Roaming Data Redundancy for Assurance in Critical Data Services

Chin-Tser Huang, Alexander B. Alexandrov, Prasanth Kalakota

Department of Computer Science and Engineering
University of South Carolina
{huangct, alexand2, kalakota}@cse.sc.edu

***Abstract*— Static data redundancy has been proven useful in providing fault tolerance and load balancing, but it may not provide enough assurance on the continuous availability of mission critical data in the presence of a determined adversary which locates all the static redundant copies and shuts them down using DoS attacks. In this paper, we propose a roaming data redundancy scheme to address this problem. This scheme requires that if a certain percentage of static redundant copies of the critical data has already been shut down, then a small number of roaming redundant copies will be stored in randomly selected hosts that are changed periodically. Simulation results show that this scheme can effectively mitigate the impacts of DoS attacks and host failures to ensure continuous availability of critical data, with better efficiency compared to the static data redundancy scheme.**

***Index Terms*—Data Redundancy, Assurance, Availability, Denial-of-Service Attacks.**

I. Introduction

Data redundancy in host-based services is widely used for the purposes of fault tolerance and load balancing. In general, replications of the target data are made and stored on hosts other than the original source. When the original source is down, the target data can be recovered from the redundant copy. Moreover, access to the target data may also be provided via the redundant copies, thus distributing the load from the original source. However, data redundancy does not provide complete assurance, because it is possible that the original source and the hosts that keep the redundant copies are down at the same time. If the data in question is mission critical, then it is desirable to provide greater assurance of the continuous availability of the critical data.

Consider the following scenario in which a specialized host in a military network is responsible for the update and maintenance of one type of critical data. (For example, this host is attached to a special sensing device, or this host is superior to other hosts in its computing power.) The clients, soldiers in this example, need continuous access to the critical data, as this critical data is essential for the clients to obtain timely awareness of the situation and perform their tasks properly. However, if the server that is responsible for the critical data is shut down by a Denial-of-Service (DoS) attack, then the critical data is no longer available to the clients.

One solution to provide assurance on availability of the critical data is by using static data redundancy. In this case, multiple hosts in the network are selected to keep a redundant copy of the critical data. If the specialized server of this critical data is shut down by a system failure or a DoS attack, then the clients can still get access to the critical data through the hosts that keep the redundant copies. However, if the hosts that keep the redundant copy of the critical data are also down because a determined adversary locates these hosts (via observing ongoing traffic or sending fake inquiries) and launches more DoS attacks against these hosts, then again the critical data become unavailable to the clients. In order to provide greater assurance, it is tempting to store larger number of redundant copies in the network, such that the adversary does not have enough power to shut down all of the redundant copies simultaneously. However, this scheme may incur too much overhead because it needs to update many redundant copies when the original copy is updated. In addition, if more redundant copies of the critical data is stored in more hosts, then the risk of critical data leakage due to physical compromise also increases.

In this paper, we propose a novel roaming data redundancy scheme to provide greater assurance in critical data services. In this solution, when a certain portion (say two thirds) of static redundant copies has already been shut down, a small number of the remaining hosts in the network are selected to keep a roaming redundant copy of the critical data. Periodically, the roaming redundant copies are updated by the critical data server and moved to another randomly selected host(s). An adversary may launch in parallel a number of DoS attacks on several random hosts in the network, but if at least one host that keeps a copy of the critical data (either original copy or redundant copy) is not attacked, then the clients can still get access to the critical data. If the original source of critical data is down for an extended time, the alive redundant copies will keep roaming to avoid the attack. This solution is generic in itself and is applicable to mobile networks and static networks alike.

There are four goals in the design of the roaming data redundancy scheme. Firstly and foremost, we aim to keep the critical data constantly available. We achieve this by periodically changing the locations of redundant copies, such that the adversary cannot easily locate a host that keeps a redundant copy and then shut it down by launching a DoS attack. Secondly, we want to prevent the adversary from misleading the hosts about the content of the critical data and the locations of redundant copies by modifying a message or replaying an old message. This goal is achieved through public key encryption and sequence numbers. Thirdly, we want to periodically update the redundant copies to keep them closely consistent with the original critical data. It is advantageous but hard to always keep the redundant copy exactly the same as the original critical data if the original copy is updated frequently. However, keeping an approximate redundant copy of the critical data is better than no copy at all in the presence of serious DoS attacks. Fourthly, we aim to reduce the storage requirement on hosts other than the original source of the critical data. This goal is achieved by storing just a small number of roaming redundant copies. By doing this, we also save the overhead of updating the redundant copies.

