A Simulation Study of the Proactive Server Roaming for Mitigating Denial of Service Attacks*

Chatree Sangpachatanaruk[§] Sherif M. Khattab[†] Taieb Znati[†,§] Rami Melhem[†] Daniel Mossé[†]
[§]Department of Information Science and Telecommunication
[†]Department of Computer Science
University of Pittsburgh, Pittsburgh, PA 15260
{chatree, skhattab, znati, melhem, mosse}@cs.pitt.edu

Abstract

The main goal of the NETSEC project is to design and implement a framework for mitigating the effects of the node-based and link-based DoS attacks. Our strategy employs three lines of defense. The first line of defense is to restrict the access to the defended services using offline service subscription, encryption and other traditional security techniques. The second line of defense is server roaming*, by which we mean the migration of the service from one server to another, where the new server has a different IP address. Finally, each server and firewall(s) implement resource management schemes as a third line of defense. For example, deploying separate input queues to allocate different classes of service requests. In this paper, we show our simulation study on the second line of defense, the* server roaming*. The design and procedure of the sever roaming on the NS2 is described. The promising results of applying the server roaming to mitigate the DoS attack in the simulation are also shown with analysis.*

1 Introduction

Along their paths between a client and a server, network packets consume various kinds of resources including access link bandwidth, router buffers, server memory buffers, file descriptors, operating system processes, etc. By injecting maliciously designed packets, network Denial of Service (DoS) attacks aim to deplete some of these resources so that no more resources are available to service legitimate requests. For example, in the TCP SYN attack, attackers send illegitimate TCP SYN packets to a TCP server with the intention of depleting the memory buffers which hold TCP state information of pending connections [9]. Another example is depleting a link's bandwidth by flooding the link with illegitimate packets at such a high rate that causes legitimate packets to get dropped from router buffers. Typically, the link targeted by such an attack is a low-bandwidth link connecting a server, or a cluster of servers, to the high-bandwidth Internet backbone. The synchronized participation of many attacking nodes can achieve such a high illegitimate packet rate using a Distributed Denial of Service (DDOS) attack [6, 7].

We categorize DoS attacks with regard to the location of the resources they are targeting. Whereas *node-based* DoS attacks focus on depleting the resources at servers, the main goal of *link-based* DoS attacks is to saturate critical links in the path of legitimate requests. Typically, node-based attacks make use of known vulnerabilities of operating systems and network protocol implementations. Also, due to current link speeds, the number of illegitimate packets needed to launch a node-based attack is much less than their link-based counterparts. If the server can distinguish between legitimate and illegitimate packets, DoS attacks can be stopped by simply dropping the illegitimate packets. One can think of classifying packets based on their source address, restricting access to the server to packets coming from known legitimate source addresses. Unfortunately, current Internet routing protocols enable IP spoofing, where the source address of a packet can be a fake one. Another solution is to use more complex cryptography-based authentication schemes. The overwhelming the authentication process by many illegitimate requests, however, is a DoS attack in itself.

Current DoS research focuses on three tracks: (1) *mitigation* techniques: which aim to mitigate the effect of DoS attacks assuming that we can not accurately distinguish illegitimate packets; (2) *classification-based*

*The authors were supported in part by NSF under grant ANI-0087609.

techniques: which move the classification process away from the server, either to achieve simpler classification criteria (like the case of ingress filtering [2]) or to replicate the classification process, making it less vulnerable to attacks; and (3) *attack-tracking* techniques: which try to identify DoS attack sources, to stop them and discourage attackers.

Our goal in the network security group (NETSEC) at the University of Pittsburgh is to design and implement a framework for mitigating the effects of both types of DoS attacks, node-based and link-based. Our strategy employs three lines of defense. The first line of defense is to restrict the access to the defended services using offline service subscription, encryption and other traditional security techniques. The second line of defense is *server roaming*, by which we mean the migration of the service from one server to another, where the new server has a different IP address. Finally, each server and firewall implements resource management schemes as a third line of defense. For example, deploying separate input queues to allocate different classes of service requests.

We begin by studying the cost and benefit of the second line of defense, the *server roaming*, via a simulation and a real prototype implementation. In this paper, however, we will focus on the simulation modeling and analysis. The organization of the paper is the following. First, the background of server roaming is presented in Section 2. The overview of our simulation model is described in Section 3. The experiment design and procedure are explained in Section 4. Section 5 shows the results and analysis. Last, the conclusion and the future work are presented in Section 6.

