Available online at www.sciencedirect.com
SCIENCE DIRECT®
The Journal of Systems and Software 73 (2004) 15–29
The Journal of Systems and Software
www.elsevier.com/locate/jss

Design and analysis of a replicated elusive server scheme for mitigating denial of service attacks ☆

Chatree Sangpachatanaruk [a], Sherif M. Khattab [b], Taieb Znati [a,b,*], Rami Melhem [b], Daniel Mossé [b]

[a] Department of Information Science and Telecommunication, University of Pittsburgh, Pittsburgh, PA 15260, USA
[b] Department of Computer Science and Telecommunication, University of Pittsburgh, Sennott Square, Pittsburgh, PA 15260, USA

Received 11 July 2003; received in revised form 2 September 2003; accepted 3 September 2003
Available online 25 December 2003

Abstract

The paper proposes a scheme, referred to as *proactive server roaming*, to mitigate the effects of denial of service (DoS) attacks. The scheme is based on the concept of "replicated elusive service", which through server roaming, causes the service to physically migrate from one physical location to another. Furthermore, the proactiveness of the scheme makes it difficult for attackers to guess *when* or *where* servers roam. The combined effect of elusive service replication and proactive roaming makes the scheme resilient to DoS attacks, thereby ensuring a high-level of quality of service. The paper describes the basic components of the scheme and discusses a simulation study to assess the performance of the scheme for different types of DoS attacks. The details of the NS2-based design and implementation of the server roaming strategy to mitigate the DoS attacks are provided, along with a thorough discussion and analysis of the simulation results.

Keywords: Denial of service attacks; Security; Elusive servers; Replication; Migration; Simulation; Performance analysis

1. Introduction

The widespread need and ability to connect machines across the Internet, in a world where intelligent objects rather than documents are exchanged, has caused the network to be more vulnerable to intrusions and has facilitated break-ins of a variety of types. Most of the methods currently available to deal with network vulnerability to abuse and attacks are either inadequate, inefficient or overly restrictive. Compounding the problem is the need to maintain an acceptable level of quality of service (QoS).

The proposed research assumes the existence of intrusion detection mechanisms and aims at developing new and potentially revolutionary approaches for the development of scalable and efficient tools for network service protection. We will consider two types of faults: *benign malfunctions* and *malicious intrusions*. The former can be caused by a faulty, yet legitimate client that loses control over its behavior, while the latter occurs with the intent to cause damage, such as *denial of service* (DoS). Both types of faults can severely affect the performance of the network and compromise the integrity and security of its services.

We consider a network where servers provide probabilistic QoS guarantees to clients through a *contract protocol*. These servers may be subject to two forms of faults, namely protocol breach and contract violation. Both types of faults can be either benign or malicious. In order to protect the servers and the network, we propose

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

* Corresponding author. Address: Department of Computer Science and Telecommunication, University of Pittsburgh, Sennott Square, Pittsburgh, PA 15260, USA. Tel.: +1-412-624-8417; fax: +1-412-624-8050.

E-mail addresses: chatree@cs.pitt.edu (C. Sangpachatanaruk), skhattab@cs.pitt.edu (S.M. Khattab), znati@cs.pitt.edu (T. Znati), melhem@cs.pitt.edu (R. Melhem), mosse@cs.pitt.edu (D. Mossé).

0164-1212/$ - see front matter © 2003 Published by Elsevier Inc.
doi:10.1016/j.jss.2003.09.012

a *fault avoidance* technique based on the concept of *replicated elusive servers*. The concept of replicated elusive service can be implemented using a roaming address for fixed servers in a wireline network, or switching to a different frequency for mobile servers in a wireless network. Replication is coordinated through group communication supported by an underlying multi-cast mechanism.

In this paper, we show our study of the replicated elusive servers using proactive roaming technique to mitigate these types of faults, more specifically the faults caused by DoS attacks. 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 of the authentication process caused by many illegitimate requests, however, is a DoS attack in itself.

Our goal is to design and implement a practical framework to mitigate the effects of both types of DoS attacks and not to scarify too much resources. We begin by studying the cost and benefit of our proposed technique, the *proactive server roaming*, via a simulation and a real prototype implementation. Our simulation in the previous work (Sangpachatanaruk et al., 2003) showed that the proactive server roaming has potential to improve the DoS defensive strategy. In this paper, we extend our work to cover the scheme in a larger simulated network, and show our design, including the algorithms used in the main components, and analysis of the system in more detail.

The organization of the paper is the following. First, the related works are presented in Section 2. Second, the system model is described in Section 3. Third, the sever roaming background is explained in Section 4. Fourth, the overview of our simulation model is described in Section 5. Fifth, the experiment design and procedures are explained in Section 6. Seventh, we show the results and analysis in Section 7. Last, the conclusion and the future work are presented in Section 8.

2. Related works

2.1. Denial of service attack defenses

In the following, we present some of the current defenses and mitigation techniques for the two categories of DoS attacks, namely node-based and link-based.

2.1.1. Node-based attack defenses

In node-based attacks, the imbalance between the amount of resources needed to make a request and the amount of resources needed to satisfy this request creates a cost-effective DoS attack opportunity. By sending a cheap illegitimate request, an attacker can hold a quantifiable amount of resources. Launching a large amount of such requests can exhaust the finite server resources. Some mechanisms were developed to defend against such attacks. *Resource pricing* (Mankins et al., 2001) assigns a dynamic cost to each resource based on the system load. This cost has to be dispensed from the requesting client before the resource is allocated. *Client puzzles* (Juels and Brainard, 1999) is a special case of such a pricing mechanism, where the client has to solve a cryptographic problem with varying complexity before the server allocates resources to the request and starts servicing it. Both of these techniques can be implemented transparently to the server application, either as a gateway or as a middleware layer. The main disadvantage is the requirement of special client software.

The lack of resource management facilities in current widely-deployed operating systems is another cause of the node-based DoS attacks. The ability to account for resources allocated to each client according to negotiated contracts, detect contract violations and recover misused resources is a design target of the Scout operating system (Spatscheck and Petersen, 1999). Using the techniques employed in *quality of service regulation* (Garg and Reddy, 2002), such as rate-based and window-based regulation, can provide a means for resource management, leading to a graceful performance degradation under DoS attacks. In the latter mechanism, a *bastion host* is used to keep track of server resource usages at the network level and regulate all traffic passing through it. All the above techniques fall into the DoS mitigation track.

