



## ABSTRACT

Title of Thesis: PERFORMANCE ANALYSIS OF AN APPLICATION-LEVEL  
MECHANISM FOR PREVENTING SERVICE FLOODING IN  
THE INTERNET

Degree Candidate: Zubair Baig

Degree and Year: Master of Science, 2003

Thesis directed by: Dr. Virgil D. Gligor

Department of Electrical and Computer Engineering

One of the most impacting technological developments during the last few years has been the emergence of the Internet. With rapid growth of the Internet, it is becoming increasingly difficult to provide the necessary services to all users within a designated time period. As the gap between the network-line and application-server rates is growing, it is getting easier to launch Distributed Denial of Service (DDoS) attacks against services on the Internet, and remain undetected within the network. Gligor's rate control scheme is a novel mechanism for providing

strong access guarantees to clients for accessing public services, by generating and enforcing simple user-level agreements on dedicated special purpose servers.

This thesis studies the results obtained from simulations, when this rate control scheme is applied to two kinds of networks, namely, Content Distribution Networks, and Domain Name Server-based networks. In particular, the server utilization, and client waiting times were studied with the aim of finding bounds on parameters that improve server performance, and of providing clients with reasonable maximum waiting times to service.

PERFORMANCE ANALYSIS OF AN APPLICATION-LEVEL MECHANISM FOR PREVENTING SERVICE FLOODING IN THE INTERNET

by

Zubair Baig

Thesis submitted to the Faculty of the Graduate School of the  
University of Maryland, College Park in partial fulfillment  
of the requirements for the degree of  
Master of Science  
2003

Advisory Committee:

Professor Virgil D. Gligor, Chairman/Advisor  
Professor Manoj Franklin  
Professor Charles B. Silio

© Copyright by  
Zubair Baig  
2003



## ACKNOWLEDGEMENTS

I am extremely grateful to my advisor, Dr. Virgil D. Gligor, for giving me the opportunity to work with him during the past year. He spent many hours advising and guiding me at every stage of the thesis. His invaluable efforts have helped mature my way of thinking and have made me a more dedicated researcher and engineer.

I would also like to thank my parents for their many sacrifices, love, and unconditional support all along. I thank Dr. Charles B. Silio, Jr. and Dr. Manoj Franklin for reading this thesis and being part of my thesis committee. Finally, I extend my thanks to my colleagues, especially, Aamer Jaleel, Rakesh Bobba, and Farshad Bahari for their support, and hallway guidances, that helped me reach this stage of my academic efforts.

## TABLE OF CONTENTS

<b>List of Figures .....</b>	<b>iv</b>
<b>CHAPTER 1:</b>	
<b>Introduction .....</b>	<b>1</b>
<b>CHAPTER 2:</b>	
<b>Related Work .....</b>	<b>7</b>
<b>CHAPTER 3:</b>	
<b>Rate Control Scheme with Maximum Waiting Time Guarantees.....</b>	<b>16</b>
<b>CHAPTER 4:</b>	
<b>Simulation Analysis .....</b>	<b>23</b>
4.1 Rate Control Scheme applied to CDNs .....	23
4.1.1 Working of CDNs .....	24
4.1.2 Simulation Experiment .....	25
4.1.3 Results and Observations .....	27
4.2 Rate Control Scheme applied to DNS Root Name Servers .....	39
4.2.1 DNS Name Resolution Scheme.....	39
4.2.2 Simulation Experiment .....	41
4.2.3 Results and Observations .....	42
<b>CHAPTER 5:</b>	
<b>Conclusions and Future Work .....</b>	<b>54</b>
<b>APPENDIX A:</b>	
<b>The Simulator .....</b>	<b>57</b>
<b>BIBLIOGRAPHY .....</b>	<b>72</b>



## LIST OF FIGURES

Fig. 2.1 ---- Flooding-based DDoS Attacks: Direct and Reflector.....	9
Fig. 2.2 ---- Honeypots for Protection against DDoS Attacks.....	12
Fig. 3.1 ---- Rate Control Scheme.....	17
Fig. 4.1 ---- Server Utilization vs. $l$ .....	34
Fig. 4.2 ---- Server Utilization vs. $l$ .....	35
Fig. 4.3 ---- Server Utilization vs. $l$ .....	35
Fig. 4.4 ---- Server Utilization vs. $l$ .....	36
Fig. 4.5 ---- Average Waiting Time vs. $l$ .....	36
Fig. 4.6 ---- Average Waiting Time vs. $l$ .....	37
Fig. 4.7 ---- Average Waiting Time vs. $l$ .....	37
Fig. 4.8 ---- Average Waiting Time vs. $l$ .....	37
Fig. 4.9 ---- Average Waiting Time vs. Client Population.....	38
Fig. 4.10 ---- Average Waiting Time vs. Client Population.....	38
Fig. 4.11 ---- Average Waiting Time vs. Client Population.....	38
Fig. 4.12 ---- Average Waiting Time vs. Client Population.....	39

Fig. 4.13 ---- Impact of $\delta_r$ variation on the server utilization.....	43
Fig. 4.14 ---- Access window allocation to clients .....	46
Fig. 4.15 ---- Server Utilization vs. $l$ .....	49
Fig. 4.16 ---- Server Utilization vs. $l$ .....	49
Fig. 4.17 ---- Server Utilization vs. $l$ .....	50
Fig. 4.18 ---- Server Utilization vs. $l$ .....	50
Fig. 4.19 ---- Average Waiting Time vs. $l$ .....	51
Fig. 4.20 ---- Average Waiting Time vs. $l$ .....	51
Fig. 4.21 ---- Average Waiting Time vs. $l$ .....	52
Fig. 4.22 ---- Average Waiting Time vs. $l$ .....	52
Fig. 4.23 ---- Average Waiting Time vs. $l$ .....	53
Fig. 4.24 ---- Average Waiting Time vs. $l$ .....	53

# **Chapter 1**

## **Introduction**

One of the most significant technological developments during the last few years has been the emergence of the Internet. With rapid growth of the Internet, it is becoming increasingly difficult to provide the necessary services to all users within a designated time period. As the gap between the network-line and application-server rates is growing, it is getting easier to launch Distributed Denial of Service (DDoS) attacks against services on the Internet, and remain undetected within the network. The end-to-end argument suggests that simple functions that are common to all applications be performed by network computers (e.g., routers), and complex functions required by fewer applications be implemented in end-servers [16]. With hardware performance improving day by day, network line rates tend to go higher, whereas, complex end-server applications and operating system features offset equivalent improvements at the end-system level [9].

During a DDoS attack, a server is repeatedly sent requests from numerous machines, typically called “zombies”, that are controlled by a master process. The master process will trigger a ‘go’ signal to launch an attack against the victim server in hope of flooding the server with an unusually high number of requests, and cause the server to crash. The server is thus unable to process any further requests until further action is taken to restore its state [17].

Extensive work has been done to provide solutions to the DDoS problem at the transport layer and below of an open network, as presented in Chapter 2. However, assuming all attacks at and below the transport layer are taken care of, the threat of a potential attack against publicly accessible application services still remains imminent. The main reason for this threat is the exceeding demand for services for the same server capacity, and lower server throughput as compared to the network line rate, thus making the victims more susceptible to an attack. The flooding-based attacks of February 2000 against the public service of Yahoo!, Ebay, and E\*trade, as well as the January 2001 attacks against Microsoft’s name servers had statistics that clearly showed no unusual network traffic, however the servers were incapacitated as the service demand exceeded their respective capacities [15].

The DDoS attacks against the root DNS servers during October 2002 were launched simultaneously from various attacking points on the Internet, and targeted all the thirteen root DNS servers. Only four of them withstood the attack. The attack lasted for about an hour, during which DNS was disabled. The financial losses incurred due to such attacks can be very high, as all servers, including e-commerce servers, frequently rely on root DNS services for timely completion of transactions [17].

### **Gligor's Rate Control Scheme**

The rate control scheme proposed by Gligor in [9] is a novel mechanism for providing access guarantees to clients for accessing public services, by generating and enforcing simple user-level agreements on dedicated special purpose servers. These servers cannot be flooded, as they operate at the peak network line rate of the front-end network access points (e.g. edge routers). The scheme also uses the CAPTCHA[23] technique, which is a reverse Turing test for controlling the client proliferation on adversary-controlled machines, but only to decrease the waiting time to service for legitimate clients.

When active (during peak traffic), an exception is raised by the server's request Verifier, directing the client's proxy to a special purpose server, called the rate control server (RCS), to obtain a valid ticket containing a

time window during which its request will be processed, and an access count (*wopt*) specifying the number of accesses allowed. The client then has to approach a ticket Verifier (dedicated server) that checks the validity of all requests, and mediates access to the server. In addition to checking the ticket validity, the Verifier also keeps track of the number of times the client has already accessed the service during the current time slot, so as to confirm client eligibility for service access. Within a particular time window, per client information regarding the number of times a client has already visited the server is kept, in order to enforce the agreement (time window, number of accesses) initially made between the client and the server [9].

The rate control scheme explained in detail in Chapter 3 thus controls the client request rate to the application server, thwarting the chances of a flooding attack. In addition, flash crowds (unusually high pikes in traffic during peak hours caused by legitimate clients) are also taken care of by the scheme, thus not letting the server be overwhelmed by requests at any given time [9].

### **Contributions of the Thesis**

Simulations were carried out to analyze the performance of the rate control scheme when applied to two classes of servers, namely, Content Distri-

bution Networks (CDNs), and Domain Name Server (DNS)-based networks. The experiment consisted of simulating a large network with parameters obtained from statistics of these networks, and analyzing the server utilization and client waiting times. In addition, server behavior for varying client populations was also studied.