2 Background

We define the server roaming as a framework to mitigate the effects of DoS attacks. The active server changes its location among a pool of servers to defend against unpredictable and likely undetectable attacks. Only legitimate clients will be able to follow the server as it roams.

The motivation behind our scheme of server roaming is three-fold. First, it defends against unpredictable and undetectable attacks. We assume that attacks and intrusions occur and it is not always possible to detect such malicious activities. The proactive nature allows our scheme to tolerate undetected attacks, while the reactive component allows the scheme to benefit from current advances in intrusion detection techniques [1].

Second, server roaming helps to define accurate and efficient packet filters to be deployed at the network boundaries. Except for a small transitional interval, all packets except those destined for the IP address of the current active server can be safely considered illegitimate and can thus be dropped. This adaptive filtering is done at the firewall box installed at the entry point of each server.

Third, roaming allows for the detection of misbehaving legitimate clients. A legitimate client violating its QoS contract with the service is provided a second chance after the service roams to the next server. If this client keeps violating its contract, all its further traffic is dropped (and service re-negotiation needed to resume service). A client persisting on violating its contract is faulty, doing that on purpose, or has already been infiltrated by a malicious attacker. In any of these cases, the client loses its status as a legitimate client. It should be noted that detecting misbehaving legitimate clients is only a side advantage of server roaming.

By physically moving the service from one server to another and cleaning the state of the old server, our scheme avoids the limitations of logical roaming schemes, such as *IP hopping* [5]. Only changing the IP address of the server without physically moving the service retains the server vulnerable to malicious state entries possibly implanted during the attack. Also, because IP hopping allows for filtering at the boundary of the network, the server will be still vulnerable to the attacks launched from inside the network. Moreover, illegitimate packets will still go to the same network after the address is changed, potentially causing resource depletion at intermediate nodes on the path.

Each client is required to establish a trust with the system before perceiving a knowledge of the secret location of the active server. Once a client has this knowledge, it will be able to track down the server using a self computation. Our server roaming framework also deals with the in-process connections. All connections will be migrated to the new server as the active server moves. The effectiveness of our framework relies on how the legitimate clients know where the active server is and how we migrate the in-process connections. The research related to these questions are reviewed in the following subsections.

2.1 Tracking the Active Server Location

To be able to know the active server location, a client need to have at least two information: the server address and the time that the server will be active. These information can be simply obtained by using a series of communication. To avoid the DoS attacks on the Internet, however, clients and servers need a secure communication that provides privacy and integrity to protect the information.

We propose a secure, proactive and time-triggered

Proceedings of the 36th Annual Simulation Symposium (ANSS'03)
1080-241X/03 $17.00 © 2003 IEEE

roaming scheme. This scheme defines an upper bound on the time interval between consecutive server roaming instances. This upper bound is adaptively changed to reflect the current threat level. For instance, in a high threat period this upper bound is set to be small, while it is set to a larger value in normal conditions. More investigation on the effect of this bound is left for future work. The exact roaming interval is described below.

Our scheme utilizes the idea of one-way hash functions, such as SHA-1 and MD5 [4, 3, 8]. For this class of functions, it is computationally infeasible to reverse the direction of the function application; that is, given the output of such a function, it is computationally infeasible for an adversary to know the input. The light-weight computational overhead of hash functions allows for a simple and efficient implementation, and is suitable even for the mobile clients that have computational and power constraints.

Our scheme involves an initialization phase. When a legitimate client subscribes to the service, it receives some information that allows it to create a secure channel with the service. Before it starts using the service, a client waits until it receives a message from the service carrying the necessary information for calculating roaming times and roaming target addresses. This information includes a sequence of keys such that each key is used to generate the roaming time and the address of the next roaming target for the roaming instance.

2.2 TCP-Migration

Two well-known TCP-connection migration mechanisms are the *TCP-Migrate* [11], developed at MIT, and the *Migratory-TCP* [13], developed at Rutgers University. Both attempts provide the framework for moving one end point of a live TCP connection from one location and reincarnating it at another location having a different IP address and/or a different port number. Both mechanisms deal with four issues in a slightly different way: (1) how the TCP connection is continued between the new end points; (2) impact on the network stack and application layer in both the server and the client sides; (3) how to recover both TCP and application states; and (4) when to trigger the migration mechanism. The last two issues are considered independent of the actual migration framework and are presented as examples of possible usage of the mechanism.