The remainder of this paper is organized as follows. In Section II, we survey previous works that are related to the application of data redundancy for the purpose of fault tolerance. In Section III, we discuss the assumptions we make about the critical data service and the adversary in this work. In Section IV, we present the roaming data redundancy scheme and its two major components, namely the redundant data moving protocol and the redundant data discovery protocol. In Section V, we use simulation results to evaluate the performance of our scheme. Finally, we conclude and discuss future works in Section VI.

II. Related Works

Although much work has been done in both data redundancy and distributed fault tolerance in the presence of DoS attacks, we only describe here work that is directly related to our roaming data redundancy scheme.

In the pStore framework [1] files are shared in a peer-to-peer fashion to create a secure backup system that supports versioning and takes advantage of similar chunks of information between different versions of the same file or different files. Although the approach supports distributed fault tolerance, it does not distinguish between critical data and non-critical data and concentrates mainly on data storage gains when similar data is backed up on multiple computers. Furthermore, once the system converges, the backup data does not change location and it is easy for an attacker to locate the keeper of the backup data and launch a DoS attack against it.

IP hopping [5] protects a public server by physically changing its IP address when the server is under attack. All legitimate clients follow the server through contacting the local DNS server which always keeps the current IP address of the server. While this approach may delay a DoS attack, it suffers from filtering out legitimate client requests during the migration period. Furthermore, a persistent adversary can always resolve the current IP through the DNS server and can resume the attack.

Both TCP-Migrate [9] and the Migratory-TCP [11] describe a framework for terminating a live connection at either ends and reincarnating it with all state information intact at another location. Although they help fault tolerance and mobility, they necessitate certain changes in the TCP/IP protocol and thus cannot be readily deployed in existing environments.

Contrary to IP hopping, [7] physically roam a single server for the purpose of frustrating any possible attacker. Time is divided into epochs and only legitimate clients can follow the current server while the remaining machines will just drop all requests or collect information about the attack for later analysis and incorporation into a site-specific Intrusion Detection System (IDS). However their experimental results show an increase of about 14% in average response time during an attack-free environment due to the migration of the active TCP connection between static clients and constantly migrating server. Our main goal is keeping critical data available at all times and using proactive server roaming does not achieve this goal. Depending on the epoch length, it might be possible to take down the current active server, effectively achieving a successful DoS attack before the epoch expires and the server migrates. Additionally, this approach does not provide for fault tolerance since no running replicas of the data are kept – all the servers maintain the same data and no data is transferred between two distinct servers.

The roaming honeypot framework proposed in [6] keeps n out of m servers active at any specific time, rendering the remaining m - n servers acting as honeypots. The current n active servers can be contacted through an Access Gateways Overlay Network. Only the legitimate clients know about the existence of this network and the attackers select at random a server to attack without having any way to distinguish between a legitimate server and a honeypot. Although this approach achieves fault tolerance, it physically moves servers and does not replicate data. Our approach assumes a small number of redundant copies in existence at any given time, providing both high level of assurance of availability and a limited level of load balancing.

In addition, a dynamic replica placement for scalable content delivery using Tapestry is presented in [2]. While this approach has low requirements on total bandwidth consumed and needs minimum replicas, it does not consider any defenses against a DoS attack. Nevertheless, constructing a distribution tree for more efficient client access service is something we can incorporate into our model in the future to achieve higher framework utility.

It is worthwhile to mention another related mechanism called frequency hopping [12]. Frequency hopping is an approach commonly used to secure wireless networks. The frequency of the wireless media is changed periodically in a predetermined order that only the sender and the recipient know, such that it is very hard to eavesdrop on or intercept data, let alone decrypt the data in transit. Although the idea used is similar to our approach, [12] shows that the hop sequences of Frequency Hopping radios can be determined in

less than five seconds due to slow hop rate (even Bluetooth is considered slow at 1600 hops/second), limited number of different hop patterns, and a beacon that in some implementations is transmitted each time the network hops to a new channel. Our approach provides higher security by using public/private key encryption, as well as new algorithms for roaming data discovery and roaming data movement. In addition, our approach can be used over any network media.