The simulation experiment confirmed our expectations in the following three areas:

1. Variations in the number of clients affects the client waiting time; e.g., small request-interarrival times during an attack suggest that clients arrive to the RCS at about the same time, and are provided with server accesses within time windows well ahead into the future. Simulation results showed that the average waiting time varies proportionally with increase in the number of clients.

2. Increases in the maximum inter-request time between two consecutive requests to the application server by the same client ( $\delta_r$ ) leads to higher waiting times for clients. Results obtained from the simulation confirm the expected behavior, namely that higher values of  $\delta_r$  resulted in higher waiting times as compared to lower values. The results also helped us place a bound on the value of  $\delta_r$  for the experimented servers namely, CDN and DNS, so as to provide clients with more reasonable maximum waiting times to service.

3. The average number of accesses for a protocol,  $A_r$ , also has a direct impact on the server utilization, with higher values leading to lower server utilization as compared to lower values. The results obtained from the simulations confirmed that with higher  $A_r$ , server utilization is low, as compared to lower values, as is explained in Chapter 4.

### **Thesis Outline**

Chapter 2 reviews prior work in the area of DDoS, and gives a brief explanation of schemes that have been proposed to solve this problem at various levels. Chapter 3 explains the detailed working of the rate control scheme. The analysis of the results obtained when the rate control scheme is implemented in two different server networks, namely, CDN and DNS, is given in Chapters 4, 5. Concluding remarks, with future directions for research are given in Chapter 6.



## **Chapter 2**

### **Related Work**

Denial of Service (DoS) attacks aim to deny clients access to service provided by the victim (server, router, or the network). Attackers either exploit weaknesses in the system, for which patches are later issued upon discovery of the attack, or the victim is forced to undertake computationally intensive tasks, such as exponentiation with large integers for Diffie-Hellman key exchanges [6].

In contrast, flooding-based attacks, do not rely on any particular network or system weaknesses. Instead, they tend to exploit the asymmetry that exists between the network and the victim by amassing a large clan of hosts to simultaneously send useless packets towards the victim, leading to a flood of requests at the victim's end. The intensity of the traffic is high enough to jam or crash either the victim, or its network. Launching a flooding attack has become relatively easy today owing to the free availability of a number of tools for carrying out such attacks, such as Trinoo, Trib Flood

Network 2000, and Stacheldraht. These tools allow the attacking host to install patches of the attack program on innocent agents, aka. “zombies”. The program is tuned to launch an attack against a particular victim at a particular time. Thus, the victim is flooded with requests coming in from all directions at an enormously high magnitude [6][22].

Broadly speaking, DDoS attacks can be classified into two categories:

1. **Direct Attacks:** In a direct attack, the attacker arranges to send a large number of attack packets directly to the victim. SYN flooding is the most common attack case, in which TCP SYN packets are sent to the victim’s server port. The victim will respond by sending back a SYN-ACK response to the source address of the packet. Since the source address of the packet was spoofed, the victim will not receive the third message of the 3-way handshake required for connection establishment in TCP. Thus the number of half open connections at the victim’s end consume all the available memory, forcing the victim to deny service to subsequent clients (including legitimate clients) [6].

2. **Reflector Attacks:** In a reflector attack, intermediate nodes (reflectors), are used as innocent attack launchers. The attacker sends packets with source addresses set to the victim’s address. Without realizing that the packets had spoofed source addresses, the reflectors send the response to

the requests to the victim. As a result, the victim's link is flooded with responses to reflected packets [6].

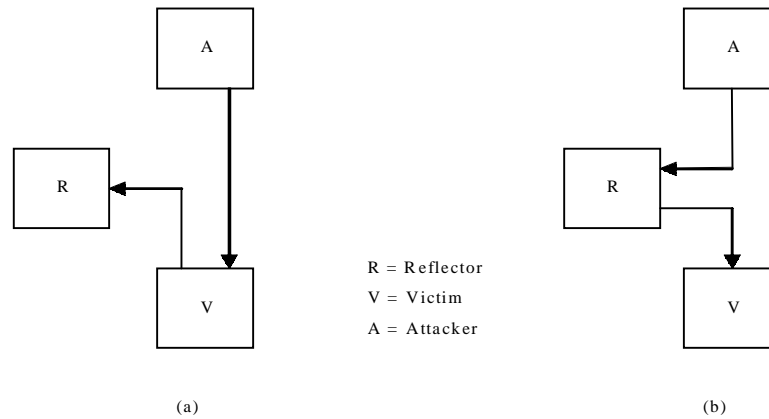


Figure 2.1: Flooding-based DDoS attacks: a) direct b) reflector. [6]

As can be seen from the two types of attacks depicted above, the attacker manages to use spoofed network addresses to flood the victim with useless packets. The solution to this problem is to place routers with capabilities of filtering packets launched from within their local networks, with spoofed IP addresses, and track down potential attackers.

### Filtering-based approach

In [6], Chang proposes a 3-tier approach for tackling the DDoS problem, namely:

- a) Attack prevention and preemption (before the attack).
- b) Attack detection and filtering (during the attack).

c) Attack source traceback and identification (during and after the attack).

The author goes on to explain that attack preemption can be done by ensuring that hosts are secured against master and agent implants, that may secretly involve the host into the attack. Attack must be detected, and IP traceback must be done in order to discover the attack sources. After identifying the attack sources, appropriate filtering must be done in order to scan and rid the network of attack packets. However, it is not guaranteed that all packets dropped were attack packets, and in the process legitimate users may be denied service.

### **Dedicated Application-based Detection Approach**

In [7], Elliott suggests host-specific security agents to be installed in hosts on different platforms, to ensure prevention of a local system from becoming a zombie agent. The proactive security agent automatically audits systems, continually finding problems, and fixing them. A security agent must be designated in an organization, who regularly takes the fingerprint of the host machine, and ensures that the key system files haven't changed. If any system changes have been made by the attacker, the auditor is authorized to fix the application which was either newly installed, or an existing application was altered.

In [11], Kashiwa et al. suggest an active shaping-based approach for tackling the DDoS problem. In their method, program modules called Active Components (ACs) are loaded into the network nodes, which may be routers, to implement application-level functions to detect, backtrack, and defend against attacks at the network level. They suggest an algorithm for detection of the attack, which heavily relies on traffic characteristics before taking any decisions. The AC watches the amount of traffic during a given time period, and if it exceeds the throughput threshold, it concludes that an attack is in progress, and creates suspicious signatures for the “attack” packets. The attack packets are classified either by the front-end router of the attacker, which figures out malicious packets by looking at the spoofed source address, or by the local AC, which looks at unusually high traffic received from specific hosts. These hosts are blacklisted, and further requests from them are considered to be a part of the attack, and thus dropped.

One of the main areas of concern for this approach is the probability of legitimate packets being dropped. These packets may be arising from clients, who are unknowingly involved in a flash crowd at the server end, and thus may be denied service because of the false assumptions made by the AC.

## Dedicated Network-based Detection Approach

In [22], Weiler proposes a honeypot mechanism to lure the attackers into a fantasy world, considered to be a honeypot, which is a mock network, while protecting the actual network behind a firewall. This is a two-pronged approach; Firstly, to defend the operational network from a DDoS attack, Secondly, to trap the attacker for possible legal action against him/her.

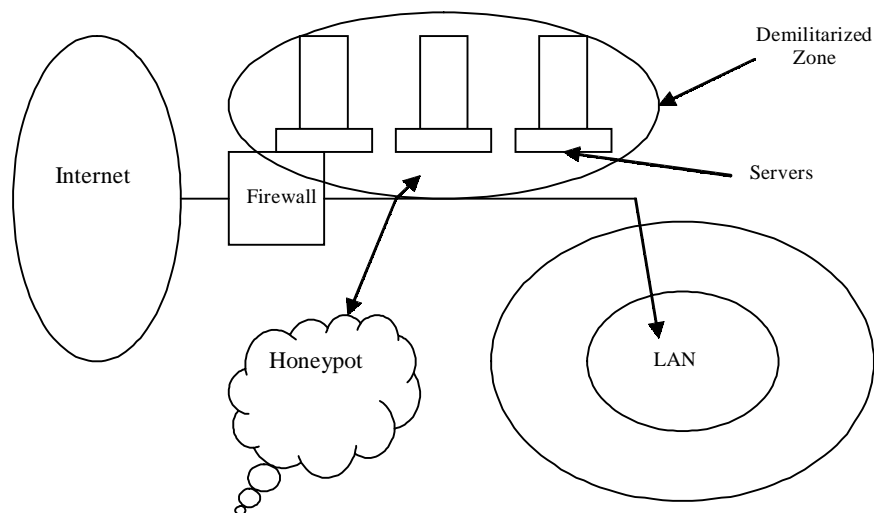


Figure 2.2: Honeypots for protection against DDoS attacks

As can be seen in the above figure, the attacker is lured by the honeypot, and is made to believe that he has successfully infiltrated and compromised an actual client to become a slave, however in reality, he's gotten himself into a trap, and can be traced. Services, such as FTP, Email, HTTP, are situ-

ated in a “demilitarized” zone, and can be accessed from the outside world. The local network is in another zone, protected by a firewall, which is regularly updated. Client signatures are employed to detect an attack, and forward subsequent requests to the honeypot rather than the actual clients [22].

The scheme proposes a novel trap for attackers, but doesn’t provide any mechanism of guaranteeing that clients that are considered to be attackers are actually so, and thus there exists a non-zero probability of denying service to legitimate clients.

### **Anomaly Detection**

Management Information Base (MIB) traffic variables were used to study anomalies in traffic patterns, and detect attacks in progress in [5]. These variables are regularly observed for unusual changes in their values at the Network Monitoring System (NMS) level. Unusual patterns in traffic are considered as attacks in progress, and necessary action is taken to prevent the server from being flooded.

