have brought back DoS attacks back to the forefront: the prospect of a whole country’s Internet infrastructure being almost brought to its knees is a very real prospect, as it was demonstrated.

The amount of effort or resources expended by the attacker (whether bandwidth, expendable source IP addresses or even individual botnet machines) to attack a single target, is in most cases negligible compared to the amount of resources the defender would have to spend to maintain an equivalent availability of service. Previous work [4, 5] has tried to discuss and formalise such tradeoffs, and several counter-measures based on protocol modifications have been proposed to try to tip these tradeoffs in favour of the defenders. However, it is commonly assumed amongst security experts that with the current availability of botnets from which to launch these attacks, there is little one can do to prevent a single target from becoming completely flooded and hence unavailable; the tradeoffs are just hopeless. Nonetheless, in the context of large-scale orchestrated DoS campaigns such as the one in Estonia, potentially involving hundreds or even thousands of targets, such tradeoffs might not be so advantageous to an attacker with finite resources. In addition, it is not only necessary to understand the tradeoffs in the context of crippling DoS attacks, where the target is reduced to 0% availability, but also the tradeoff between resources expended to degrade to or maintain an equivalent quality of service (QoS). In other words, these resource tradeoffs must also be understood in the context of Degradation of Service attacks [6, 1], where the objective is not necessarily to make a target completely unavailable but rather substantially decrease its QoS.

In previous work [7], we have explored these tradeoffs in the context of flooding attacks on connection-oriented protocols, the quintessential example of which is the SYN-flood attack on the TCP protocol. We describe in the next section the various counter-measures we studied and the attack models against which we evaluated their QoS maintenance performance. We report on the lab experiments we have performed in Sect. 3, describing laboratory setup, implementation details and testing and measurement methodologies. We describe the experimental results and compare with the theoretical results previously obtained in Sect. 4. We discuss the limitations and practical applicability of our work in Sect. 5 and summarise our findings and conclude in Sect. 6.
2. PREVIOUS WORK

In these attacks, the defender resources being expended are available slots in a pending connection queue, while the attacker resources are numbers of connection attempts (e.g. SYN packets sent, in the case of SYN flood). The measure of QoS is the percentage of legitimate connection attempt requests that get serviced by the target. In [7] we considered the relative performance of various queue management strategies with respect to maintaining this QoS measure. In particular, since illegitimate connection that make it to the queue never get completed (e.g. because the corresponding ACK packets are never sent by the attacker), it would seem intuitive that lowering the timeout values when the queue is under attack would result in more illegitimate request being flushed out than legitimate ones. On the other hand, increasing it again when the queue empties out would prevent inadvertently flushing out legitimate connections when no longer under attack. With this in mind, we considered three types of strategies for dynamically adjusting the timeout values:

A. The traditional fixed timeout strategy, where the timeout value is always the same, regardless of queue occupation.

B. The threshold strategy where the timeout changes between two fixed values, as the number of connections in the queue crosses a pre-defined threshold. This coarse-grain adjustment method is not new and is already implemented in the TCP stack of some operating systems (OS), e.g. Microsoft Windows Server 2003 [8].

C. The linear method, a straightforward generalisation of the former, where the timeout value is determined according to a linear function depending on the number of connections in the queue, with two pre-defined empty- and full-queue timeout values.

Furthermore, we considered two timeout enforcement methods for flushing connections from the queue when the timeout-out is dynamically adjusted:

1. The deterministic method, where the expiry time for each connection is deterministically set when the connection arrives in the queue.

2. The deferred method, where connections in the queue are continuously polled, and flushed if they have been in the queue longer than the current timeout value.

Finally, two attack models were considered:

I. the Poisson attack model, a simpler albeit not very realistic model, where the interarrival times of illegitimate connection attempts follow an exponential (i.e. a Poisson model) distribution.

II. the burst attack model, where illegitimate connection requests arrive in (almost) instantaneous bursts of a fixed number of attempts, with burst spaced at a fixed burst interarrival time (BIT).