In MIT's TCP-Migrate, during connection establishment, the migration feature is requested through a TCP option (Migrate-Enabled). By the means of a handshaking protocol, a shared key is established between the two connection end points. As per a migration request from one end point, represented by another TCP option (Migrate), the TCP control block at the fixed end point is updated to reflect the new location of its peer. To protect against connection hijacking, the secret key agreed upon during the connection establishment should accompany the migration request. As an application of the TCP-Migrate mechanism in a fine-grained fail-over scheme [10], state recovery in the new server is achieved via periodic state updates from the old server to the server pool. A widget implementation is responsible for extracting HTTP state from TCP packets. The migration request is issued by a new server and triggered by an overload at the old server, detected by a health monitor. Implementations at both the transport and session layers are available. The TCP-layer in both the server and the client needs to be changed. However, no application layer updates are necessary, though the widget implementation is already application-dependent.

During connection establishment between two Migratory-TCP-enabled peers, a list of available servers, along with a certificate for each server, is passed from the server to the client. A migration request, also implemented as a TCP option, consists of both the certificate of the new server and the connection id (client IP address, client port, old server IP address, old server port) of the migrating connection. However, no security measures are implemented to protect the migration process. State recovery at the new server is achieved either on-demand, that is, when the client sends the migration request to the new server, or through periodic state updates. Triggered by an internal QoS monitor in its kernel, a client can issue a migration request to any server in the server list which the client receives in the connection establishment phase. Both the server and the client TCP layers should be changed and the server application layer should also be modified to allow for application-layer state snapshots and state recovery at the new server. It should be noted that [12] mentions briefly the limitation of the on-demand state update in the case of old server's crash or failure due to an attack. As an alternative, they propose to send state check-points to the client and use this client-stored state to recover the connection at the new server.

3 Overview of the Simulation Model

The main purpose for our simulation is to study the cost and benefit of the server roaming in several environment. The cost of server roaming is measured in term of the increased response time and the total

Proceedings of the 36th Annual Simulation Symposium (ANSS'03)
1080-241X/03 $17.00 © 2003 IEEE

number of connection migrations caused by the server roaming; while the benefit of the server roaming is measured in term of the improved response time while the server is being attacked.

We use a simple FTP client-server as the model for the simulated application. To start a FTP session, a client first sends a request for a file to the server. The server, then, sends the file back to the requesting client. To enable our roaming scheme, we add an authenticator to receive the initial client requests and to distribute roaming algorithm to the legitimate clients. With the knowledge of the roaming algorithm, any client can follow the active server by its own computation.

The migration model is simplified for the simulation. The active server is scheduled to roam among the servers in the pool in a certain time interval. We call this interval the *roaming interval*. It is one of the variable that we are interested to study. Once a client contacts the authenticator, it will receive the addresses of the servers, the pointer to the active server and the roaming interval. The client, then, uses this information to migrate from one server to another throughout the session.

We have built two types of the attacks. The first type of the attack deploys Constant Bit Rate(CBR) traffic generators as the attackers, while the second one utilizes a group of normal FTP clients to attack. Both of them are the link-based attacks. The protocols used in transport layers and the attacked resources, however, are different. The CBR generator uses UDP, while the normal FTP client deploys TCP. In addition, the UDP generator generates only one way traffic; therefore, the UDP attackers deplete only the bandwidth of the paths from the source to the destination. In contrast, the TCP generator generates two-way traffic; thus, the TCP attackers target the bandwidth in both directions. The path that is for the acknowledgment, however, have a much lighter attacking traffic than the path used to carry the data.

We split the experiment into four cases: (1) no attackers & no migration, (2) migration without being attacked, (3) being attacked without migration, and (4) being attacked with migration. The first case will give us the cost of a FTP transfer, while the second cases will give us the cost of the roaming. The cost incurred by the attacks will be shown in the third case. Lastly, the fourth case will draw the benefit of deploying roaming to mitigate the attack. The procedure how we build our simulation to support the roaming model and how we simulate these four cases will be described in the next section.