The decision as to whether a particular flow is an attack or not cannot be taken at the network level, as the clients are not aware of the secret filtering policies, as well as upper limit rates at which, say, ping packets can be sent

to the front-end router before crashing it. Thus anomaly detection cannot be considered as a strong solution to the DDoS attack problem.

### **Client-Puzzle based Service Guarantees**

Client puzzles require that each client solve a puzzle as proof of work to accompany its request to the server. The server decides whether to process the clients request or not only after receiving the appropriate proof of work. The strength  $k$  of the puzzle is either determined by the client or by the server depending on the scheme. Certain servers may preempt queued requests from clients that solved simpler puzzles, with requests from clients accompanying solutions to more complex puzzles [21]. The server scheduler checks the puzzle solutions at the network-line rate. Client requests that either solved the puzzle incorrectly, or not at all, are dropped. In spite of these drops, if the client-request arrival rate is still high at puzzle level  $k$ , the server drops the extra requests, and expects clients whose requests were dropped to bid with a higher-strength puzzle, say  $k+1$  [9].

Typical client puzzles use crypto-hash functions, where the output of the hash function is between 128 and 160 bits for  $k$  between 1 and 64 bits. Thus, the puzzle computation cost to the client is exponential in  $k$ . In [2], the client challenge puzzle is to find a hash function output with  $k$  consecutive zeroes in the high-order bits.



Puzzles have the advantage of being stateless, as the server does not have to store any per-client information locally for deciding to give access to the clients, however, they are ineffective in the role of user agreements for preventing DDoS attacks, as they combine weak service-access guarantees with high request overheads. There is no way of distinguishing between good and bad clients based on the same puzzle difficulty level, and there is a weak guarantee that in spite of solving a series of puzzles with increasing difficulty levels, a client may be provided with service [9].

In addition, when adversaries with unknown computation power are present in the open network, client puzzles do not strongly guarantee access to legitimate clients even after say  $r$  retries with varying levels of puzzle difficulty. As can be seen from above, client puzzles do not provide strong access guarantees to legitimate clients during the event of a DDoS attack.

## Chapter 3

### Rate Control Scheme with Maximum Waiting Time Guarantees

The rate control service (RCS) simulated in this thesis is application-specific, and ensures that the aggregate rate of request generation of the total client population does not exceed the maximum processing rate of the application server, given by  $L/\tau = S$ , during any time interval  $\tau$  or larger, where  $L$  is the queue length at the application server, and  $S$  is the application server processing rate (requests/sec). When the rate control scheme is in operation during heavy traffic periods, clients have to obtain a valid ticket from the RCS in order to access the application service either once, or multiple times within a single time window, depending on the type of service (e.g. Authentication, Naming, Email) being accessed. Clients are allowed to place their respective requests within these time windows, and are guaranteed a maximum waiting time to service within the upper limit of the window [9].

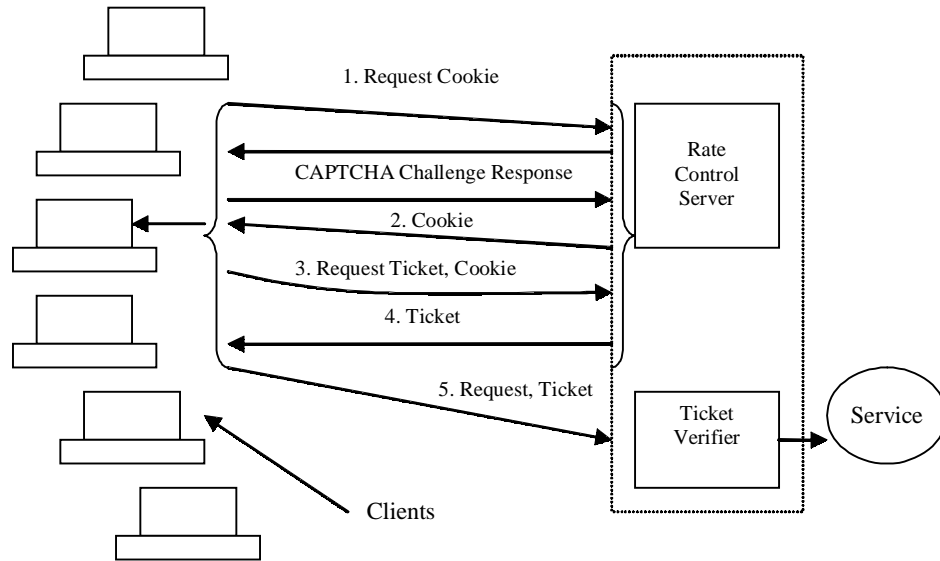


Figure 3.1: Rate Control Scheme

### **Ticket Issuance**

A client request for a ticket contains the following parameters - number of accesses desired, the source IP address from which the requests will be issued, the start time of the window in which the requests will be issued,  $t_s$ , the number of accesses desired, and the maximum interval between two consecutive requests,  $\delta_r$ , if the client wants to access the service multiple number of times. The RCS verifies that the number of accesses desired and  $\delta_r$  are consistent with the server-access protocol, and that  $t_s$  is within the ticket postdating time allowed, so that tickets with requested start times very long into the future are not issued. If these checks pass, the server

issues the ticket, and a *message authentication code* (MAC) accompanying it. [9]

The ticket contains the following parameters:

(1) Start time ( $t_i$ ); (2) End time ( $t_{i+1}$ ); (3) Maximum number of accesses,  $wopt$ ; (4) The source IP address for the request; (5) Time of ticket issue ( $t_{RCS}$ ). The start time is set to be  $t_i = t_w + \Delta$ , where  $t_w > t_s$  is the first time window available at the application server for issuance of a request. The time of ticket issue at the RCS,  $t_{RCS}$ , allows the client to synchronize with the time at the verifier. The communication delay  $\Delta$  ensures that the ticket is valid upon receipt by the client, and  $t_w > t_s$  ensures that the client has time to issue a request. The window end time is given by:  $t_{i+1} = t_i + wopt (\tau + 2\Delta + \delta_r)$ , where the network delay  $\Delta$  is for the client request to reach the verifier, for request processing in the worst case time period of  $\tau$ , and for ticket validity before the next access,  $\Delta + \delta_r$ . The verifier maintains a cache of tickets seen within the current time window, and the number of accesses already availed by each ticket, to strictly implement the access agreement made earlier with the clients.

### **Ticket Usage and Integrity**

Upon receiving the ticket, clients may send their requests to the ticket verifier along with their tickets for verification purposes, and if verification

is successful, their requests are forwarded to the application service. The verifier usually sits between the front-end router and the application server in the server network, and is time synchronized with the RCS. Both the verifier and the RCS share a symmetric key. The RCS uses the key to generate MAC for each ticket, and the verifier uses the key to verify the authenticity of the ticket. The MAC ensures that the ticket integrity is maintained, and that it is not tampered with on the way.

The computation of the MAC could be done in many ways using the shared secret key, and thus it is not possible for anyone without the knowledge of the key to compute the correct MAC. In order to manipulate the values or parameters in the ticket to increase the number of accesses, or to change the source IP address given in the ticket, the MAC has to be recomputed with the correct shared secret key, and since only the verifier and the RCS have access to the secret key, no third party can compute a new MAC with the same shared key. Therefore, any modification to the ticket is easily detectable at the ticket verifier by the verification of the MAC accompanying the ticket and the request. MAC computation is the most time consuming task performed, however, it can be performed in parallel, at rates much faster than the network line rate. In addition, the size of the ticket is very small ( $< 1$  KB), and thus the computation will not take much time.

## Session Cookie

The RCS and the verifier ensure that the aggregate request rate doesn't exceed the server's throughput, by issuing tickets in accordance with the server processing rate, however, an adversary can start a large number of clients on a number of different machines to obtain valid tickets, and either abstain from placing their respective requests in the allotted time slots to lead to an underutilization of resources at the server end, or to push legitimate clients further off into the future before service is provided to them, thus increasing their MWT beyond reasonable values. In order to prevent uncontrolled client proliferation by an adversary, the scheme requires that each ticket request from a client be accompanied by a cryptographic cookie attesting that the client has a human user behind it. The client must pass the reverse Turing test (or CAPTCHA [1][9]) in order to prove so and obtain a cookie, similar in structure to a ticket, and containing the following: (1) start time; (2) end time; (3) list of IP addresses from which ticket requests can be issued; (4)  $t_{RCS}$ ; (5) MAC for the cookie. The time window of the cookie is ideally equivalent to a login session, and thus the reverse Turing test is required only once at the beginning of the session [9].

### How many accesses to give?

The number of accesses to be provided to a client during a time window,  $w_{opt}$ , has a significant impact on both the performance of the system, as well as the client perceived waiting time. If a single access is allowed, the communication cost for the clients increases owing to the more number of visits to the RCS for tickets. In contrast, if all accesses are given within a single window, unused tickets by adversary's clients could decrease server utilization due to reserved but unused time windows (underutilization attack). The optimal window size is computed as a tradeoff between the server under-utilization and the number of requests to the RCS [9]. Letting  $c_1$  to be the unit cost of a round trip to the RCS,  $c_2$  the unit cost of lost server utilization due to abstinence from placing requests by illegitimate clients,  $A_r$  the access count per application, and  $l$  the percentage of legitimate clients in the system,  $0 \leq l < 1$ , the optimal window size in terms of the access count can be computed as a minimization of the total cost:

$$C_{total} = C_{client} + C_{server} = c_1 A_r / w_{opt} + c_2 (1-l) w_{opt}.$$

Setting  $\frac{d}{dw} C_{total} = 0$ , the optimal window size is given by:

$$w_{opt} = \sqrt{\frac{c_1 A_r}{c_2 (1-l)}}$$

where,

$$1 \leq w_{opt} \leq \min(A_r, L)$$

### **Wopt and its significance**

As can be seen from the formula given above to compute  $wopt$ , the value of  $wopt$  increases considerably with increasing value of  $l$ , for fixed  $A$ ;  $c1/c2$ . There are four different combinations of  $wopt$  and interarrival times ( $t$ ), that have varying implications in the study:

1. High  $l$  and High  $t$  imply greater percentage of legitimate clients, arriving after considerably long intervals of time (not a flash crowd).
2. High  $l$  and Low  $t$  imply a Flash Crowd of legitimate clients arriving at very short spans of time.
3. Low  $l$  and Low  $t$  imply a Distributed Denial of Service (DDoS) attack, with greater percentage of illegitimate clients, arriving at shorter intervals of time, in order to flood the server, and incapacitate it from serving legitimate clients.
4. Low  $l$  and High  $t$  imply a greater percentage of illegitimate clients, arriving after longer spans in time (not a DDoS attack).