III. Assumptions

Before we present the roaming data redundancy scheme, we discuss the assumptions we make about the critical data service and the adversary. For generality, it is assumed that there are multiple types of critical data, and that all the legitimate clients are aware of which type of critical data is maintained by which host. To protect the privacy and integrity of critical data and the locations of redundant copies, all the communications in this scheme are encrypted; unicast messages are encrypted by appropriate public key that is distributed to all the critical data service hosts and legitimate clients through a Public Key Authority, and broadcast messages are encrypted by a secret key shared among all legitimate hosts. All the hosts in the critical data service network are assumed to be trusted, which means that they will not collude with the adversary by leaking out the private key or information about the content or location of the critical data. Due to the nature of public key encryption, it is computationally impossible for the adversary to break the keys using statistical analysis.

We assume that the adversary is able to passively observe the messages exchanged between the nodes in the network; however, the adversary is unable to see the type and content of the observed message if the message is encrypted. Still, the adversary can apply traffic analysis techniques as discussed in [10] and make a guess on the current operations of the network. It is assumed that the adversary can apply two types of active attacks on the messages. First, the adversary can apply a *message modification* attack by arbitrarily modifying the content of a message. Second, the adversary can apply a *message replay* attack by making a copy of an observed message and replaying the copied message at a later time.

For the purpose of evaluation, we assume that the adversary is aware of the addresses of all the hosts in the network. We also assume that the adversary is capable of simultaneously launching *m* DoS attacks to shut down *m* hosts in the network at the same time, where the value of *m* is larger or equal to the number of redundant copies plus one (the original source). Therefore, the adversary is capable of simultaneously shutting down all the hosts that keep a copy of the critical data.

IV. Roaming Data Redundancy

In this section, we present our roaming data redundancy scheme. This scheme consists of two protocols: the *redundant data moving protocol* and the *redundant data discovery protocol*. The function of the redundant data moving protocol is to allow each host to periodically move the redundant copy of its critical data to a different location. The function of the redundant data discovery protocol is to allow a host to discover the location of the redundant copy of one type of critical data. Each host in the network executes a process of the redundant data moving protocol and a process of the redundant data discovery protocol. The redundant data moving process executes on the top of the redundant data discovery process, as the redundant data discovery process depends on the redundant data moving process to provide information about what redundant copies are currently kept by this host. Both protocols are designed to incorporate multiple types of critical data, with each type of critical data maintained by a different host. Note that as discussed in Section III, all types of messages exchanged in this protocol are encrypted by the sending process and decrypted by the receiving process using public key encryption to protect the privacy and integrity of critical data and the locations of redundant copies.

A. Redundant Data Moving Protocol

The redundant data moving protocol consists of n processes rdm[0..n-1]. Each host participating in the protocol has an input cd which represents the critical data maintained by this host. The host for a particular cd is called owner for that particular cd and only the owner of a cd has the authority to create redundant copies of it. Each rdm[i] also maintains an array rd[0..n-1] which represents the redundant copies of other host cds currently kept by this host. Each rdm[i] maintains an array sq[0..n-1] that represents the next sequence number to be used by each process to send the next request message (mov) to move the cd. Periodically, process rdm[i] randomly selects the next keeper of its critical data, broadcasts to every other host a dlt(sq[i], i, tmp) message (where sq[i] is the sequence number of message sent by rdm[i], i is the index of rdm[i], and tmp is the index of the current keeper of the redundant copy) to notify the current keeper to delete the outdated redundant copy, and unicasts a mov(sq[i], i, kprs, cd) message (where kprs is a list of next keepers) to each one of the next keepers to copy or update its critical data. Note that the dlt message is broadcasted because every time rdm[i] sends out a dlt message, sq[i] needs to be incremented by 1 in every process in order to stay synchronized. If process rdm[j] currently keeps the redundant copy, then rdm[j] will send a dltack message to rdm[i] to acknowledge the deletion. If process rdm[j] is the next keeper of redundant copy, then rdm[j] will send a movack message to rdm[i] to acknowledge the reception of the redundant copy. If the keeper that is listed first on the keeper list of a redundant copy does not hear dlt message from the owner of a critical data after 3 time periods, it assumes the owner is down and assumes the role of an owner from this timeframe – it informs the rest of the network it is the new owner and then chooses hosts to keep redundant copies of the new critical data it now owns. Similarly, if the keeper that is listed second on the keeper list of a redundant copy does not hear dlt message from the owner or the first keeper of a critical data after 6 time periods, it assumes that both the owner and the first keeper are down and assumes the role of an owner.