Using Markov chain-based queue models (in the case of Poisson attacks) and a custom-built event-driven simulator (for both types of attacks), we were able to verify that:

i. For all strategies and methods, the tradeoff between attack rate (connection attempts per second) and server queue size is essentially linear. Because of this tradeoff, the only parameter that significantly influences QoS degradation, for a fixed strategy and method, is the ratio between attack rate and queue size, which we called the relative attack virulence.

ii. Fine-grained timeout adjustment (linear) always outperforms coarse-grained (threshold) adjustment, and the latter outperforms the fixed timeout strategy.

iii. The deferred method generally performs better for than the deterministic one, except for the case of the linear deferred which performs worse than the linear deterministic method for burst attacks.

iv. In the case of burst attacks, some strategies and methods are quite sensitive to attack parameter optimisations. In particular, the lowest QoS for each defensive strategy is achieved when the BIT is set such that the queue is filled with a single burst. We called this phenomenon the queue resonance effect.

These results, if confirmed in real-life settings, would be of high practical interest. They would indicate how simple choices in queue management algorithms could result in dramatic improvements in resilience against QoS degradation attacks. Of course, this is only true for attack virulences that are not too low or not too high, i.e. attacks whose virulence is within the window of interest, since for attacks outside this window the QoS degradation is equally negligible or overwhelming, for all strategies. Therefore, we enunciate the following hypothesis based on the theoretical evidence of [7]:

Main Hypothesis. Within the window of interest (attack virulences within 1/8 and 8 s⁻¹), finer-grained timeout adjustments strategies using the deferred method will always perform better than ones using coarser-grained adjustment or the deterministic method against SYN-flood attacks.

Until we can verify this hypothesis for real-life networks, we have to content ourselves with gathering supporting evidence from testing on laboratory networks. The setup and methodology we used to do so in the case of SYN-flood attacks is described in the next section.

3. EXPERIMENTAL SETUP AND TESTING METHODOLOGY

The XXXX components of our experimental setup are the following:

1. The attack traffic generator, generating illegitimate SYN packets on the network.

2. The legitimate traffic generator, attempting to establish fully fledged TCP connections.

3. The network, on which both kinds of traffic travel.

4. The target host, whose TCP stack half-open connection queue is being flooded.

1Intuitively, the attack virulence indicates how many times per s the attack could fill the queue, if there were no legitimate requests and a very low timeout value.
3.1 Attack Traffic Generator

For this component, we used the IXIA 400T a special purpose traffic generator chassis, built for for performance and conformance testing of network applications. We model used has four separate Ethernet ports, capable of generating traffic up to 1 Gbps.

In order to generate the two types of malicious traffic we wanted to test (Poisson and burst), we used the IxExplorer application that runs on the IXIA hardware. Since neither the hardware nor the software can natively generate Poisson traffic, this type of attack was synthesised byby cyclically sequencing 255 different modes, each mode consisting in sending one single SYN packet. For each attack rate, pauses between modes were statically set to random values following an exponential distribution. We performed a Kolmogorov-Smirnov test on the inter-arrival times of the IxExplorer-generated traffic measured on the server. The maximum difference between the theoretical exponential and the observed cumulative distribution functions (cdf) was as low as 0.12 for an attack of 1000 packets/s, which confirms that the served cumulative distribution functions (cdf) was as low as 0.12 for an attack of 1000 packets/s, which confirms that the traffic follows the Poisson process model closely.

For burst traffic, we ran experiments with different BIT values, where the number of packets in a burst was chosen so that overall attack rate remained the same for all experiments. IxExplorer allowed us to generate burst attack traffic using only one mode, the burst mode, for BIT < 8s. For Burst attacks with BIT ≥ 8s, several modes were sequenced, each mode sending an entire burst followed by one or several “pause” modes. In the first case, we were able to script several experiments at various BIT values, one after the other. A pause at least as long as the server’s largest timeout value was inserted between attacks in order to prevent the experiments results from being contaminated by previous ones.