2.1.2. Link-based attack defenses

While defenses against node-based DoS attacks can create a shielded boundary protecting resources behind it, a link-based attack can target the uplink connecting this boundary to the high-bandwidth network backbone. Once under attack, two reactions need to be done. A system may deploy either appropriate packet filters at the network egress points with the minimum false positives and false negatives, or identify attack sources in

order to stop them. Two dangerous link-based attacks: spoofed source address and reflector attacks (Paxson, 2001), however, are very difficult to be filtered and identified. Although reflector attack packets have valid source addresses, the huge amount of participating reflectors create the difficulty of tracing back to the sources of the original attack packets which are being reflected, and the fact that most attack packets are targeted to vital network services make this attack type one of the most challenging DoS attacks.

Ingress packet filtering (Ferguson and Senie, 2001) is proposed to defend against some DoS attacks. The filter is required to placed on the boundary of every single sub-network. *Route-based distributed packet filtering* (*DPF*) (Park and Lee, 2001) requires the cooperation of a small subset of intermediate routers strategically placed within the network (one possibility is placing the filtering routers on the vertex cover of the network graph). A router R drops a packet if there is no path in the graph from the packet's source to the packet's destination. This mechanism does not block all spoofed packets. However, upon receiving an attacking spoofed packet, the attacked host can limit the attack source within a small number of sub-networks. The main difficulty is how to collect and maintain this kind of routing information within the participating routers. In the *secure overlay services* (*SOS*) architecture (Keromytis et al., 2002), only packets coming from a small number of nodes, called servlets, are assumed to be legitimate. These servlets act as proxies for legitimate client traffic which can reach the servlets through secure hash-based routing inside an overlay network. This helps keeping the address of these servlets secret. In order to gain access to the overlay network, a client has to authenticate itself with one of the replicated access points (SOAPs). In *VIPnets* (Brustoloni, 2002), legitimate traffic is assumed to be the traffic coming from networks implementing the VIPnet service. All other traffics are considered as low-priority and can be dropped in the case of an attack. The architecture relies on the provision of QoS mechanisms, such as *diffserv* (Blake et al., 1998), in intermediate routers. *Mutable services* is a framework to allow for relocating service front-ends and informing legitimate clients of the new location (Ivan et al., 2002). All these techniques are examples of classification-based DoS defenses.

Fast and accurate tracing of an attack to its sources can help in both deterring attackers and identifying attacking nodes to be able to stop them from sending illegitimate packets. *IP traceback* (Savage et al., 2000) utilizes path information injected by intermediate routers as either separate packets or as fields inside packets. *Deterministic packet marking* requires each router to contribute into the packet path information field, while in *non-deterministic packet marking* (Park and Lee, 2000; Song and Perrig, 2001) a router makes this contribution with a probability p. An attacked host can reconstruct the attack path from the large number of received attacking packets. In *hash-based IP traceback* (Snoeren, 2001), even a single packet can be traced to its actual source allowing for the tracing back of node-based and original reflector attack packets as well. These mechanisms are instances of the attack tracking DoS defenses.

IP hopping (Jones, 2000) is the technique that a server changes its IP address after detecting it is under attack. In order for the clients to reach the server, DNS servers have to be updated with the new IP address. In this mechanism, the server does not change its location. All packets destined to the old IP address can be filtered at the network perimeter by a firewall. To avoid continuous server reconfiguration, a NAT (Network Address Translation) gateway can be inserted at the network entry point. IP hopping can be used both reactively and proactively. One disadvantage of this mechanism is that during the period of time in which the DNS servers are reflecting the change in the IP address, all legitimate client requests are also filtered out. IP hopping is an example of DoS mitigation defenses. Table 1 summarizes the previous classification.

Table 1
DoS defense classification

Attack type	Defense technique		
	Mitigation	Classification-based	Attack-tracking
Node-based	Resource pricing Client puzzles QoS regulation Resource management	None	Hash-based IP traceback
Link-based	IP hopping	DPF SOS Mutable services VIPnet	Packet marking Hash-based IP traceback

2.2. TCP-Migration

Two well-known TCP-connection migration mechanisms are the *TCP-Migrate* (Snoeren et al., 2001b), developed at MIT, and the *Migratory-TCP* (Sultan et al., 2002), 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 (Snoeren et al., 2001a), 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 (Sultan et al., 2001) 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. System model

3.1. System components

Our system consists of the following components:

Server pool and firewalls: A pool of N homogeneous servers physically deployed as in Fig. 1. In this configuration, each server is connected to the outside network through a firewall. Geographically dispersing the servers and/or deploying them over a *resilient overlay network* (*RON*) (Andersen et al., 2001) provide for better path independence among the servers. Servers in the server pool, together with the firewalls, employ techniques of *threshold cryptography* (Desmedt, 1994), *proactive secret sharing* (Herzberg et al., 1995), and *secure group communication* (Wong et al., 1998). In order to achieve secure group communication among them, servers in the server pool share a secret group key. This key is shared among all the servers without any single server being able to recover the whole key. We assume loose clock synchronization among servers (i.e., the clock shift is bounded by a constant, δ) and tight clock synchronization between each server and its firewall.

Clients: Two classes of clients are assumed: (1) *legitimate clients*, which subscribe to the service and can establish secure channels with any servers. By the term "secure channel", we mean the provision of confidentiality, authentication, integrity, and message freshness (immunity to message replay attacks) (Perrig et al., 2001). For each legitimate client we assume a QoS contract with the service. According to this contract, the current server allocates a certain amount of resources to each legitimate client request; (2) *illegitimate clients* represent malicious attackers trying to degrade the service responsiveness.

The service: Our service is a generic TCP-based service. The service can be replicated. Only one server is active and providing the service at any point of time. Depending on the granularity of clock synchronization, the active interval of one server can be interleaved with the active interval of another one while the service is being roamed. It should be noted that it is easy to extend this to a scheme in which k servers are active to achieve load balancing for example.

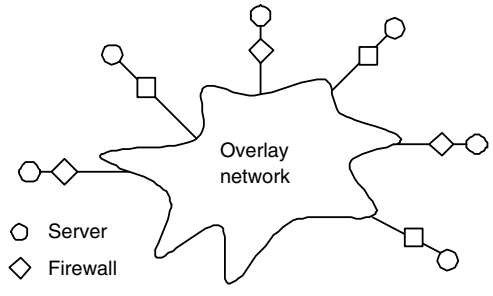

Fig. 1. Server pool and firewalls.