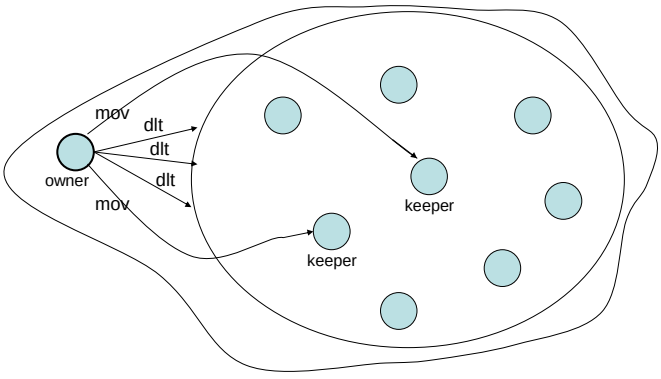
There are five actions in this protocol. In the first action, the timer expires after T seconds passed since last movement of redundant copy, and process rdm[i] checks that it is not waiting for acknowledgment from the last keeper and current

keeper of the redundant copy, then rdm[i] randomly selects a host to keep the critical data next. rdm[i] sends a dlt message to every other host so that the last keeper can delete its outdated copy, increments the sequence number sq[i] associated with its mov message, and sends a mov message to the next keeper of the redundant copy of that particular cd. Notice that the dlt message is sent to every other process so that every other process can update sq[i], and that sq[i] is incremented after sending the dlt message and before sending the mov message in order to ensure consistency of sq[i] in every process.

In the second action, when process rdm[i] receives a dlt message from another process rdm[j], rdm[i] first checks that the dlt message is really sent by rdm[j] and is not a replayed message. Then, rdm[i] checks if rdm[i] itself is currently the owner of the redundant copy of the critical data maintained by rdm[j]. If so, rdm[i] deletes the outdated redundant copy and sends a dltack message to rdm[j]. Otherwise, rdm[i] will just disregard the message. In both cases, rdm[i] updates the sequence number of rdm[j] to ensure consistency.

In the third action, when process rdm[i] receives a mov message from another process rdm[j], rdm[i] first checks that the mov message is really sent by rdm[j] and is not a replayed message. Then, rdm[i] checks if rdm[i] is chosen by rdm[j] as the next keeper of the redundant copy of the critical data maintained by rdm[j]. If so, rdm[i] stores the redundant copy included in the mov message and sends a movack message to rdm[j]. Otherwise, rdm[i] will just disregard the message. In both cases, rdm[i] updates the sequence number of rdm[j] to ensure consistency.

In the fourth action, rdm[i] receives dltack from the last keeper of the redundant copy, so rdm[i] sets waitdlt to false. In the fifth action, rdm[i] receives movack from the current keeper of the redundant copy, so rdm[i] sets waitmov to false. Figure 1 illustrates the basic operations in the redundant data moving protocol.



At every time step each node holding critical data broadcasts dlt message and randomly chooses next n nodes (mov) to hold the critical data for the next time step.

Figure 1: Basic operations of redundant data moving protocol.