## **Chapter 4**

### **Simulation Analysis**

#### **4.1 Rate Control Scheme applied to CDNs**

Content Distribution Networks (CDNs) are widely popular distributed systems on the Internet that distribute client requests to an appropriate server based on a number of factors; *viz.*, server load, network proximity, cache locality, so as to minimize the load on the system, and to reduce the client perceived response time (latency). With exponential growth in the usage of the Internet and a lack of proportional growth of server resources, resources tend to get exhausted more often, and are more vulnerable to flooding-based attacks, such as DDoS. Even if a system is not under attack, it may be that the server resources are exhausted due to “flash crowds”, which may be caused by lots of legitimate clients who unknowingly place their requests at very short time intervals, thus flooding the server, and bringing it down to an irrecoverable state.

### **4.1.1 Working of CDNs**

Content Distribution Networks (CDNs), geographically distribute server surrogates that cache pages, instead of placing them all within the same subnet. Thus, a client requesting the same page twice may be led to a different server each time. The aim of this content distribution is to reduce the client perceived latencies, by redirecting clients to appropriate servers based on their geographical locations, server surrogate load, and other factors, which may include priority to important clients. Several algorithms were proposed [20] for deciding the distribution of client requests. Some of them are:

1. Modulo Hashing: The URL is hashed to a number modulo the number of servers. The resultant value is the server number, which is given to the client.
2. Consistent Hashing: The URL is hashed to a number in a large, circular space, as are the names of the servers. The URL is assigned to the server that lies closest on the circle on its hash value. If a server node fails, the load is shifted to its neighbors.
3. Highest Random Weight: A list is generated by hashing the URL and the server's name, and sorting the results. Each URL then has a determinis-

tic order to access the set of servers, and this list is traversed until a suitably loaded server is found.

4. Dynamic Replication with Network Proximity: The effective load on a server is multiplied with the distance between the client and the closest server, and the appropriate server is selected to provide service to the client.

The average number of requests per second handled by a typical CDN server is 600 [20].

Considering the wide ranging impact that a DDoS attack can have on a CDN network, owing to the extent of usage of such a network, we decided to run simulations by implementing the rate control scheme described earlier to CDNs, with parameters closely resembling many CDNs widely deployed on the Internet today.

### **4.1.2 Simulation Experiment**

Simulations were carried out to analyze the performance of the server, and the client waiting times, when the rate control scheme is implemented in a CDN. The front-end router processing rate operates at the ticket generation + processing rate of the rate control server, so as not to flood the rate control server at any time. The simulator was written in C, and was run for different client populations, with exponential traffic arrival rate to the rate control server.

## Assumptions

The following assumptions were made for the simulations:

1. The counter values ( $w_{opt}$ ) given to individual clients based upon their requests, were decided as follows:

$$1 \leq w_{opt} \leq \min(Ar, L), \text{ where, } w_{opt} = \sqrt{\frac{c1 \cdot Ar}{c2 \cdot (1-l)}}$$

$c1$ : unit cost of communication = 200 ms

$c2$ : unit cost of computation = 16, 1000 ms

$Ar$ : access count per application = 6-60

$l$ : percentage of legitimate clients in the system

$\delta_r$ : maximum inter-request delay requested by a client = 3-30 ms [14]

2. Clients were assumed to have independent, non-overlapping windows of time at the Application Server, during which they may place their requests.

3. Time window computation was done as follows:

$$t_i = t_w + \Delta$$

$$t_{i+1} = t_i + w(\tau + 2\Delta + \delta_r)$$

4. The traffic arrival process at the RCS was poisson, with exponential interarrival times.

5. The Verifier rejects requests that are not eligible to fall within the current time window.

6. Server Rate = 600 reqs/sec [20];  $\Delta$  = 200 msec [24][25][26];  $L$ = 1024;

$$\tau = L/S$$

The unit costs of communication, computation, namely,  $c_1$  and  $c_2$ , were taken as network communication latency to the RCS, and computation delay at the application server, respectively. For CDNs, the communication latency is on average 200 ms [24][25][26], and the computation latency may range from 200 ms to 1000 ms [27][28][29]. The value of  $A_r$ , which is the average access count required by a client per application was taken at two boundary values: 6, 60 [30][31][32][33]. Usually clients have varying request patterns, but on average very few clients exceed sixty accesses to the CDN server during any session. The interarrival time ( $t$ ), was taken to be in the range 0.01 ms to 200 ms, where 0.01 ms is the case of a typical DDoS attack [34][35], during which attempts are made to fully flood the server with large number of requests originating at short spans of time.  $t=200$  ms is the typical interarrival time to the server during normal operation.

Note: For  $t=200$  ms, the rate control scheme is not required, as requests are coming in at a rate lower than maximum server rate.

### **4.1.3 Results and Observations**

Due to the randomness in the arrival process to the RCS, 100 samples were taken at each plot value, and a 95% confidence interval was built at each point on the plot. Assuming that the sample mean of  $n$  ( $=100$ ) obser-

variations is  $\bar{Y}$ , the random variable  $\bar{Y}$  is normalized by the transformation:  $Z = \frac{(\bar{Y} - \mu)\sqrt{n}}{\sigma}$ , where,  $\sigma$  is the population variance computed for the different plot values.  $Z$  has a standard normal distribution, and by letting  $z_{\alpha/2}$  denote the upper  $\alpha/2 \times 100$  percentile of the standard normal distribution, where  $\alpha = 0.05$ , we obtain:

$$P\left[\bar{Y} - z_{0.5\alpha} \cdot \frac{\sigma}{\sqrt{n}} \leq \mu \leq \bar{Y} + z_{0.5\alpha} \cdot \frac{\sigma}{\sqrt{n}}\right] = 1 - \alpha, \text{ where}$$

the random interval  $\bar{Y} \pm z_{0.5\alpha} \cdot \frac{\sigma}{\sqrt{n}}$  is the confidence interval, and  $1 - \alpha$  is the confidence level. For the experiment, we took the value of the confidence interval to be 0.95 i.e. we are 95% confident that the actual mean lies in the confidence interval calculated, for which  $z_{0.025} = 1.96$  [13].

Due to the deterministic nature of the server utilization and the waiting times, and owing to its direct dependence on the parameter values, in particular on the value of  $l$ , the confidence intervals turned out to be at a small range of less than 1% deviation from the mean values, and thus did not overlap, as can be seen in the plotted graphs.

### Utilization

1. The application server utilization was observed to be lower (~7%) for  $\delta_r = 30$  seconds, as compared to the case with  $\delta_r = 3$  seconds, where the utilization is almost 50% for larger values of  $l$ , as can be seen in Figures 4.1, 4.2, 4.3 and 4.4. This behavior is caused by the fact that the value of  $\delta_r$  has

a direct relation with the time window size. Thus, increasing values of  $\delta_r$  lead to larger time windows for the clients to place their requests in, and considering the fact that the number of accesses ( $wopt$ ), is the same for both the cases, the server utilization went down for increasing  $\delta_r$ .

This phenomenon can be verified from Figures 4.1, 4.2, 4.3 and 4.4, where the server utilization is steadily increasing for increasing values of  $l$ , and is better for  $\delta_r=3$  seconds, than for  $\delta_r=30$  seconds. For increasing values of  $l$ , the  $wopt$  value increases, and since increasing  $l$  implies increase in the population of legitimate users, fewer users ( $1-l$ ), abstain from placing their requests in the allotted time slots for causing an underutilization attack against the server resources. Thus, the server utilization steadily increases with increase in the value of  $l$ .

2. During the event of a “flash crowd” (i.e., high  $l$  and low  $t$ ), the utilization of the server was roughly 50-70%, as can be seen in Figures 4.1, 4.2, 4.3 and 4.4. This shows that the rate control scheme never allows the demand to the application server to exceed capacity at any time, and at the same time ensures reasonably good utilization. In this case, the high  $l$  and low  $t$  imply majority legitimate clients, who actually place their respective requests during the allotted time slots, and arrive at the server at short time interarrivals.

3. A DDoS attack (i.e., low  $l$  and low  $t$ ) against the server is defined as an underutilization attack, which may occur when a number of illegitimate clients request for tickets to the rate control server, and when provided with tickets, abstain from utilizing their respective time slots at the application server. Owing to this, the application server remains underutilized during those particular time slots, and hence the server utilization drops. The rate control scheme adjusts to this case by reducing the value of  $w_{opt}$ , and hence reducing the overall time at the server, during which it remains idle due to the attack. As we can see from Figures 4.1, 4.2, 4.3 and 4.4, the utilization of the server remains around 5% even during the case where  $l=0.1$  (90% of the clients are illegitimate), thus showing that the attackers do not fully succeed in their attempt to cause an underutilization attack.

### **Average Waiting Time**

The average waiting time perceived by the clients is directly proportional to the value of  $\delta_r$ , with higher values of  $\delta_r$  leading to higher waiting times, and vice versa. The individual time windows assigned to the clients increase in size with increasing value of  $\delta_r$ , thus pushing subsequent clients further off into the future before service is provided to them. Therefore, as can be seen in Figures 4.5, 4.6, 4.7 and 4.8, the waiting time is very high for  $\delta_r=30$  seconds, and much lower for  $\delta_r=3$  seconds.