3.2 Legitimate Traffic Generator

We used a home-made C++ application to generate the legitimate traffic necessary for successful TCP handshake. Both the SYN and ACK messages were sent with exponentially distributed inter-arrival times. Contrary to TCP stack implementations in standard OS, this test application will not send a SYN retry message if there is no response from the server. This was a deliberate choice meant to keep the connection attempt rate constant and independent of the connections complete rate. For performance measuring purposes, all the legitimate SYN messages came from the same IP address. This address is discriminated only when counting the total number of legitimate connection attempts. After a TCP handshake is completed, the application will send a RST message in order to free the connection on the server side. The libpcap library was used to handle IP packets. We deployed the legitimate traffic generator on a dedicated machine running Gentoo Linux, with 2 GB of memory, which allowed us to experience with queue sizes up to 16384.

Rather than modifying the TCP stack kernel code, which neither easy nor would it be practical in real-life deployments, we chose to implement the dynamic timeout strategies on a separate Queue Guardian (QG) application, in manner transparent to the server and the legitimate clients. The QG has four different roles:

1. It maintains an up-to-date mirror of the server queue. This is achieved by sniffing on the network connection and interpreting packets being send and received by the server. We used the libpcap library to sniff all the IP packets on the network.
2. It polices connections in the mirror queue, according to the chosen timeout adjustment strategy and connection expulsion method.
3. The deferred method generally performs better
4. The deferred method generally performs better
5. It forces the server queue to drop the same connections dropped from the mirror queue, according to its own queue management strategy and policing method. This is achieved by sending RST packets to the server. The source IP address and TCP port are spoofed so that the message appears to come form the client that initiated the connection. In order to send the spoofed RST packets at high speeds, this role was implemented using raw sockets.
6. It regularly logs the state of the queue, as well as the number of different types of packets sniffed on the network. This log is used later for evaluating the performance of the timeout strategy under test.

For the deterministic method, we used a priority queue implemented as a red-black tree to store the connections, ordered by their expiration time. When all legitimate connections get served, the complexity of the algorithm is \(O(l\log(c)N_m + cN_l)\), where \(c\) is the size of the server queue and \(N_m\) and \(N_l\) are the number of SYN-ACK responses sent to malicious and legitimate SYN packets, respectively. In the deferred method, only the oldest connection in the queue needs to be analysed: if it is present in the queue for longer than the current timeout, it will be dropped from the queue. Hence, a single FIFO ring-buffer can be used to implement this method. When all legitimate connections get served the complexity of the algorithm is \(O(N_m + cN_l)\). In practice, however, the legitimate connections are almost always at the end of the queue so only \(N_m + N_l\) atomic operations need to be performed. Finally, and for performance reasons, we chose to implement each of these four roles in a separate thread in the QG application. The QG is run on a separate machine, based on a Intel Core 2 Duo processor at 2.16 GHz.

3.3 Network Setup

A 16-port gigabit switch (Linksys SRV-2016) was used to connect all these components together. The legitimate traffic generator machine, the server and the IXIA traffic generator were each connected to a separate port on the switch. For sniffing purposes, the QG machine was connected on a switch port setup to mirror the server port. For sending RST packets, a separate card on the QG machine was connected to another network port on the switch. Other deployment schemes are possible as well and we will discuss them in Sect. 5. Fig. 1 illustrates the network connections between the components we have used.

3.4 Testing Methodology

In all the experiments we ran, the following steps were followed in sequence:

1. The server queue size was configured with the value required for testing.
2. The server timeout was configured to be at least as long as the longest timeout on the QG. This way, all the connections drops were triggered by the QG.

3. The legitimate connection traffic generator was started with the connection arrival and connection completion rates required for testing.

4. The QG was configured with the required parameters and started.

5. The attack traffic parameters were configured in IxExplorer.

6. The attack was started and the experiment was run during a period of time ten times longer than the longest timeout on the QG.

7. The connection success rate was computed based on the QG’s log.

Connection completions correspond to ACK messages being sent to the server. Legitimate connection attempts correspond to SYN messages being sent from the legitimate IP address. The connection success rate was computed as the ratio between the connections completed and the legitimate connection attempts during the attack.

4. RESULTS AND ANALYSIS

We measured the performance of the two dynamic timeout strategies, threshold and linear along with the fixed timeout strategy for comparison purposes. For the dynamic strategies, we tested both the deterministic and deferred methods of assigning timeouts to connections. We compare these results with those obtained in a previously built home-made traffic simulator (described in [7]) that implements these strategies. The attack traffic was generated using both models described above. In both cases we tested the attacks against a small queue size of 128 and a more reasonable queue size of 1024.