There are two possible fault situations that deserve more discussion. First, a message that contains a redundant copy of critical data is corrupted in transit, either due to modification by an adversary or transmission error. This will be detected by the receiving process because if an encrypted message is corrupted, its fields will become inconsistent after decryption. According to the protocol a corrupted message will be discarded for the sake of consistency. Second, the receiving process may crash due to host failure. This will be detected by the sending process because no acknowledgment is received. In either case of message corruption or host failure, we will not retransmit or find another host to keep a redundant copy. Instead, the protocol just regards that one redundant copy is lost in the current period. Note that losing one redundant copy is okay if there is at least one alive copy of the critical data in the current period. In Section V we use simulation to show that our roaming data redundancy scheme is resilient in tolerating attacks and failures.

B. Redundant Data Discovery Protocol

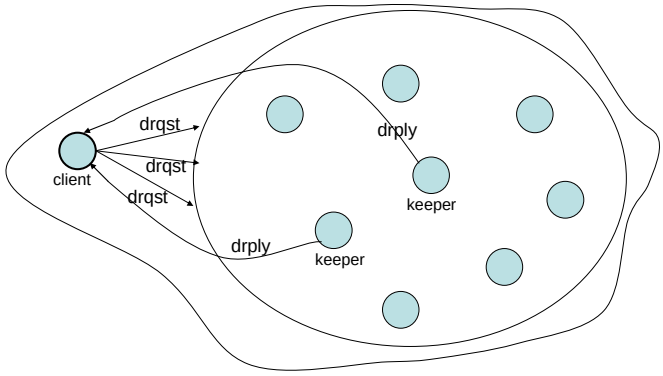
The redundant data discovery protocol consists of n processes rdd[0..n-1]. Each process rdd[i] maintains an input array rd[0..n-1] which is provided by rdm[i] in the redundant data moving protocol. Similar to the redundant data moving protocol, each process rdd[i] also maintains an array sq[0..n-1] that represents the next sequence number to be used by each process to send the next query (drqst). Each process rdd[i] in the redundant data discovery protocol can send to every other process a drqst(sq[i], tgt, i) message, where sq[i] is the sequence number of drqst message sent by rdd[i], tgt is the index of the target critical data, and i is the index of rdd[i]. Every time rdd[i] sends out a drqst message, sq[i] needs to be incremented by 1 in every process in order to stay consistent. If process rdd[j] currently keeps the redundant copy, then rdd[j] will send a drply(sq[i], tgt, j) message to rdd[i], where sq[i] is the corresponding sequence number of rdd[i], tgt is the index of the target critical data, and j is the index of rdd[j]. The other processes that do not keep the redundant copy will just disregard the drqst message.

There are three actions in this protocol. In the first action, process rdd[i] checks that it does not have any pending request, and then randomly selects a target type of critical data to request for. If the original source of the target critical data is down, then rdd[i] must discover the redundant copy by sending out a drqst message to every other process. A monotonically increasing sequence number is attached to each drqst message to counter replay attacks. Therefore, an adversary cannot replay a legitimate drqst message in the hope to learn the location of the redundant copy of the requested critical data.

In the second action, when process rdd[i] receives a drqst message from another process rdd[j], rdd[i] first checks that the drqst message is really sent by rdd[j] and is not a replayed message. Then, rdd[i] checks if rdd[i] itself is currently the keeper of the redundant copy of the requested critical data. If so, rdd[i] sends a drply message to inform rdd[j] that the redundant copy of the requested critical data can be accessed through rdd[i]. Otherwise, rdd[i] will just disregard the message. In both cases, rdd[i] updates the sequence number of rdd[j].

In the third action, when process rdd[i] receives a drply message from another process rdd[j], rdd[i] first checks that the drply message is really sent by rdd[j] and is not a replayed message. Then, rdd[i] checks if the drply message is

corresponding to the target type of critical data that rdd[i] requested in the drqst message it sent out. If so, rdd[i] will proceed to access the critical data through rdd[j]. Otherwise, rdd[i] will just disregard the message. Figure 2 illustrates the basic operations in the redundant data discovery protocol.



Every node requesting specific critical data broadcasts drqst message. All the nodes that contain a replica reply with drply and the requesting node accesses the nearest copy.

Figure 2: Basic operations of redundant data discovery protocol.

V. Simulation and Evaluation

We have developed a simplified model of our roaming data redundancy scheme in Java and conducted simulation experiments to study the effects of different parameters on our scheme. Specifically, we evaluate the tolerance of our scheme against DoS attacks and host failures. We compare the results to another scheme in which a large number of static redundant copies are stored.