For increasing values of  $Ar$ , the waiting time increases as well, and again this is due to the direct proportionality of the value of  $Ar$  to the value of  $w_{opt}$ , with higher  $Ar$  leading to larger optimal window sizes (accesses), and thus larger time windows, thus in turn pushing subsequent clients further off into the future before service is provided to them.

The following observations were made from the results:

1. For  $c1/c2 = 200/16$ , the waiting time was around 10-40 seconds for  $\delta_r=3$  sec,  $Ar=6$ , and for  $Ar=60$  with other parameters remaining the same, the waiting time went up to 100-900 seconds. Varying value of  $Ar$  has a significant impact on the per-client average waiting time. This is because for larger values of  $Ar$  (60 in this case), the per-client accesses provided are higher, and thus larger time windows are reserved for clients at the application server; consequently, subsequent clients have to wait for longer time periods, before being provided service.

2. Increasing value of  $\delta_r$  also has a significant impact on the waiting time, with higher values of  $\delta_r$  leading to higher waiting times compared to lower values. This behavior is due to the fact that the time window provided to clients increases with increase in the value of  $\delta_r$ , and thus larger time windows are provided to clients to place their requests in, and hence subsequent clients have to wait for longer before service is provided to them.

3. For  $c1/c2 = 200/1000$ , the waiting time is lower, as compared to  $c1/c2 = 200/16$ , as can be seen in Figures 4.5, 4.6, 4.7 and 4.8. Higher value of  $c2$  implies that server side computation is more expensive than the communication delay to the rate control server, and hence it can prove expensive to lose it. The  $wopt$  (accesses) value provided to clients is lower for  $c2=1000$ , as compared to  $c2=16$ , due to the inverse proportionality between  $wopt$  and  $c2$ , as can be seen from the formula for computation of  $wopt$ . Thus, for higher  $c2$  ( $=1000$  in this case), the  $wopt$  value is lower, and hence clients are provided with smaller time windows for placing their respective requests, and therefore, subsequent clients do not have to wait for long before service is provided to them.

4. During a DDoS attack (i.e., low  $l$  and low  $t=0.01$ , 1 ms), as can be seen in Figures 4.5, 4.6, 4.7 and 4.8, the average waiting time is around 100 seconds for  $c1/c2=200/16$ ,  $\delta_r=3$  sec, and  $Ar=60$ , and is around 20 seconds for  $c1/c2=200/1000$ . The waiting time is even better for the case with  $Ar=6$ , with  $c1/c2=200/16$  giving a waiting time of around 15 seconds, and  $c1/c2=200/1000$  giving a waiting time of 10 seconds. This shows that for systems where the waiting time is critical, the cost of computation may be increased beyond the communication cost, or the average number of requests given per client may be brought down to say  $Ar=6$ , rather than having a large value for it. In addition,  $\delta_r \leq 3$  seconds is a reasonable value

for the maximum interrequest delay for a particular client, as in Figures 4.7, 4.8 we can see enormously high waiting times for cases with  $\delta_r=30$  seconds.

5. During the event of a “flash crowd” (i.e., high  $l$  and low  $t$ ), the waiting time is around 600-800 seconds for  $c1/c2=200/16$ ,  $\delta_r=3$  seconds,  $Ar=60$ , and is around 50-90 seconds for the same parameters, but for  $Ar=6$ , as can be seen in Figure 4.5. For the case with  $c1/c2=200/1000$ ,  $\delta_r=3$  seconds,  $Ar=60$ , the waiting time is around 100-180 seconds, and for the same parameters but with  $Ar=6$ , the waiting time is around 50 seconds. This result further strengthens our argument for placing an upper limit on the value of  $Ar$ , used for computation of the optimal window size ( $w_{opt}$ ), with higher  $Ar$  leading to very high waiting times, and lower values providing reasonable waiting time guarantees to the clients, before actual service is provided to them. Again, for the case with  $\delta_r=30$  seconds, we have obtained very high values for the waiting times, and thus it may not be considered as an implementation case.

### **Effect of Variations of Number of Clients on Utilization, Waiting Time**

The server utilization remains the same for variations in the value of  $N$ , the number of clients. This is because neither the time window size, nor the

*wopt* value, that are provided to the individual clients are affected by the client population.

The client waiting time is severely impacted by variations in the client population. Due to the relatively small interrequest delays at the rate control server, we have clients coming in more or less at the same time, and for larger populations, this means that the per-client waiting time goes up with the client population, as is evident from Figures 4.9, 4.10, 4.11 and 4.12, where we have a steady increase in the waiting time with increase in the client population.