In the case of Poisson attacks, we explored virulences from 0.015 to 8 s\(^{-1}\). The attack speed varied from 2 to 1024 packets/s when testing against a queue size of 128 and from 32 to 8192 packets/s when testing against a queue size of 1024. The legitimate connections attempt rate was of 10 cnx/s and the mean RTT time for the legitimate traffic was of 200 ms (as inspired from experimental measures in [9]).

The fixed timeout strategy used a timeout value of 10 s and the dynamic timeout strategies used empty- and full-queue timeout values of 10 s and 200 ms, respectively. Results for the tests against a queue size of 128 are shown in Fig. 2.

Overall, the experimental results are very similar to the simulation results. The average difference between the simulation and experimental results is 2%. The greatest discrepancy (17%) was measured for the linear deterministic strategy faced with an attack of virulence 8 s\(^{-1}\) against a queue size of 1024.

As anticipated from the simulations, results for low and high virulences are not interesting. For low virulence values (< 0.05 s\(^{-1}\)) the attack is not strong enough to degrade QoS at the server, even when using the fixed timeout strategy. For very high virulence values (> 8 s\(^{-1}\)) the attack is so strong that none of the dynamic timeout strategies can maintain a connection success rate greater than 50%. In between these values, the window of interest, several conclusions can be drawn that confirm the predictions of previous theoretical work.

First, the dynamic timeout strategies perform better or equivalent than the fixed timeout strategy. We measured differences of up to 85% between the linear deferred strategy and the fixed timeout strategy, and up to 50% between the threshold deterministic and the fixed timeout strategy around virulences of 1 s\(^{-1}\). Second, the deferred technique always performs better than the deterministic technique. Differences up to 30% can be observed between the deferred and the deterministic techniques around virulences of 2 s\(^{-1}\). This is due to the fact that the deferred technique is more reactive, deciding whether a connection should expire or not based on the current status of the queue, as opposed to the status of the queue at the time of the connection arrival in the case of the deterministic technique. Third, the linear
timeout strategy performs better than the threshold timeout strategy with the exception of the deterministic technique for virulence values greater than 1 s⁻¹. The threshold strategy has an overprotective behaviour when faced to an attack, and this seems to correct some of the delayed reactivity of the deterministic technique for medium and high virulence values.

In order to study the generality of these results with respect to different attack types we also used a deterministic process to generate bursts attacks. A virulence of 0.5 s⁻¹ was chosen, which corresponds to attack rates of 64 and 512 packets/s when testing against queue sizes of 128 and 1024, respectively. This virulence value is illustrated in Fig. 2 by the vertical black line. The average connection success rates over 9 experimental runs and their corresponding variance for the queue of size 128 are illustrated in Figs. 3 and 4. The vertical black lines at BIT = 0.015625 s in Figs. 3 and 4 represent that the packet inter-arrival time is the same as the mean packet inter-arrival time in the Poisson experiments at virulence 0.5 s⁻¹, marked by the vertical black line in Fig. 2. The dashed black vertical lines at BIT = 2 s in Figs. 3 and 4 mark the point where one single burst would fill up an empty queue entirely. Fig. 5 offers a three-dimensional illustration of the correspondence between the Poisson and the Burst figures.

Two “phases” can be observed when analysing the burst attack results. The “liquid phase” at the leftmost part of the figures with BIT < 2 s, corresponds to attack traffic bursts smaller than the queue size. The “solid”, rightmost phase, for BIT > 2 s, corresponds to attack traffic bursts greater than the queue size. The resonance effect is created at BIT = 2 s, corresponding to bursts of the same size as the server queue. In simulations, the fixed timeout strategy performance is practically null at this value. This is due to the fact that both the simulation attack and legitimate traffic start at the same time and the attack burst instantly fills up the entire queue. During a period 10 s, equal to the timeout value, the queue is full and no legitimate connection attempts can be processed. After this period, exactly after the malicious connections are dropped from the queue, the following burst arrives and fills up all the queue once again.

We can see that the burst traffic is perfectly synchronized with the queue timeout. In experiments, however, we do not observe the same behaviour and results for attack bursts of the same size as the queue.