3.2. Threat model

The malicious adversary we are defending against is interested in launching a denial of service attack against our services. The adversary can be passive, active, or both. A passive adversary can eavesdrop on server-to-server and/or client-to-server communication and apply cryptanalysis techniques. An active adversary can impersonate as a legitimate client, obtain access to a legitimate client, and inject malicious messages to servers and/or to legitimate clients or tamper the messages. The result of some or all of these actions is a denial of service attack, node-based or link-based, deployed from the outside network or from inside a network on which a server resides. Our adversary can launch attack packets at a very high but finite rate.

4. Server roaming

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 (Axelsson, 2000).

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* (Jones, 2000). 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.

Vulnerability to node-based attacks comes from the fact that the server is opening its ports and accepting connections. Replication by itself, while being able to reduce the effect of link-based DoS attacks, fails to provide for protection against node-based attacks. This is because all the replicas are active and accepting connections at the same time. Our server roaming scheme decreases the amount of time a server is accepting connections and the number of open ports. It also proactively flushes the service state each time the server roams. This allows for better resilience to node-based attacks.

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 active server. 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.

4.1. Secure proactive roaming

To be able to know the active server location, a client need to know at least the server address and the server active time. 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 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.

Our scheme utilizes the idea of one-way hash functions, such as SHA-1 and MD5 (Goldreich et al., 1995,

1986; Rivest, 1992). 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. More information of the computation and algorithm is described in Khattab et al. (2003).

5. Overview of the simulation model

The main purpose for our simulation is to study the cost and benefit of the server roaming. The cost of server roaming is measured in term of the increased response time and the total 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 information 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 by deploying a fix migration interval for a 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, has a much lighter attacking traffic than the path used to carry the data.

We split the experiment into four cases: (1) no attackers and 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.

6. 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 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 Fig. 2.

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 is required 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

[1] NS-2: http://www.isi.edu/nsnam/ns/.

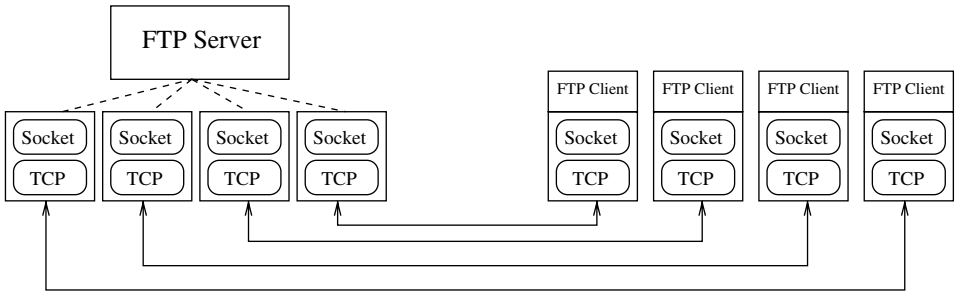

Fig. 2. Logical connections of the FTP multi-thread server.

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. The algorithms we use are described in the procedures below.

FtpClient::procedure *FTP_Request*(*Auth, File_Size*)
1: *auth_socket* = *get_connection*(*Auth*)
2: *secret* = *auth_socket* → *get_secret_tracking*( )
3: *a_server* = *secret* → *get-active-server*( )
4: *auth_socket* → *close*( )
5: *m_socket* = *get_connection*(*a-server*)
6: *m_socket* → *request-file*(*File_Size*)
7: *set_migrate_timer* = *get_next_migrate*(*secret* → *migrate_interval*)

FtpClient::procedure *Download_complete*( )
cancel_migrate_timer( )
collect_statistics( )

FtpClient::procedure *Timeout*( )
`if` *Download_File_Not_Done* `then`
m_socket → *migrate*(*secret* → *get_next_server*)
`end if`

MSocket::procedure *Migrate*(*new_server*)
1: *byte_remain* = *byte_requested* − *byte_received*
2: *byte_requested* = *byte_remain*
3: *byte_received* = 0
4: *old_tcp_agent* = *current_tcp_agent*
5: *old_tcp_agent* → *close*( )
6: *current_tcp_agent* = *new TCP_Agent*(*new_server*)
7: *current_tcp_agent* → *connect*( )
8: *current_tcp_agent* → *request*(*byte_remain*)

We experiment the simulation on two topologies, namely 10- and 40-node networks. We start the simulation from the first topology to study the behaviors of all players, i.e. clients, attackers, servers, TCP agents, and etc., in detail; while the second topology allows us to study all players in a more practical environment. The parameter settings and the main simulation procedures for these two schemes are described in the next subsections.

6.1. Simulation design for a small network

First, a 10-node network, as shown in the Fig. 3, is composed of three servers, one client, four attackers and two router nodes. All FTP clients are originated from the same client node. Some configuration parameters, including the link rate and propagation delay for each link are also shown in the figure. For each simulation with the 10-node network, 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 1 Mbps. 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 the poison process to model the arrival of the FTP requests. The inter-arrival time of the FTP-client (IT_{Carr}) 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

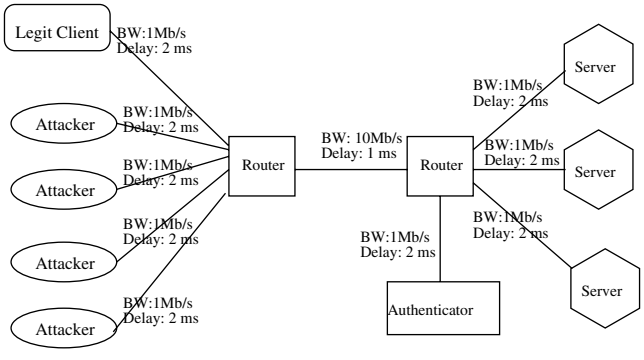

Fig. 3. Topology of the simulated network.

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 setups, a client takes about 1.1 s in average to finish a transfer. Therefore, the roaming intervals are varied among 2, 4, 6, 8 and 10 s.

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 is distributed uniformly to all four attack nodes. For the 10-node network we experiment two types of attacks, UDP and TCP attacks. 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 ($\mathrm{Tot}_{\mathrm{att}}$) for each run. We calculate the rate R_{att} of each CBR traffic generator from the formula

$$R_{\mathrm{att}} = (\mathrm{Tot}_{\mathrm{att}} * \mathrm{BW}_{\mathrm{blink}})/(1000 * N_{\mathrm{att}}),$$

where $\mathrm{BW}_{\mathrm{blink}}$ 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 Mbps 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 $\mathrm{IT}_{\mathrm{Carr}}$ 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. In other words, 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 Table 2.