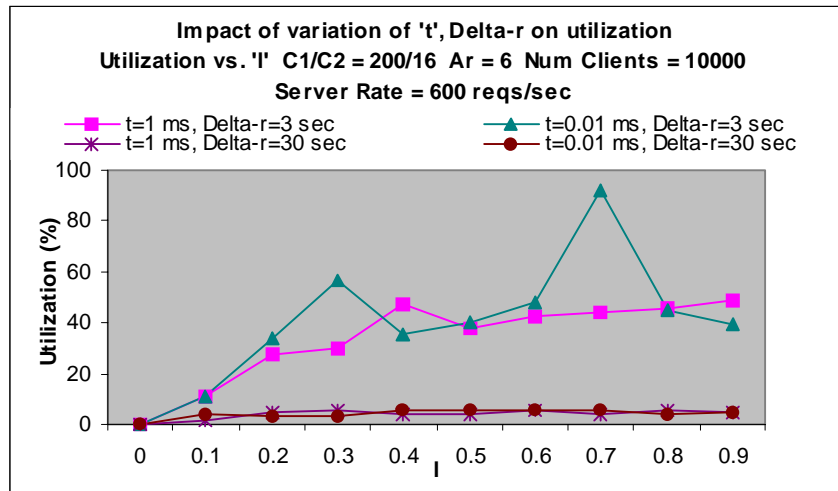


Figure 4.1: Server utilization vs. *I*

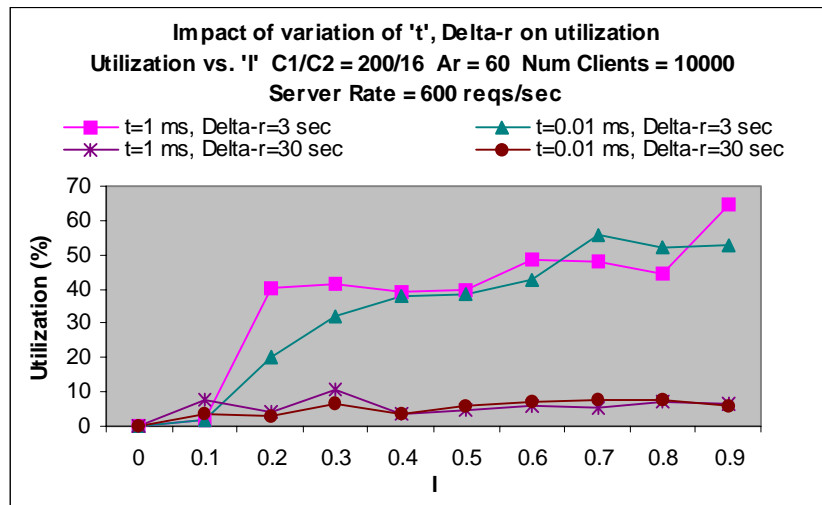


Figure 4.2: Server utilization vs. l

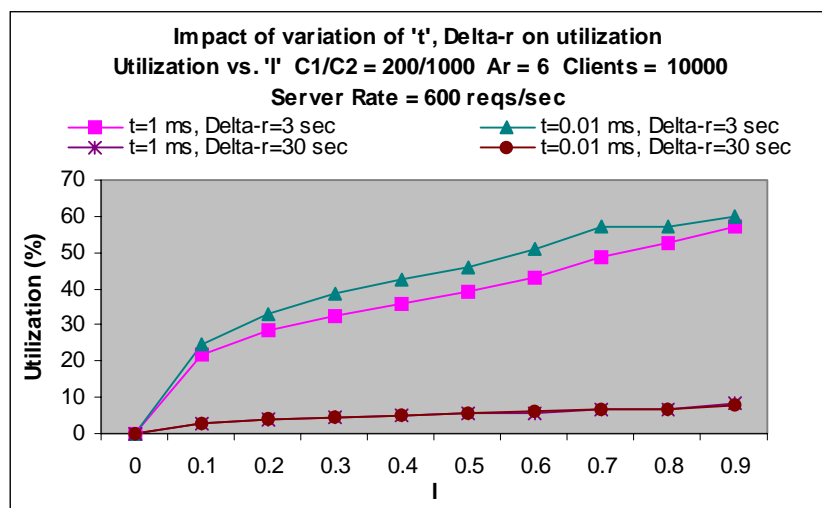


Figure 4.3: Server utilization vs. l

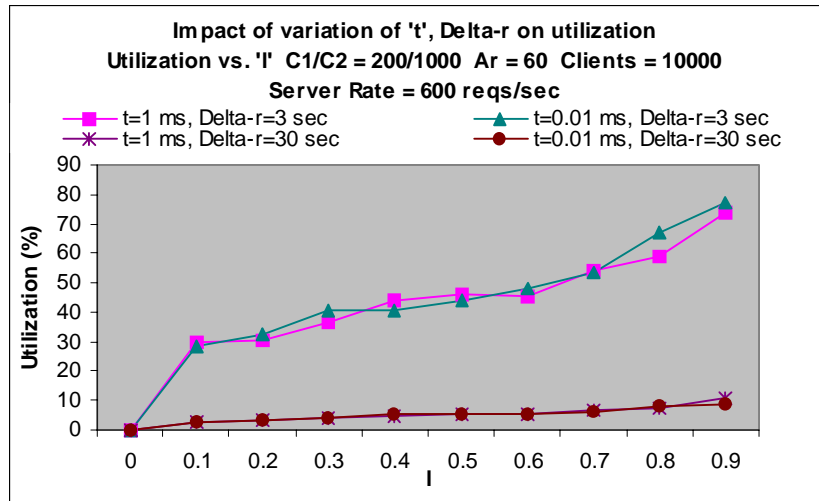


Figure 4.4: Server utilization vs. l

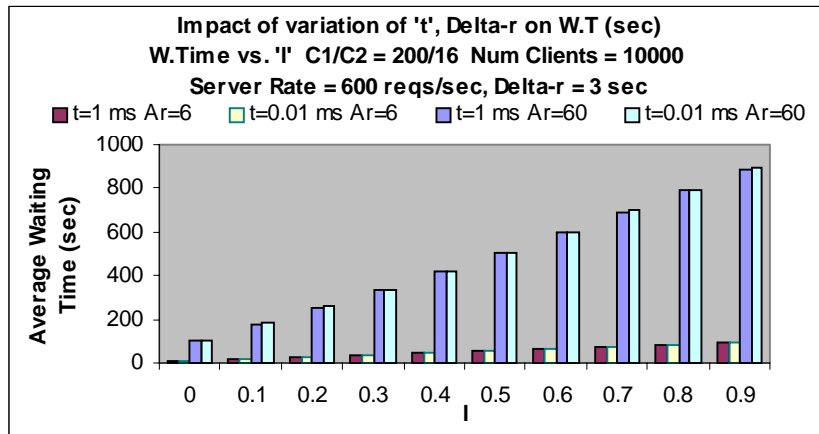


Figure 4.5: Average Waiting Time vs. l

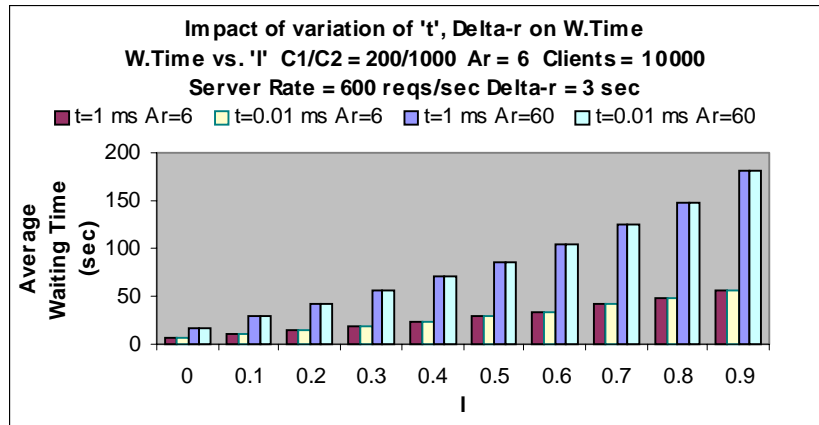


Figure 4.6: Average Waiting Time vs. l

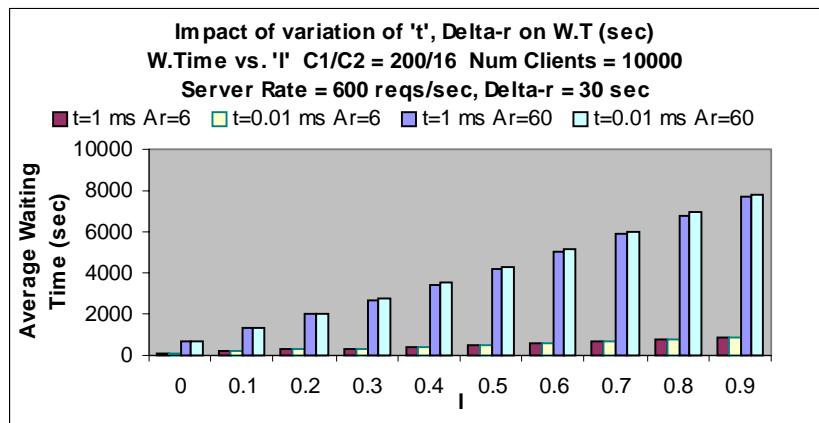


Figure 4.7: Average Waiting Time vs. l

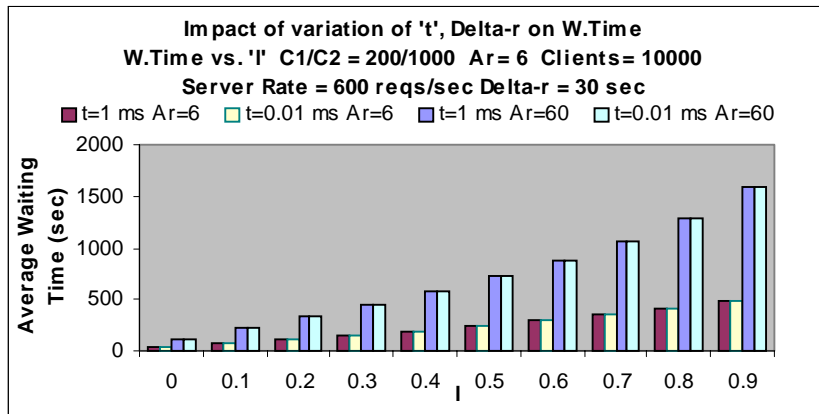


Figure 4.8: Average Waiting Time vs. l

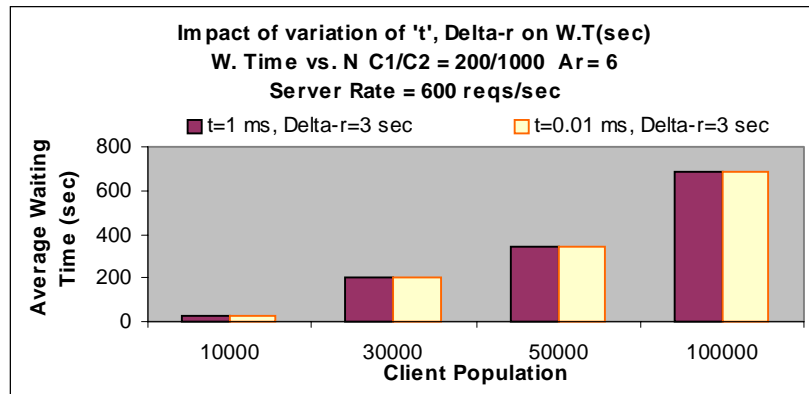


Figure 4.9: Average Waiting Time vs. Client Population

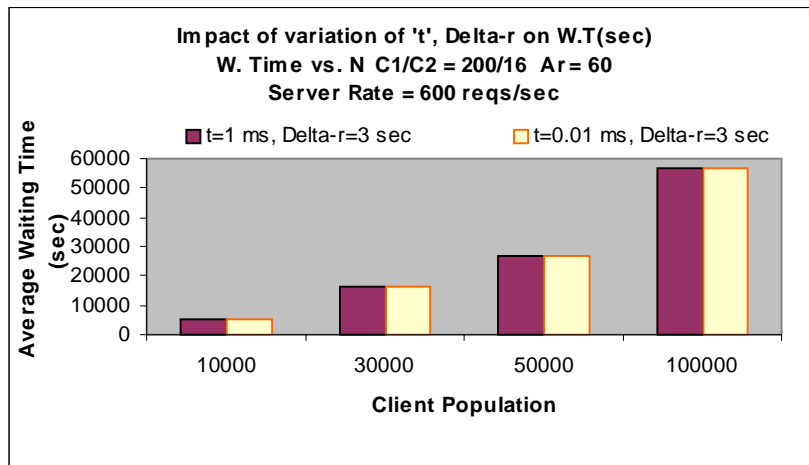


Figure 4.10: Average Waiting Time vs. Client Population

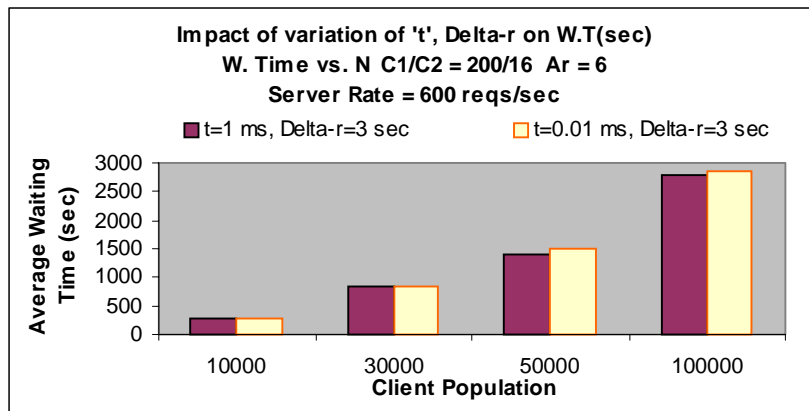


Figure 4.11: Average Waiting Time vs. Client Population



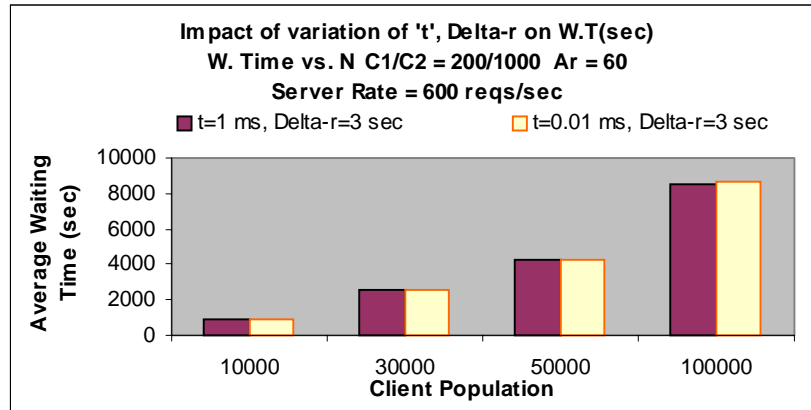


Figure 4.12: Average Waiting Time vs. Client Population

## 4.2 Rate Control Scheme applied to DNS Root Name Servers

The Domain Name System, DNS, translates domain names to IP addresses. The data used for this mapping is stored in a tree-structured distributed database, where each name server is responsible for its portion of the hierarchy. The Root Name Servers are located at the root of this tree, and play a major role in name resolution at the high levels [3][8].