4 Experiment Design and Procedure

We use Network Simulator version 2(*NS-2*[1]) for our simulation. We modified the TCP agent module and added the socket layer to support the TCP migration. In addition, we created the multi-threaded FTP server and client modules to be used as our testbed application for the simulation. They works on top of the socket layer, where by the migration and TCP agent management take place. All of these components are combined to simulate a practical client and server model. A client is required to initiate communication with a server and then the server creates a thread to serve the request. The thread remains alive until the request is fulfilled completely. A simple scenario of a server serving four clients by using four threads can be viewed in the Figure 1.

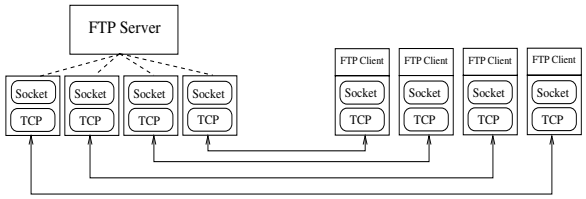


Figure 1. Logical connections of the FTP multi-thread server

According to the migration model mentioned in the previous section, each client needs to connect to the authenticator to get information about the servers in the pool and the roaming interval before starting a FTP session. We achieve this by deploying a module in the FTP client. Each FTP client requires to setup a TCP connection to the authenticator before receiving the servers' addresses and the roaming interval. Then, the client initiates another TCP connection with the active server and starts a transfer session. Later, if the migration schedule is reached before finishing the session, the client will use the socket to manage the migration. A socket manages a migration by first recording the current state (e.g. number of remaining bytes) of the connection and then simply dropping the current TCP agent, and starting a fresh TCP agent to connect to the current active server. The client will deploy the old connection state, such as the number of byte left, to control this TCP agent. The state of the previous TCP, however, is simply dropped, since the state along the path to the new server is likely to be different.

Our simulated network is composed of three servers, four attacker nodes, one legitimate client node and one

[1]NS2: http://www.isi.edu/nsnam/ns/

Proceedings of the 36th Annual Simulation Symposium (ANSS'03)
1080-241X/03 $17.00 © 2003 **IEEE**

authenticator. All FTP clients are originated from the same client node. Two routers are interconnected in between to simulate a wide area network environment. The topology of the simulated network is shown in Figure 2. Some configuration parameters, including the link rate and propagation delay for each link are also shown in the Figure.

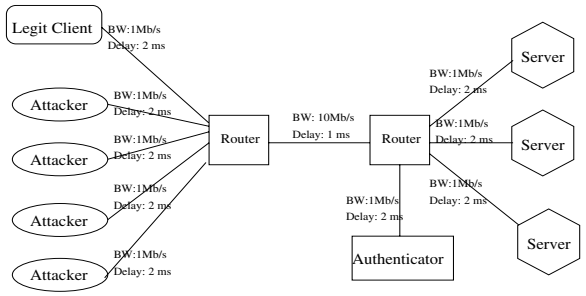


Figure 2. Topology of the simulated network

For each simulation, we take 20 runs with the total number of 100 independent FTP client requests for each run. The file size of each transfer is fixed to one megabits. The total load of the clients in the system is varied from 0.1 to 0.9. To control the load of each run, we use poison process to model the arrival process of the FTP clients. The inter-arrival time of the FTP-client (IT_{arr}) is computed by

$$IT_{Carr} = exp(At_{ftp}/Tot_{load}),$$

where At_{ftp} is the average total time for a file transfer, Tot_{load} is the given client total load, and $exp(x)$ is the exponential distribution with mean x.

We carefully consider the roaming intervals to study according to the setup that we have. The small interval will cause unnecessary migration, while the large interval will gain higher chance for being attacked. We select the roaming intervals that are big enough to allow at least one client to finish a transfer within at most one migration in the lightly load environment. According to this simulation setup, a client took about 1.1 seconds to finish a transfer. Therefore, the roaming intervals were varied among 2, 4, 6, 8 and 10 seconds.