6.2. Simulation design for a large network

A 40-node network, shown in the Fig. 4 is composed of three servers, labeled node 12, 14 and 23 and one authenticator, labeled node 0, and 35 client nodes. The links are assigned as recommended in Ratnassamy et al. (2002) with the following bandwidths and delays: 1 Mbps bandwidth and 10 ms delay for intra-stub links and 10 Mbps bandwidth and 1 ms delay for all inter-stub links, i.e. links connecting to node 0 and 1, and the 33–36 and 9–16 links.

We design the experiment for this network different from the the 10-node network simulation. Basically, the

Table 2
Parameter settings for the simulation

Case of simulation	Parameter settings		
	Client load	Migration interval (s)	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

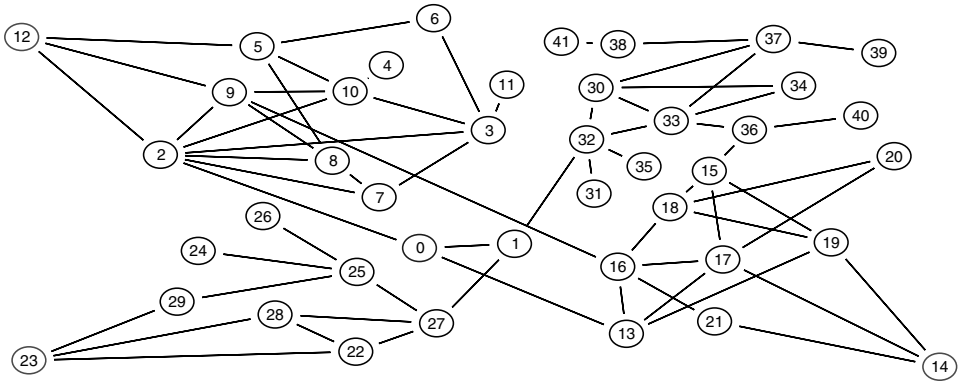

Fig. 4. Topology of the simulated network.

network is larger and we need to introduce the randomness to the locations of the senders of the legitimate and illegitimate requests. In addition, the servers are places at the different stub domains as defined in our system models in Section 3 to simulate an effective model for the roaming scheme. To simplify the analysis, we carefully place the servers where they all have the same access bandwidth (i.e. 3 Mbps) and where they all have some independent paths from clients to access. We take five runs with a total clients requests generated in the first four hundred seconds for each run. After the four hundredth second, we stop generating the request and let the simulation run until all transfers are completed. We schedule the requests using the following procedure.

Procedure *Schedule_FTP_Req_Forty_Node_Network* (*client_load*)

```
1: avg_it_arrival_time = At_ftp/Tot_clientload
2: c_start = c_init_start_time; i = 0
3: while c_start ⩽ 400.0 do
4:     cit = exp(avg_it_arrival_time)
5:     c_start = c_start + cit
6:     client(i).attach_node(uniform(1,35))
7:     start client(i) at time c-start
8:     increment i
9: end while
```

For the roaming interval, since a client takes about 1.5 s in average to finish a transfer, the roaming intervals were varied among 4, 8, 10, and 20 s for the 40-node network. For the attack model, we use the same simple model of attacking only one server as does in the small network simulation. The only type of the attack we run, however, is only the TCP.

The attackers' nodes are picked uniformly among client nodes in the same way we pick the client node for a request. 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 use the total attack load of 1.0 of the total access bandwidth, i.e. 3 Mbps. Moreover, the simulation starts introducing the attackers at the hundredth second and stop the attack at the three hundredth second. The summary of all the values used in each simulation are described in Table 3.

Table 3
Parameter settings for the simulation

Case of simulation	Parameter settings		
	Client load	Migration interval (s)	Attack load
1. No Mig & No Att	0.1–0.9	n/a	n/a
2. Mig & No Att	0.1–0.9	4, 8, 10, 20	n/a
3. No Mig & Att	0.1–0.9	n/a	1.0
4. Mig & Att	0.1–0.9	4, 8, 10, 20	1.0

7. Results/analysis

In this section, the results of all simulations will be described with our analysis. The result and analysis of the simulation of a 10-node network and the 40-node network will be described respectively.

7.1. Result/analysis of the 10-node network simulation

We study the cost of the roaming from the comparison of the results from case (1) and (2) in Table 2. 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.

7.1.1. 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 Figs. 5 and 6 respectively. According to these results, the roaming interval is the main factor for the roaming cost. As the roaming interval decreases, the average response time and number of migrations for a transfer increase. This relation can be explained that as the roaming interval

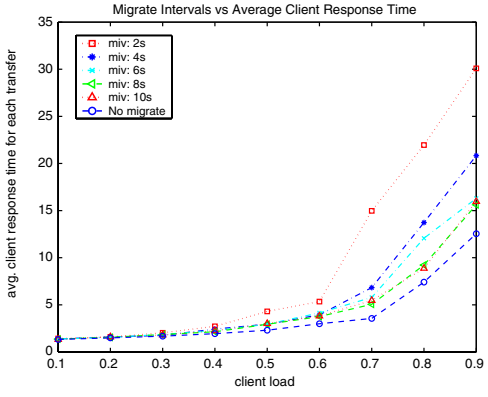

Fig. 5. Cost of the roaming: increased response time.

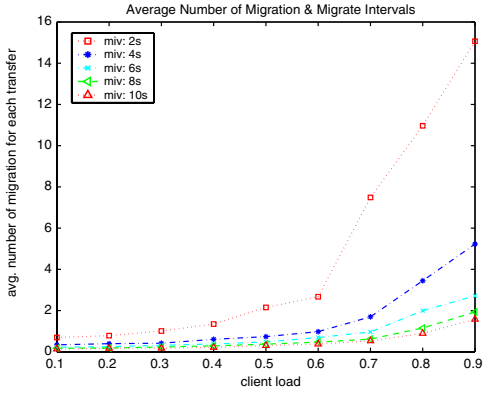

Fig. 6. Cost of the roaming: number of migrations.

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.