### 4.2.1 DNS Name Resolution Scheme

DNS specifications are most popularly implemented using the Berkeley Internet Name Domain (BIND) software. The process of name resolution is

completely transparent to the end user, however, it may lead to unusually long delays before the user may be given access to the service [8].

Initially, the client (end user application) in a given local area network sends a request for host name resolution to the local name server. The local name server looks up the name in its local cache, if found, returns the address to the client. In the case when the name is not present in the local cache, the local name server recursively follows referrals until it gets an answer. The root of the tree contains the root servers, which are responsible for name resolution of top level domain (.com, .net, .edu etc.) servers.

DNS root name servers are a key center to most activities on the Internet, as local name servers frequently need to update their respective caches with name to IP-address mappings by sending requests to the DNS root name servers. Therefore, if the root name servers come under a DDoS attack, they may be incapacitated from providing regular and timely services to their respective clients, as is evident from the October 21<sup>st</sup>, 2002 attack [17], during which, nine of the thirteen root name servers were temporarily flooded with requests originating from spoofed IP addresses, thus disabling them from providing any further service for about an hour.

Due to the significance of the problem, we decided to carry out experiments examining the performance of the servers, and client waiting times, when the RCS scheme explained earlier, is integrated into the DNS system.

## 4.2.2 Simulation Experiment

We carried out simulations to mimic the behavior of a distributed client-server system, when the Rate Control Scheme was implemented in the system. The simulator was written in C, and was run for different client populations, generating requests for service to the RCS.

### Assumptions

The following assumptions were made before carrying out the simulations:

1. The counter values ( $w_{opt}$ ), which are given to individual clients based upon their requests, are decided as follows:

$$1 \leq w_{opt} \leq \min(Ar, L) \text{ , where, } w_{opt} = \sqrt{\frac{c1 \bullet Ar}{c2 \bullet (1-l)}}$$

c1: unit cost of communication = 1

c2: unit cost of computation = 10

Ar: access count per application = 6 [36][37]

l: percentage of legitimate clients in the system

$\delta_r$ : maximum inter-request delay requested by a client = 90 ms [41].

2. Clients are assumed to have independent, non-overlapping windows of time at the Application Server, during which they may place their requests.

3. Time window computation is done as follows:

$$t_i = t_w + \Delta$$

$$t_{i+1} = t_i + w(\tau + 2\Delta + \delta_r)$$

4. The traffic arrival process at the RCS is poisson, with exponential interarrival times.

5. The Verifier rejects requests that are not eligible to fall within the current time window.

6. Server Rate = 5000 or 12000 reqs/sec [38][39];  $\Delta = 81$  msec [40];  $L = 1024$ ;  $\tau = L/S$

DNS computation service time is more valuable to lose as compared to the communication delay, as there are many mission-critical, as well as real-time operating clients waiting in anticipation of being provided access to the naming service within a guaranteed time period. Therefore, the ratio  $c1/c2$  was selected to be 1/10, implying that the server-side computation in the DNS environment is 10 times more expensive to lose as compared to the communication delay to the RCS. This ratio of the two costs gives a strict lower bound on the value of  $w_{opt}$ , and further changes in the values of  $l$  (percentage of legitimate clients) lead to higher values of  $w_{opt}$ , but not crossing the upper bound given by:  $Min(Ar, L)$ .

### **4.2.3 Results and Observations**

As with the case for the CDNs, the confidence interval for the various plot points was taken at 95%, and the results showed less than 1% deviation from the resulting mean, and thus did not overlap with adjacent curves.

## Utilization

1. For higher values of  $\delta_r = 1\text{sec}$  (the maximum per-client interrequest time), the time window size ( $t_i - t_{i+1}$ ), that was given to the clients, based on the following formulae

$$t_i = t_w + \Delta, \text{ and}$$

$$t_{i+1} = t_i + w(\tau + 2\Delta + \delta_r)$$

was higher, as compared to cases with  $\delta_r = 1\text{ms} - 90\text{ms}$ . The reason being that, for higher  $\delta_r$ , clients are provided with larger time windows to place the same number of requests as in the case with lower  $\delta_r$ , thus leading to a lower utilization at the application server, as is seen in Figures 4.15, 4.16, 4.17 and 4.18.

In Figure 4.13, the upper case shows the situation, where the time window size provided to the clients is larger owing to the higher  $\delta_r$  requested, and the Utilization is  $20/(10 * \text{Server rate})$ . In the lower case, the Utilization is  $20/(5 * \text{Server rate})$ , thus showing the better utilization for lower  $\delta_r$ 's.

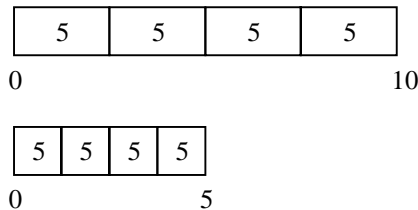


Figure 4.13: Impact of  $\delta_r$  variation on the server utilization

2. During the event of a “flash crowd” (i.e., high  $l$  and low  $t$ ), the utilization of the server was roughly 12% for server rate ( $S$ ) = 5000 reqs/sec, and around 5% for  $S = 12000$  reqs/sec, which are both below the maximum server capacity, thus displaying the working of the rate control mechanism in disallowing any type of flooding that may take place at the application server during peak traffic hours by majority legitimate clients ( $l \sim 1$  implies majority legitimate clients).

Note: The utilization is much higher for lower server rates, as compared to higher rates for the same number of accesses ( $w_{opt}$ ), due to the inverse proportionality between server utilization and the server rate (by definition).

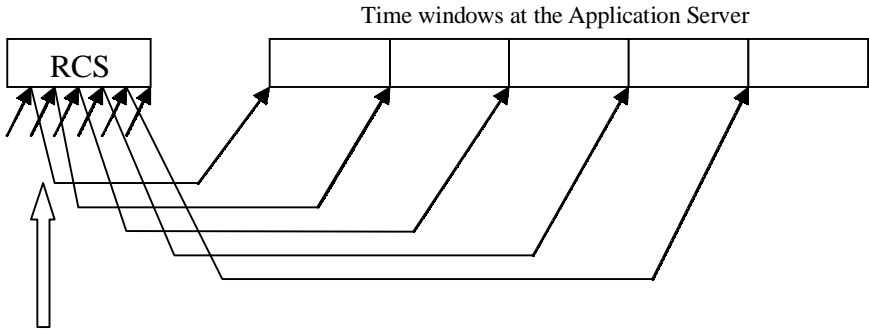
3. During the event of a DDoS attack (i.e., low  $l$  and low  $t$ ), the utilization still remains above zero (seen in Figures 4.15, 4.16, 4.17 and 4.18), as the  $w_{opt}$  size shrinks for lower values of  $l$ , thus disallowing illegitimate clients from being successful in their attempt to cause an under-utilization attack at the server, by abstaining from placing requests in their respective time slots at the application server. The low value of  $w_{opt}$  leads to smaller time window sizes, and thus illegitimate clients, who are trying to launch an underutilization attack will not be successful to the extent they would have been in the case with larger time windows.

The ideal DDoS attack situation is when we have the attackers coming to the RCS with high frequency (low  $l$  with low  $t$ ), and requesting tickets with a very high  $\delta_r$ , thus making the RCS provide time windows much larger than normal. These big time windows will most certainly lead to a severe underutilization at the server, as clients will be provided with the same number of accesses ( $wopt$ ), but with larger time windows, as is seen in Figures 4.15, 4.16, 4.17 and 4.18. The solution to this problem is to place an upper bound on the value of  $\delta_r$ , which the clients may request. From Figures 4.15, 4.16, 4.17, 4.18, it is evident that the utilization is better for  $\delta_r=1\text{ms}$ , and is reasonable for  $\delta_r=90\text{ms}$ , which is the also the usual average that may be requested by clients in the DNS service, as compared to cases with  $\delta_r=1\text{sec}$ . Therefore, the condition  $1 < \delta_r < 90\text{msec}$ , must be placed on the value of  $\delta_r$ , in order to ensure better resistance to the DDoS underutilization attack against DNS root name servers.

### **Average Waiting Time**

The average client waiting time to service is directly proportional to  $\delta_r$ . For higher values of  $\delta_r$ , the time window provided to the clients is larger, and thus subsequent clients are pushed off further into the future before service is provided to them. This higher waiting time is observable in the DNS rate control scheme, where the average interarrival times are less than a

millisecond, unlike other services, where the average interarrival times are higher, and thus waiting times may be lower. Owing to this low interarrival time in the DNS setup, the larger the client population, the higher the average waiting time. This behavior takes place due to the fact that a larger number of clients are coming in to the RCS to request for tickets, with more or less the same arrival time, and they are provided with time windows well ahead into the future. Thus the average waiting time per client goes up substantially with the total population.



Clients requesting tickets at very small intervals.