Simulation model: We create a simulation model in Java that implements our model. We start with a simulation designed to evaluate the tolerance of our scheme against DoS attacks. Without loss of generality we perform our evaluation with 1, 2, or 3 servers which hold the roaming redundant copies. We expect the results to be similar when the number of redundant copies increases; indeed, similar trend is manifested in the simulation results. Every time unit the original source server randomly chooses 1, 2, or 3 servers out of 100 total servers to keep the redundant copies. Every time unit the adversary randomly chooses 10, 20, or 30 servers to attack simultaneously. If all the current roaming redundant copies are hit by the DoS attacks, then the attack is regarded successful and we measure the time elapsed in time units. Otherwise, in the next time unit the original source server again randomly chooses 1, 2, or 3 servers and the adversary again randomly chooses 10, 20, or 30 servers to attack. The longer the elapsed time before the attack succeeds the better the performance is, since the network proves to be more robust to the attack.

Then, the simulation is extended to evaluate the tolerance of our scheme against host failures. Every time the original source server chooses a server to keep a redundant copy, a random number is used to determine whether this server is up or crashed. If this server is crashed, then one redundant copy is considered lost in the current period. If all the current roaming redundant copies are hit by the DoS attacks or crashed due to server failure, then the attack is regarded successful and we measure the time elapsed in time units. Server failure probability of 5% is used in the simulation, which is already a very high server failure rate and is not likely to happen often in reality.

We compare our model with another scheme in which N static redundant copies are stored in N servers out of a total of 100 servers. At the beginning of the simulation the source chooses 10, 20, or 30 servers to keep the redundant copies. At the beginning, the attacker randomly selects 10, 20, or 30 servers to attack. If the adversary hits a server that keeps a redundant copy, the adversary keeps one attack on that server. The adversary uses its remaining attacks to keep attacking until it locates and shuts down all the servers that keep a redundant copy, and we measure the time elapsed in time units.

Both models assume either direct or overlay node connectivity and a constant number of simultaneous DoS attacks in a single simulation.

Evaluation: Figure 3 shows the statistics of 1000 runs for our roaming data redundancy model – 1, 2, 3 roaming copies in 100 total servers under 10, 20, 30 attacks. The results are compared with the same setup but with 5% host failure probability. As the number of attacks increases, the average time needed for the adversary to succeed in its attack decreases. As we increase the number of roaming copies, the time needed for the adversary to succeed increases exponentially. Therefore by increasing the number of roaming copies by just one, we can achieve exponential increase in the difficulty for the adversary.

Figure 4 shows the statistics for our comparison model – 10, 20, 30 static copies distributed in 100 total servers. Note that the number of simultaneous attacks needs to be larger than the number of static copies in order for the adversary to be able to successfully shut down all the static copies. While the increase of the number of static copies increases the time necessary for a successful attack, the increase is smooth and the average time needed for a successful attack is apparently shorter than when 2 or 3 roaming copies are used.

From the figures it is clear that the roaming data redundancy scheme performs better than N static copies, and Figure 3 shows that the benefits of using our approach increase when the number of roaming copies increases, as the average time needed for the attack to succeed increases by around 10 times with every additional roaming copy.

VI. Concluding Remarks

In this paper, we point out the need for greater assurance of the continuous availability of critical data services, and show that current solutions are not sufficient to provide the desired level of assurance under serious DoS attacks. We then introduce a novel roaming data redundancy scheme that aims to ensure constant availability of critical data by changing the locations of the redundant copies of critical data periodically. Simulation results show that with a small number of roaming redundant copies, the roaming data redundancy scheme effectively mitigate the impacts of DoS attacks and server failures, with better efficiency compared to another scheme that stores many static redundant copies.

In the future work, we will incorporate untrusted hosts into the network to increase the flexibility of our scheme. The cryptographic accumulator proposed in [3] suggests a scheme for authenticating data provided by a mirror site, which may be incorporated into our scheme. Moreover, we will conduct simulations and experiments to show when is the best time to start using the roaming data redundancy scheme, and how frequently the roaming copies should be moved. Furthermore, we would like to investigate the impacts that the topology of the critical data service network and the routing algorithm of the messages have on the overall performance of the roaming data redundancy scheme.