7.1.2. The loss by the attack

The costs introduced by UDP and TCP attacks are shown in Figs. 7 and 8 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 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

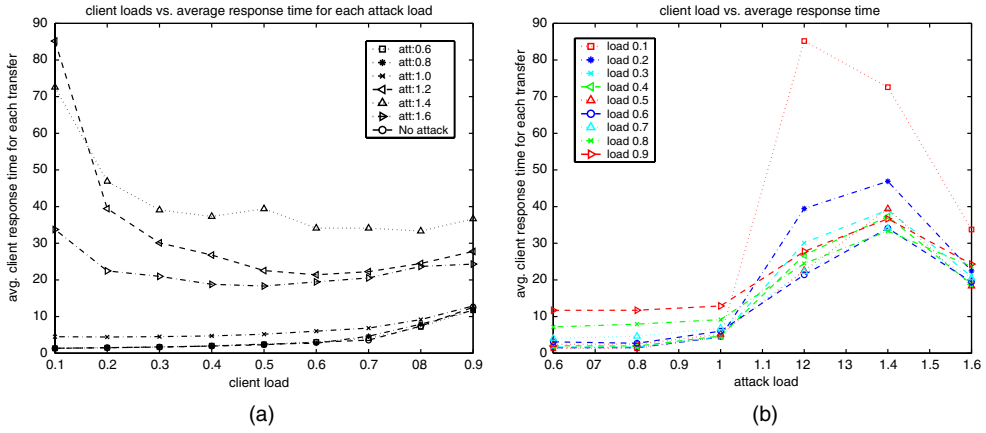

Fig. 7. Cost caused from the UDP attacks: (a) effect of total load to each attack load case, (b) effect of attack load to each client load case.

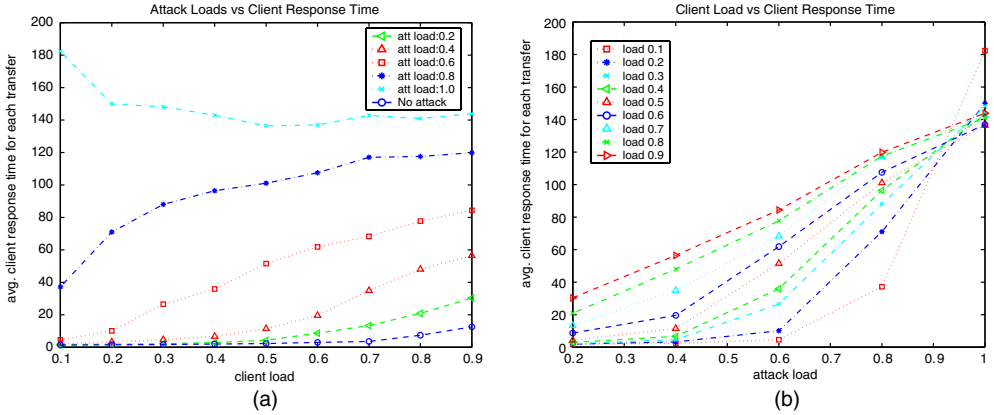

Fig. 8. Effect of the TCP attacks: (a) attack effect (total attack load), (b) attack effect (total client load).

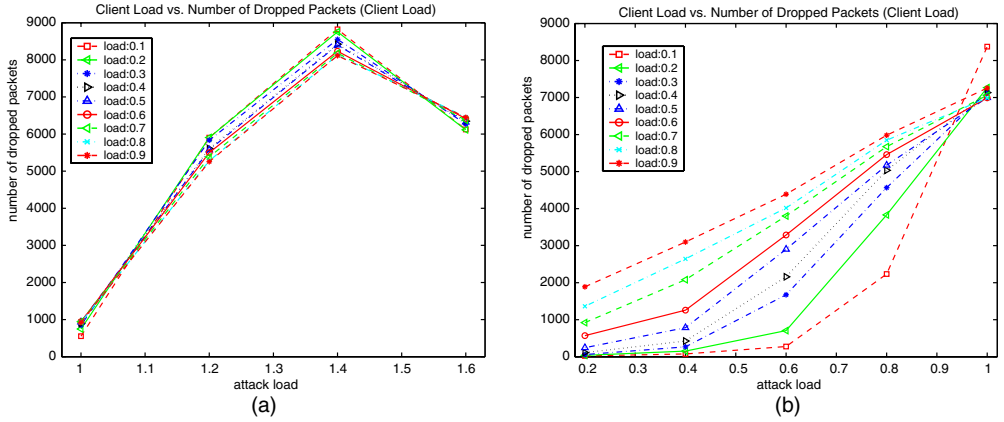

Fig. 9. Number of dropped packets categorized by client load: (a) number of dropped packets of the UDP attack, (b) number of dropped packets of the TCP attack.

response time at the light load of clients and high load of attack by drawing the number of dropped packets shown in Fig. 9. 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, we can explain by deploying the proportion of the number of packet drops to arrivals from legitimate clients. As shown in Fig. 10, 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 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 along 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.

7.1.3. *Benefit of the server roaming*

The benefits of applying the server roaming to the UDP and TCP attack cases are shown in Figs. 11 and 12 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. Fig. 11(a) and (b) show the significant improvement, even in the case of high migration cost of 2-s roaming interval. In the case of the TCP attack, the roaming improves the average response time in all simulated cases. The reason

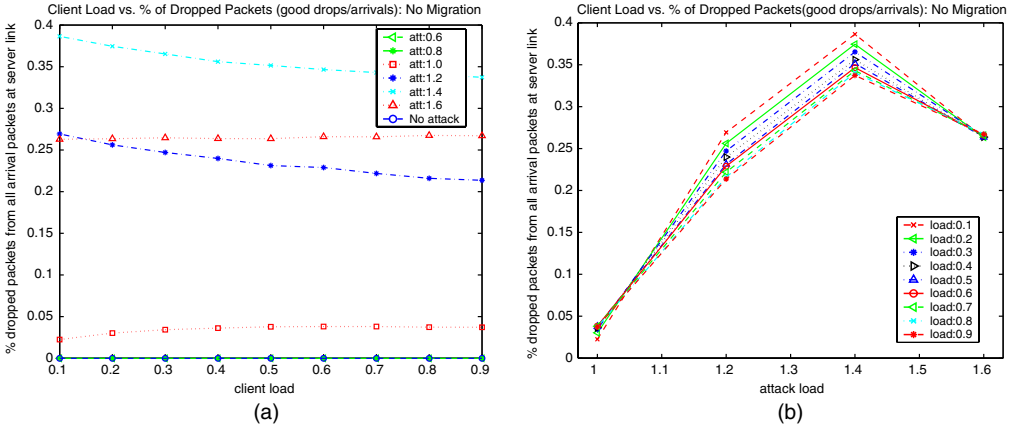

Fig. 10. Proportion of packet drops to arrivals: (a) categorized by attack load, (b) categorized by client load.