We simplify the attack model by assuming that the attackers would attack only one active server. They do not know where the other servers are. The attack load are distributed equally to all four attack nodes. In the case of UDP attack, we setup one CBR traffic generator on each attacker node. The fixed size 1000 bit for a packet is used. The rate of attack depends on the given total attack load (Tot_{att}) for each run. We calculate the rate R_{att} of each CBR traffic generator from the formula

$$R_{att} = (Tot_{att} * BW_{blink})/(1000 * N_{att}),$$

where BW_{blink} is the bandwidth of the bottle neck link and N_{att} is the number of attackers (CBR sources). We vary the attack loads among 0.6, 0.8, 1.0, 1.2, 1.4 and 1.6 of the bottle neck link bandwidth (1 Mb/s in this configuration). In the case of TCP attack, the attack load is computed in term of arrival rate of the attackers, similar to the IT_{Carr} formula above. The load of the attacker, however, are distributed among the attacker nodes. We vary the total attack load among 0.2, 0.4, 0.6, 0.8 and 1.0 of the bottle link bandwidth. We use the range of the attack loads lower than those used in the UDP attacks because the TCP seems to be more aggressive and sensitive to the available bandwidth than does the UDP. The high rate of TCP attack may cause the system fluctuate too drastically to study. Moreover, the UDP attack will compete with the lighter legitimate traffic along the acknowledgment (ACK) path; while the TCP attack will deplete resources along the data path directly. The UDP attack will deplete resources only along the path from client to server; while the TCP attack, deploying the same FTP request-response, uses up most of resources along the path from the server to the client. The summary of all the values used in each simulation are described in the Table 1.

Case of Simulation	Parameter Settings		
	Client Load	Migration Interval (seconds)	Attack Load (of bottle link bandwidth)
1. No Mig & No Att	0.1-0.9	n/a	n/a
2. Mig & No Att	0.1-0.9	2,4,6,8,10	n/a
3. No Mig & Att	0.1-0.9	n/a	TCP att: 0.2,0.4,0.6,0.8,1.0 UDP att: 0.6,0.8,1.0,1.2,1.4,1.6
4. Mig & Att	0.1-0.9	2,4,6,8,10	TCP att: 0.2,0.4,0.6,0.8,1.0 UDP att: 0.6,0.8,1.0,1.2,1.4,1.6

Table 1. Parameter settings for the simulation

5 Result/Analysis

In this section, the results of all simulations will be described with our analysis. First we study the cost of the roaming from the comparison of the results from case (1) and (2) in Table 1. Later, we shows the benefit of the roaming by first introducing the cost incurred by the attacks and then comparing the results of case (3) and (4) from the Table.

5.1 The Cost of the Roaming

We measure the cost of the roaming in term of the increased response time and the average number of migrations for a transfer. They are shown in Figure 3 and Figure 4 respectively. According to these results, the roaming interval is a main factor for the roaming cost. As the roaming interval decreases, the average response

Proceedings of the 36th Annual Simulation Symposium (ANSS'03)
1080-241X/03 $17.00 © 2003 **IEEE**

time and number of migrations for a transfer increase. This relation can be explained that as the roaming interval decreases, the probability that a transfer will finish without any migration diminishes. This will cause average number of migrations per a transfer increases and lead to a longer time to finish a transfer, due to the delay introduced by starting fresh connections. In addition, the cost, which can be seen as the vertical distance between the base case (no migration) and the case of interest in the Figures, in both terms will get higher as the total load in the system grows. The total load of FTP clients reflects how many FTP connections are at the same time. When the total load increases, the competition among the connections for the limited bandwidth of the bottleneck link get more intense, leading to the longer average transfer time. This gains the chance for a connection to be migrated before finishing a transfer. As a result, with the effects of the bandwidth competition and the migration, the cost of migration will increase exponentially as the total load of the system increases.

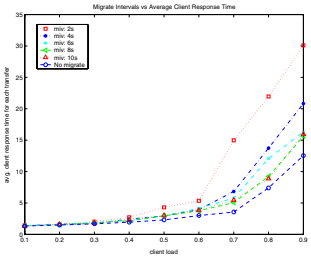


Figure 3. Cost of the Roaming: Increased Response Time

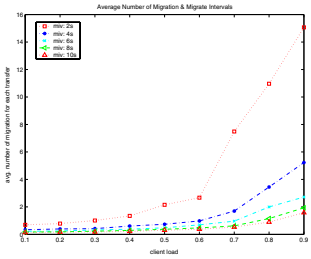
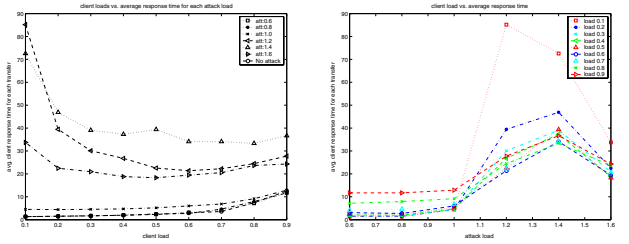