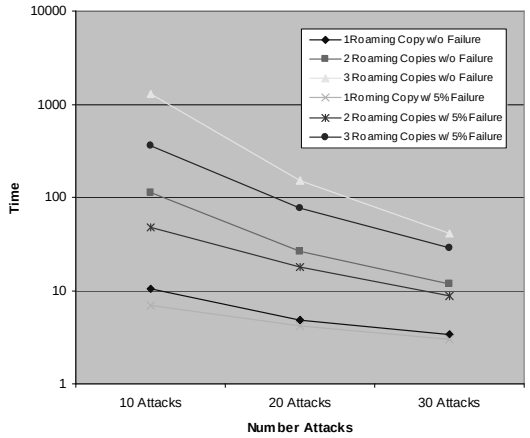


Figure 3: Time for successful DoS attack, with 1, 2, 3 roaming copies

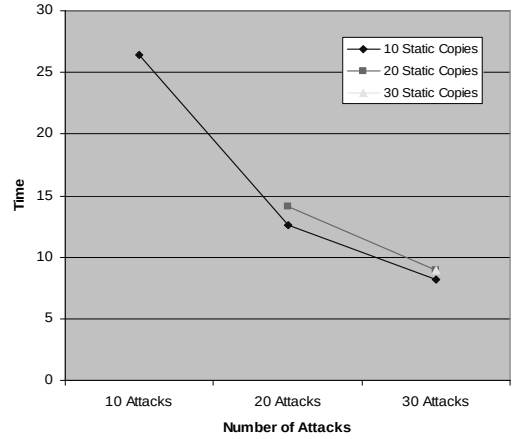


Figure 4: Time for successful DoS attack, with 10, 20, 30 static copies respectively

REFERENCES

[1] C. Batten, K. Barr, A. Saraf, S. Trepetin, "pStore: A Secure Peer-to-Peer Backup System," MIT LCS Technical Memo 632, 2002.

[2] Y. Chen, R. H. Katz, J. D. Kubiatowicz, "Dynamic Replica Placement for Scalable Content Delivery", Proceedings of First International Workshop on Peer-to-Peer Systems (IPTPS 2002), Cambridge, MA, March 2002.

[3] M. T. Goodrich, R. Tamassia, "An Efficient Dynamic and Distributed Cryptographic Accumulator," Proceedings of the 5th International Conference on Information Security, September 2002.

[4] M. G. Gouda, Elements of Network Protocol Design, John Wiley & Sons, New York, NY, 1998.

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

[6] S. M. Khattab, C. Sangpachatanaruk, D. Mossé, R. Melhem, T. Znati, "Roaming Honeypots for Mitigating Service-Level Denial-of-Service Attacks," Proceedings of 24th International Conference on Distributed Computing Systems, March 2004.

[7] S. M. Khattab, C. Sangpachatanaruk, D. Mossé, T. Znati, "Proactive Server Roaming for Mitigating Denial-of-Service Attacks", Annual Simulation Symposium, 2003.

[8] S. M. Khattab, C. Sangpachatanaruk, D. Mossé, R. Melhem, T. Znati, "A Simulation Study of the Proactive Server Roaming for Mitigating Denial of Service Attacks", Proceedings of the 36th Annual Simulation Symposium (ANSS'03).

[9] A. C. Snoeren, H. Balakrishnan, and M. F. Kaasoek, "The migrate approach to Internet mobility," in Proc. of the Oxygen Student Workshop, July 2001.

[10] W. Stallings, Cryptography and Network Security: Principles and Practices, 3rd ed., Prentice Hall, NJ, 2003.

[11] F. Sultan, K. Srinivasan, D. Iyer, and L. Iftode, "Migratory TCP: Connection migration for service continuity in the Internet," Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS), July 2002.

[12] J. Zyren, T. Godfrey and D. Eaton, "Does frequency hopping enhance security?", available at http://www.packetnexus.com/docs/20010419_frequencyHopping.pdf