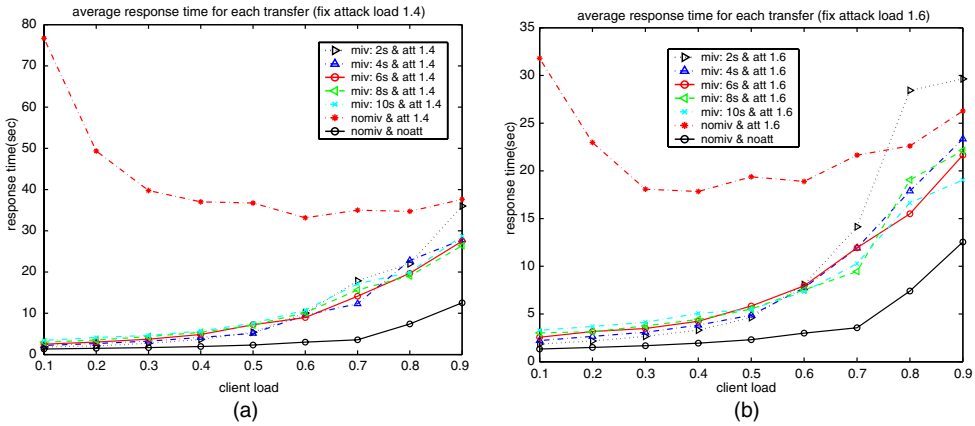

Fig. 11. Benefit of roaming: UDP attacks. (a) 1.4 UDP attack, (b) 1.6 UDP attack.

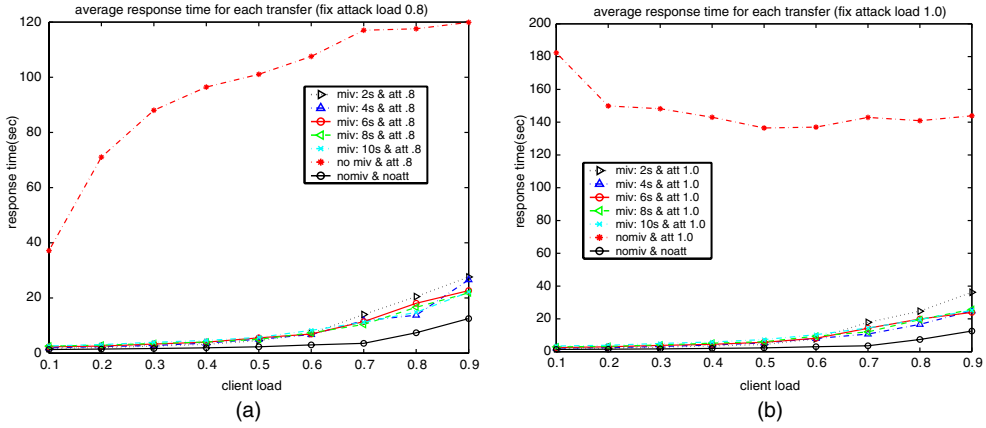

Fig. 12. Benefit of roaming: TCP attacks. (a) 0.8 TCP attack, (b) 1.0 TCP attack.

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.

7.2. Result/analysis of the 40-node network

For the simulation of the 40-node network, we focus to study statistics in the aggregate level. The cost and benefit of the roaming schemes are drawn from those perceived by the clients scattering in the network. We collect the start times and the response times of all clients to analyze the cost and benefit of clients based on the time they send the requests. To simplify the analysis, the FTP requests are grouped based on their start times in the granularity of 10 s. For example, the average response time of a FTP request in the thirtieth second (RT_{avg30}) is computed by $RT_{avg30} = TRT_{30}/NR_{30}$, where TRT_{30} is the total response times of all requests generated from time 21st to 30th and NR_{30} is the total number of the requests generated during that time.

7.2.1. The cost of the roaming

To measure the cost of the roaming, we compare case (1) and (2) as done in the 10-node network simulation. We pick the cases of total client load 0.4, 0.6, and 0.8 to study. The result amazes us as shown in Fig. 13. The cost of the roaming has changed to the profit as the total client load increase. Our explanation is that, at the light load (0.1–0.5), the migration introduces the unnecessary cost of disconnect and reconnection; at the high load (0.6 and over), however, the migration cost is outweighed by the profit of reducing effect from the congestion mechanism caused by the TCP. Once a congested TCP connection is moved to a new path, the congestion state is reset and the path is likely to have more available bandwidth.

7.2.2. The benefit of the roaming

In the cases of the attacks, the result shows that the migration improves the response time in all cases. As shown in Fig. 14, all the migration cases (i.e. migration interval 4, 8, 10, and 20 s) are below the non-migration