Figure 4. Cost of the Roaming: Number of Migrations

5.2 The Loss by the Attack

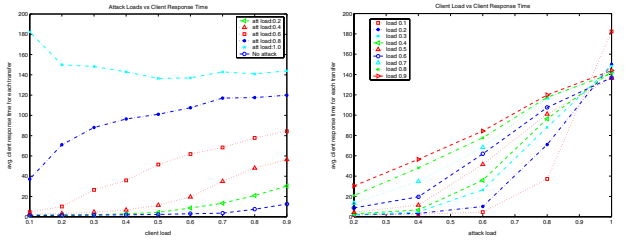
The costs introduced by UDP and TCP attacks are shown in Figure 5 and Figure 6 respectively. In the case of the UDP attack, only the cases that the total attack load higher than 1.0 affect the average transfer time of the FTP client. These high attack loads (attack loads of 1.2, 1.4 and 1.6), however, show an interesting and doubtful results. Two relations drawn from the results in particular are our interest. First, when the client load is in the range of 0.1–0.3, the average response time is getting better as the load of the client increases. Second, increasing the attack load from 1.2 to 1.4 brings the average response time up; while increasing the attack load from 1.4 to 1.6 improves the average response time. In the case of the TCP attack, all attack loads affect the average response time of the FTP clients. The effect gains significance as the attack load increases. In general, the results of the TCP attacks seem reasonable; however, the result at the 1.0 TCP attack load is skeptical since the result in the case of the light load of client shows the worse average response time comparing to the results from the the higher client load simulations. We validate



(a) Effect of total load to each attack load case

(b) Effect of attack load to each client load case

Figure 5. Cost caused from the UDP attacks



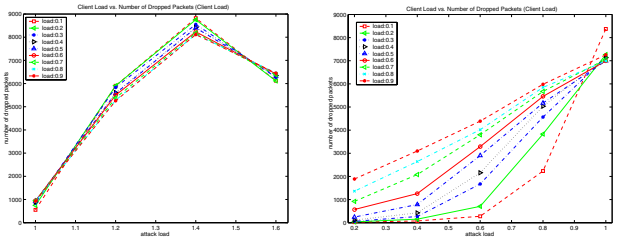
(a) Attack effect (total attack load)

(b) Attack effect (total client load)

Figure 6. Effect of the TCP attacks

the results by monitoring and tracing the simulations. We first look at the number of dropped packets for each case. We explain the worst average response time at the light load of clients and high load of attack by drawing the number of dropped packets shown in Figure 7. According to the graphs, when the attack load is high (1.2 for UDP attack and 1.0 for TCP attack), the number of the dropped packets in the cases of 0.1 client load is the highest comparing to those in other higher load cases. This can be explained that the light load of traffic from the legitimate clients allure their TCP agents to inject packets to the servers faster than the rate they should be. Therefore, their packets are dropped a lot more than those in the higher client load cases, where by the agents rather perceive congestion along the path. For another questionable relation, shown in results from UDP high load attacks, can be explained by deploying the proportion of the number of packet drops to arrivals from legitimate clients. As shown in Figure 8, this proportion in the case of the 1.4 total attack load is the highest comparing to those in all other attack load cases; the case of 1.6 total attack load has the smallest proportion in all three high load attack

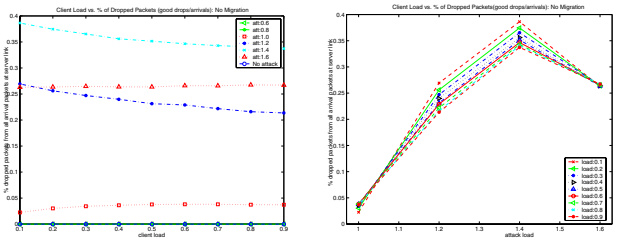
Proceedings of the 36th Annual Simulation Symposium (ANSS'03)
1080-241X/03 $17.00 © 2003 **IEEE**



(a) Number of dropped packets of the UDP attack (b) Number of dropped packets of the TCP attack

Figure 7. Number of dropped packets categorized by client load

cases. Our conjecture of this phenomenal is that the TCP state has changed when the total attack load is higher than 1.4. In other word, when the total attack load is higher than 1.4, the TCP agents for the legitimate FTP clients have perceived the congestion or the tense of the path to the server and adjust themselves by shrinking their sending windows. This will, in return, let them complete delivering more packets to the server and receive better response times than do the agents in the cases of the attack load 1.2 and 1.4.