First of all, the legitimate traffic and the malicious traffic are not synchronised. By the time the first attack burst arrives, around three slots in the queue are already used by legitimate connection attempts, so three of the attack packets are discarded by the server. During a period of 10 s, only the number of slots used by legitimate connection attempts at the time the first burst arrived will be available. However, because there are only 10 legitimate connection attempts per second, and because the legitimate connections complete rather quickly (5 every second), the few free slots in the server queue are enough for a large percentage of legitimate connection attempts to complete. Furthermore, in experiments, the burst are never instantaneous due to packet transmission times and eventual collisions in the Ethernet network. This allows for legitimate connection attempts to infiltrate the burst and thus reduce the burst efficiency for the attacker. Due to the above mentioned factors, we can say that the network acts as a “low-pass filter” thus greatly diminishing the resonance effect. In simulations, the fixed timeout strategy is influenced by the resonance effect at “harmonics” with BIT = 2⁻ᴷ s, for K = 0..5. In experiments, however, the resonance effect is absorbed by the network. The only two strategies that seem to be slightly affected by the resonance effect in experiments, are the linear deterministic and the threshold deferred timeout strategies, and this only for the harmonic at BIT = 1 s.
Is it important to note that the deferred method, which performs better than the deterministic one, is also more robust and consistent, having lower standard deviation values. The fixed timeout strategy, on the other hand, is the most instable, both in simulation and in experiments, with maximum standard deviation values of over 10%.

5. LIMITATIONS AND FUTURE WORK

Although the QG application itself has a low CPU overhead, there are two limitations that prevented us from testing higher attack rates and higher queue sizes.

The first limitation is due to the network architecture we implemented. The malicious SYN packets and the QG-generated RST packets are sent through the same switch and thus, at high attack rates, some RST packets are dropped by the switch due to Ethernet collisions. This creates a cumulative difference in the size and content of the mirror queue compared to the server queue. For example, an average of 0.3% of the resets sent by the QG are dropped by the switch for an attack rate of 8192 packets/s, which is equivalent to 6.65 Mbps. For attack rates higher than 8 Mbps the RST packet loss is so significant that the QG’s mirror queue is completely corrupt in a matter of seconds. This limitation can be overcome by using a different network architecture, where the QG would be deployed either directly on the server or on a network equipment directly connected to the server, while still preserving the transparency to both the server application and OS and the legitimate clients.

The second limitation is due to CPU consumption on the QG machine. When using the deterministic technique to protect a queue size of 1024 with 10 legitimate connections per second against an attack of 6.65 Mbps, the average CPU usage was 9%. The deferred technique consumed in average 8% of the CPU, for the same attack parameters. Most of this CPU time is spent in kernel mode, handling the sending and sniffing of IP packets. We ignored the previous architectural limitation of RST packets loss and managed to reach attack rates as high as 300 Mbps for the deterministic method and 62 Mbps for the deferred method, compiled in debug mode, before the approaching 100% CPU usage, and hence missing some of the packets that traverse the network. It is our belief that a more powerful machine should be able to handle up to gigabit attack traffic.

6. CONCLUSIONS

In this paper, we tested the hypothesis that within the window of interest (attack virulences within 1/8 and 8 s⁻¹), finer-grained timeout adjustments strategies using the deferred method perform better than ones using coarser-grained adjustment or the deterministic method. We implemented both a fine-grained, linear and a coarse-grained, threshold dynamic timeout adjustment strategy in their deterministic and deferred variants. The performance measures obtained in these laboratory experiments for the different strategies against Poisson attack traffic was consistent with the performance measures against Burst traffic. First, that using a dynamic timeout strategy is always a good idea. Second, that the deferred method performs better than the deterministic technique, and slightly lower CPU usage, due to having a lower algorithmic complexity Third, the linear, fine-grained timeout adjustment strategy performs better than the threshold, coarse-grained timeout adjustment strategy when in their deferred implementation. Finally, the resonance effect that we expected when testing against burst attack traffic is very limited in experiments, due to network delays created by network equipment buffers, Ethernet collisions and non-instantaneous packet send times.

7. REFERENCES