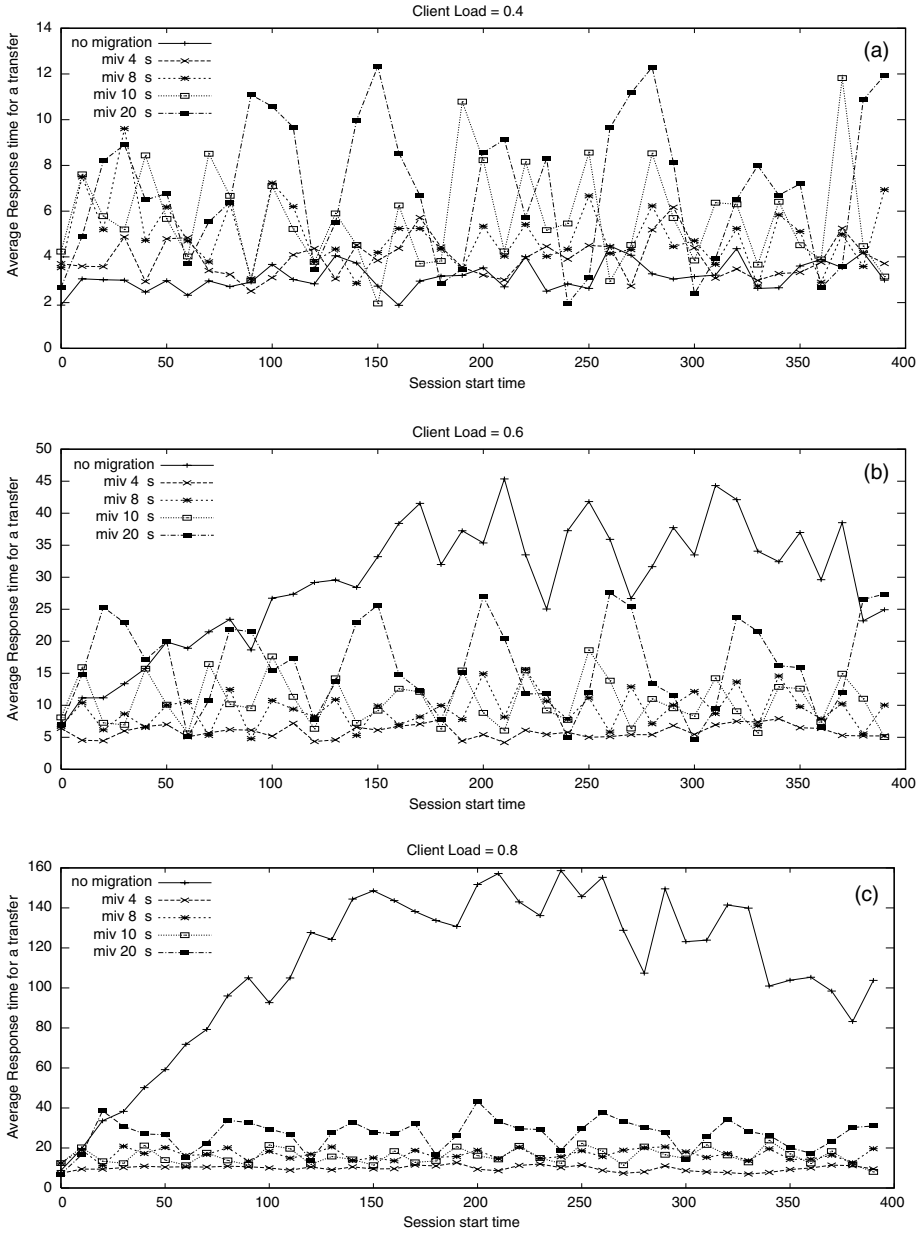

Fig. 13. Cost of the roaming in 40-node network: (a) client load 0.4, (b) client load 0.6, (c) client load 0.8.

cases, where they perceive the same attack load. In addition, as shown in the high client load case (0.8 client load, Fig. 14(b)), the migration improves the average response time of the clients in the cases of server being attacked surpass even that of the clients in the cases of the non-attack server. This can be explained as in the previous section that in the case of high client load, the TCPs seem to shrink their windows too much and quite often wait for timeout to occur; while the migration will refresh these TCP and let them start with the new fresh paths. Therefore the migrating TCPs are likely to progress more than the non-migrating ones.

8. Conclusion and future work

The paper proposes a framework for mitigating the effects of both node and link-based DoS attacks. A scheme, referred to as *Proactive Server Roaming*, was described and its implementation was discussed. The scheme uses the novel concept of "replicated elusive service" to mitigate the effects of DoS attacks. Based on this scheme, 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.

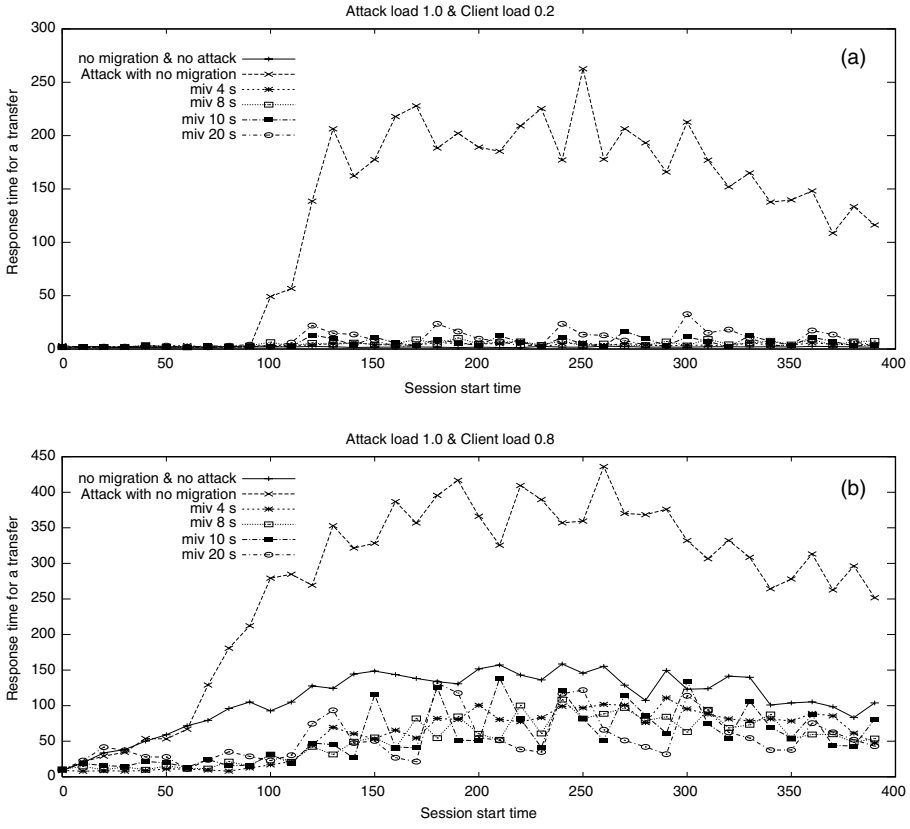

Fig. 14. Benefit of the roaming in 40-node network: (a) client load 0.2, (b) client load 0.8.

To study the roaming behavior of the clients and its effect on DoS, a simulation model was developed. The model uses NS2 and adds a socket layer to enable roaming of clients and servers, and handle connection migration. In addition, FTP server and client modules to simulate request–reply type of network applications were also created. These modules reside on top of the socket layer.