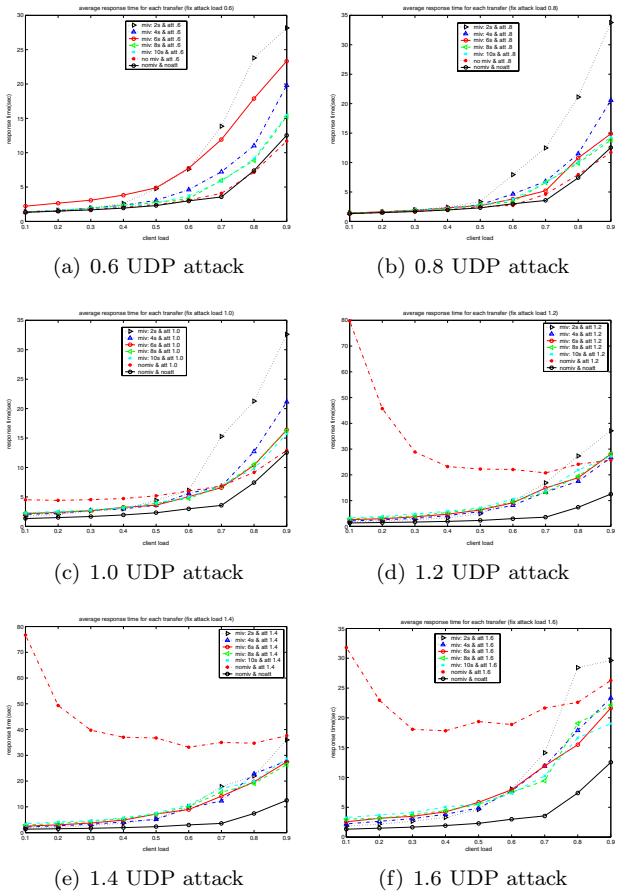


(a) Categorized by attack load (b) Categorized by client load

Figure 8. Proportion of packet drops to arrivals

5.3 Benefit of the Server Roaming

The benefits of applying the server roaming to the UDP and TCP attack cases are shown in Figure 9 and Figure 10 respectively. In the case of UDP attack, the average response time is reduced significantly when the roaming server is applied in the case of high attack load. Figures 9(d)– 9(f) show the significant improvement, even in the case of high migration cost of two-second roaming interval. In the case of the TCP attack, the roaming improves the average response time in all simulated cases. The reason is simply that migrating connections from the attacked server to a non-attacked server increase opportunity for the transfers to be completed a lot quicker than leaving them with the stalled server.



(a) 0.6 UDP attack (b) 0.8 UDP attack

(c) 1.0 UDP attack (d) 1.2 UDP attack

(e) 1.4 UDP attack (f) 1.6 UDP attack

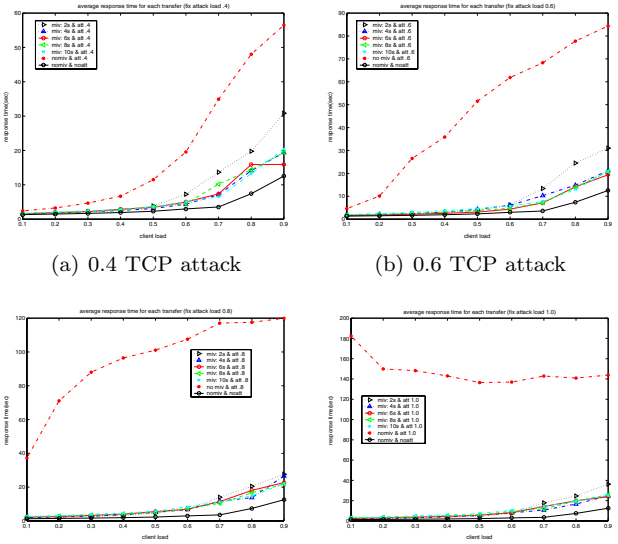
Figure 9. Benefit of Roaming: UDP attacks