The experiments focused on four cases: 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; while the second case is experimented to find the extra cost caused by the roaming. The third case, no-roaming with attack, is simulated to discover the loss caused by the attack. Last, the case of roaming with attack is simulated to find the benefit of applying the server roaming to the servers in the malicious environment. In addition, we generate two topologies, namely 10- and 40-node networks, with different sizes to study the impact of the roaming scheme in the different scale of networks. 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.

The 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 the attacked services to explore different behaviors of the attackers and the roaming strategies to fight against them.

References

Andersen, D., Balakrishnan, H., Kaashoek, M., Morris, R., 2001. Resilient overlay networks. In: Proceedings of 18th ACM SOSP, pp. 131–145.

Axelsson, S., 2000. Intrusion detection systems: a survey and taxonomy. Tech. rep., Department of Computer Engineering, Chalmers University.

Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., Weiss, W., 1998. An architecture for differentiated services. In: IETF, RFC 2475.

Brustoloni, J., 2002. Protecting electronic commerce from distributed denial-of-service attacks. In: Proceedings of the Eleventh International World Wide Web Conference, pp. 553–561.

Desmedt, Y., 1994. Threshold cryptography. European Transactions on Telecommunications 5, 449–457.

Ferguson, P., Senie, D., 2001. Network ingress filtering: defeating denial of service attacks which employ ip source address spoofing. In: RFC 2827.

Garg, A., Reddy, A.L.N., 2002. Mitigation of dos attacks through qos regulation. In: Proceedings of the tenth IEEE International Workshop on Quality of Service, pp. 45–53.

Goldreich, O., Goldwasser, S., Micali, S., 1986. How to construct random functions. Journal of the ACM 33, 792–807.

Goldreich, O., Levin, L., Nisan, N., 1995. On constructing 1-1 one-way functions. Proceedings of the Electronic Colloquium on Computational Complexity 2, 1–11.

Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M., 1995. Proactive secret sharing or: How to cope with perpetual leakage. In: LNCS 963, Proceedings of the Crypto'95, vol. 5. Springer-Verlag, pp. 339–352.

Ivan, A., Harman, J., Allen, M., Karamcheti, V., 2002. Partitionable services: a framework for seamlessly adapting distributed applications to heterogeneous environments. In: Proceedings of the IEEE International Conference on High Performance Distributed Computing (HPDC), pp. 103–112.

Jones, J., 2000. Distributed denial of service attacks: defenses, a special publication. Tech. rep., Global Integrity.

Juels, A., Brainard, J., 1999. Client puzzles: a cryptographic countermeasure against connection depletion attacks. In: Proceedings of NDSS '99, Networks and Distributed Security Systems, pp. 151–165.

Keromytis, A., Misra, V., Rubenstein, D., 2002. Sos: secure overlay services. In: Proceedings of the SIGCOMM'02 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. ACM Press, pp. 61–72.

Khattab, S.M., Sangpachatauaruk, C., Znati, T., Melhem, R., Moss, D., 2003. Proactive server roaming for mitigating denial-of-service attacks. In: Proceedings of 1st International Conference on Information Technology Research and Education (ITRE), pp. 1–5.

Mankins, D., Krishnan, R., Boyd, C., Zaho, J., Frentz, M., 2001. Mitigating distributed denial of service attacks with dynamic resource pricing. In: Proceedings of Annual Computer Security Applications Conference (ACSAC 2001), pp. 1–11.

Park, K., Lee, H., 2000. On the effectiveness of probabilistic packet marking for ip traceback under denial of service attack. Tech. rep., Department of Computer Sciences, Purdue University.

Park, K., Lee, H., 2001. On the effectiveness of route-based packet filtering for distributed dos attack prevention in power-law internets. In: Proceedings of the SIGCOMM'0l Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. ACM Press, pp. 15–26.

Paxson, V., 2001. An analysis of using reflectors for distributed denial-of-service attacks. ACM Computer Communications Review (CCR) 31 (3). Available from <citeseer.nj.nec.com/paxson01analysis.html>.

Perrig, A., Szewczyk, R., Cullar, D., Wen, V., Tygar, J.D., 2001. Spins: security protocols for sensor networks. In: Proceedings of MOBICOM'01, pp. 189–199.

Ratnassamy, S., Handly, M., Karp, R., Shenker, S., 2002. Topologically-aware overlay construction and server selection. In: Proceedings of the INFO-COMM'02 Conference, vol. 3. pp. 1190–1199.

Rivest, R., 1992. The md5 message-digest algorithm. In: RFC 1321.

Sangpachatanaruk, C., Khattab, S.M., Znati, T., Melhem, R., Moss, D., 2003. A simulation study of the proactive server roaming for mitigating denial of service attacks. In: Proceedings of the 36th Annual Simulation Symposium 2003, pp. 7–14.

Savage, S., Wetherall, D., Karlin, A., Anderson, T., 2000. Practical network support for ip traceback. In: Proceedings of the SIGCOMM'00 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication. ACM Press, pp. 295–306.

Snoeren, A.C., 2001. Hash-based ip traceback. In: Proceedings of the SIG-COMM'01 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. ACM Press, pp. 3–14.

Snoeren, A.C., Andersen, D.G., Balakrishnan, H., 2001a. Fine-grained failover using connection migration. In: Proceedings of 3rd USENIX Symposium on Internet Technologies and Systems (USITS), pp. 221–232. Available from <citeseer.nj.nec.com/snoeren01finegrained.html>.

Snoeren, A.C., Balakrishnan, H., Kaashoek, M.F., 2001b. The migrate approach to internet mobility. In: Proceedings of the Oxygen Student Workshop, July 2001, pp. 1–2.

Song, D.X., Perrig, A., 2001. Advanced and authenticated marking schemes for IP traceback. In: Proceedings IEEE Infocomm 2001, pp. 878–886.

Spatscheck, O., Petersen, L.L., 1999. Defending against denial of service attacks in scout. In: Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI'99), New Orleans, Louisiana, pp. 59–72.

Sultan, F., Srinivasan, K., Iyer, D., Iftode, L., 2001. Migratory tcp: highly available internet services using connection migration. Tech. rep., Rutgers University.

Sultan, F., Srinivasan, K., Iyer, D., Iftode, L., 2002. Migratory tcp: connection migration for service continuity in the internet. In: Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS) 2002, pp. 369–370.

Wong, C.K., Gouda, M., Lam, S., 1998. Secure group communications using key graphs. In: Proceedings of SIGCOMM'98, pp. 68–79.