6 Conclusion and Future Work

Our goal in the NETSEC project at University of Pittsburgh is to design and implement a framework for mitigating the effects of both types of DoS attacks, node-based and link-based. In this paper, we study the defense strategy of the *server roaming* via simulation. We define the server roaming as a framework to mitigate the effects of DoS attacks. The active server changes its location among a pool of servers to defend against unpredictable and likely undetectable attacks. Only legitimate clients will be able to follow the server as it roams. To study the roaming behavior of the clients in NS2, we add the socket module to manage connection migration. In addition, we create FTP server and client modules to simulate request-reply type of network applications. They are used on top of the socket layer.

Our main purpose for the simulation is to study the cost and benefit of the server roaming to mitigate DoS attacks. We split experiments into four cases:

Proceedings of the 36th Annual Simulation Symposium (ANSS'03)
1080-241X/03 $17.00 © 2003 **IEEE**



(a) 0.4 TCP attack (b) 0.6 TCP attack

(c) 0.8 TCP attack (d) 1.0 TCP attack

Figure 10. Benefit of Roaming: TCP attacks

no-roaming and no-attack, roaming and no-attack, no-roaming and attack, roaming and attack. The first case is simulated to find the basic cost of a FTP transfer. The case of roaming and no-attack is experimented to find the extra cost caused by the roaming. The third case, no-roaming and attack environment, is simulated to discover the loss caused by the attack. Last, the case of roaming and attack is simulated to find the benefit of applying the server roaming to the attack environment. The results of the simulations are promising. The benefit of the server roaming outweighs the cost of the roaming and the loss caused by the attacks. In addition to the simulations, the implementation of the server roaming strategy is also being experimented in our networking lab. We have designed a more complex model of the secure roaming, which protects information being passed among clients, servers and authenticator.

Our next step for the simulation is to develop better and more practical attack and roaming models. Some heuristic techniques are needed. We plan to seek for the attack patterns in the real world and draw experiment designs to improve our models. Additionally, we will deploy the other well-known network applications, such as a web server and domain name servers, as attacked services to explore different behaviors of the attackers and the roaming strategies to fight against them.

References

[1] S. Axelsson. Intrusion detection systems: A survey and taxonomy. Technical report, Depart. of Computer Engineering, Chalmers University, 2000.

[2] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing. In *RFC 2827*, 2001.

[3] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. In *Journal of the ACM*, volume 33, pages 792–807, 1986.

[4] O. Goldreich, L. Levin, and N. Nisan. On constructing 1-1 one-way functions. In *Proceedings of the Electronic Colloquium on Computational Complexity*, 1995.

[5] J. Jones. Distributed denial of service attacks: Defenses, a special publication. Technical report, Global Integrity, 2000.

[6] F. Lau, S. H. Rubin, M. H. Smith, and L. Trajovic. Distributed denial of service attacks. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 2275–2280, Nashville, TN, USA, Oct. 2000.

[7] J. Mirkovic, J. Martin, and P. Reiher. A taxonomy of ddos attacks and ddos defense mechanisms. Technical report, Computer Science Department, University of California, Los Angeles, 2002.

[8] R. Rivest. The md5 message-digest algorithm. In *RFC 1321*, 1992.

[9] C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni. Analysis of a denial of service attack on TCP. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 208–223. IEEE Computer Society, IEEE Computer Society Press, May 1997.

[10] A. C. Snoeren, D. G. Andersen, and H. Balakrishnan. Fine-grained failover using connection migration. In *Proc. of 3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, 2001.

[11] A. C. Snoeren, H. Balakrishnan, and M. F. Kaashoek. The migrate approach to internet mobility. In *Proc. of the Oxygen Student Workshop, July 2001*, 2001.

[12] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode. Migratory tcp: Highly available internet services using connection migration. Technical report, Rutgers University, 2001.

[13] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode. Migratory tcp: Connection migration for service continuity in the internet. In *Proceedings of The 22nd International Conference on Distributed Computing Systems (ICDCS) 2002*, 2002.

Proceedings of the 36th Annual Simulation Symposium (ANSS'03)
1080-241X/03 $17.00 © 2003 **IEEE**