Figure 4.14: Access window allocation to clients

The following observations can be made from the results:

1. For higher  $\delta_r$  (~1sec), with client population of 10K, the average waiting time was between 45-100 seconds, as compared to cases with  $\delta_r=90$  ms and  $\delta_r=1$ ms, where the average waiting time was between 8-20 seconds, as is seen in Figures 4.19 and 4.20. The reason for such a long waiting period is that  $\delta_r$  has a direct impact on the time window sizes given to



individual clients, during which they may place their requests, as is seen in the time window computation formulae below:

$$t_i = t_w + \Delta$$

$$t_{i+1} = t_i + w(\tau + 2\Delta + \delta_r)$$

Thus, with higher values of  $\delta_r$ , the clients are provided with larger time windows to place their requests in, and hence subsequent client requests are pushed off further into the future, before service is provided to them.

2. During the event of a “flash crowd” (i.e., high  $l$  and low  $t$ ), the average waiting time stays below 20 seconds for cases with  $0 < \delta_r < 90ms$ . Thus it is advisable to place an upper bound on the value of  $\delta_r$  that may be requested by a client, so as to reduce the waiting time to service for subsequent clients. Clients who may not be able to place their requests during the time window provided to them due to the smaller  $\delta_r$  value may re-issue a request for a ticket to the RCS, for subsequent accesses.

3. During the event of a DDoS attack (i.e., low  $l$  and low  $t$ ), for higher values of  $\delta_r$  (~1sec), which typically is the case during an underutilization attack, the average waiting time was around 45 seconds, as is seen in Figure 4.19 and 4.20, whereas for lower  $\delta_r$ , the waiting time was between 8-10 seconds, which again suggests that the system designer place an upper

bound on the requested value of  $\delta_r$ . So we may again conclude that  $\delta_r$  must lie in the following range:  $0 < \delta_r < 90$ .

### **Effect of Variations of Number of Clients on Utilization, Waiting Time**

The server utilization remains the same for variations in the number of clients ( $N$ ). This is because neither the time window size, nor the *wopt* value provided to the individual clients are affected by  $N$ . Therefore, having 20 requests in 20 seconds, or having 40 requests in 40 seconds give the same server utilization.

The client waiting time is severely impacted by variations in the client population. The small interarrival times of the DNS system ( $\sim 0.86$  ms, 0.03 ms), imply that a large number of clients are requesting tickets from the RCS in a back to back fashion, and are provided with time intervals pushed off well into the future. Thus the per-client average waiting time goes up substantially, as is evident from Figures 4.23 and 4.24, where for  $l=0.1$ ,  $\delta_r=1$  ms and  $N=100K$ , the average waiting time is around 100 seconds, and for  $l=0.9$  and other factors remaining the same, the waiting time is around 200 seconds. Again, Figures 4.23 and 4.24 show a very high waiting time for cases with  $\delta_r=1$  sec, thus further strengthening our earlier suggestion for placing an upper limit on the value of  $\delta_r$  for the DNS system.

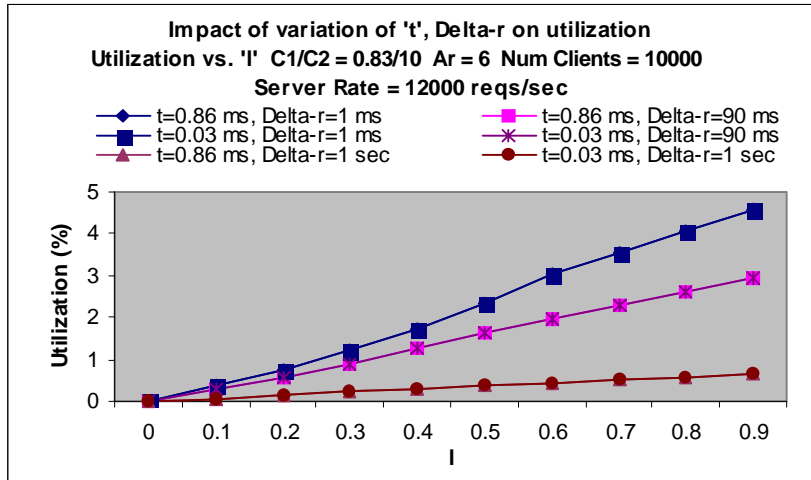
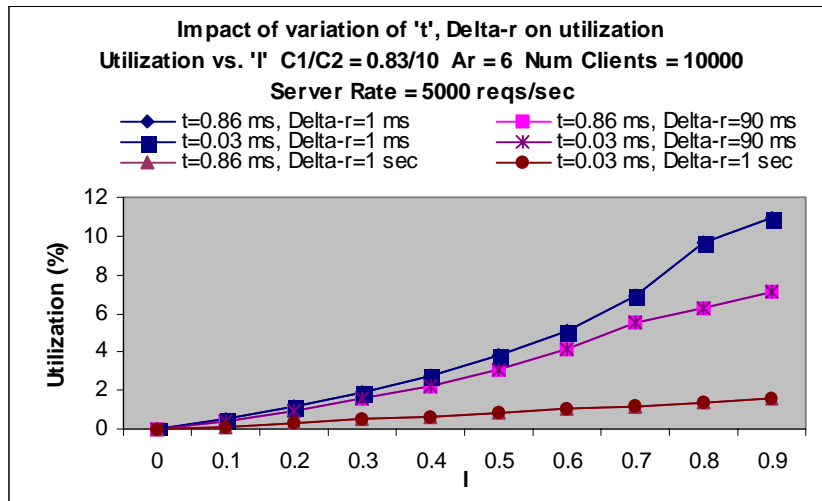


Figure 4.15: Server Utilization vs. l



Figures 4.16: Server Utilization vs. l

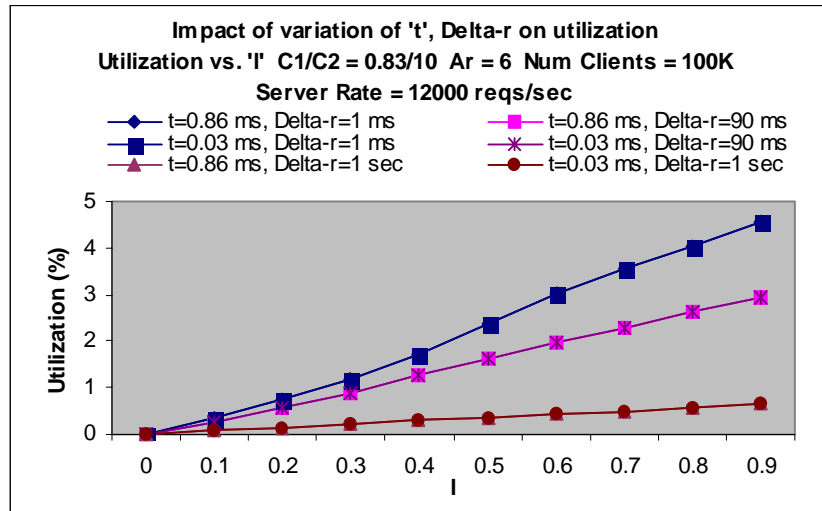


Figure 4.17: Server Utilization vs. l

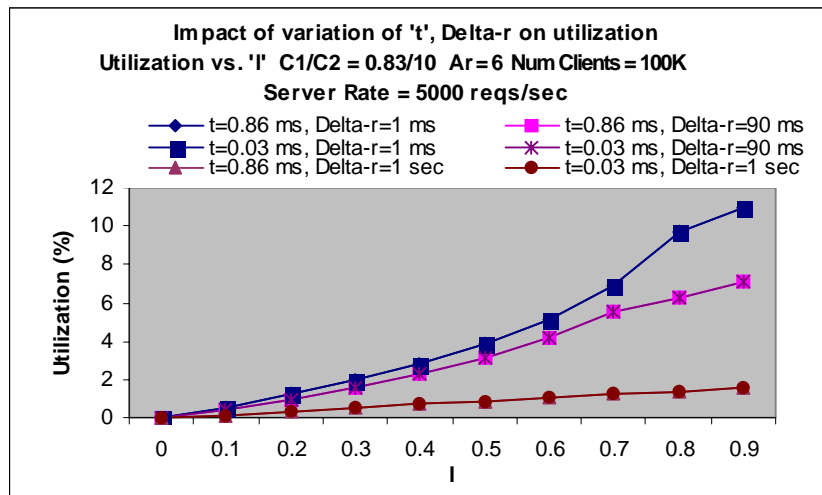


Figure 4.18: Server Utilization vs. l

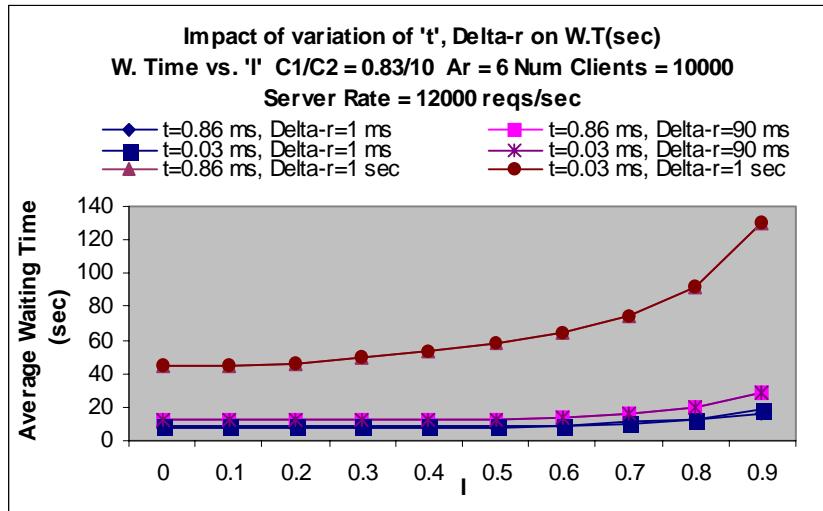


Figure 4.19: Average Waiting Time vs. l

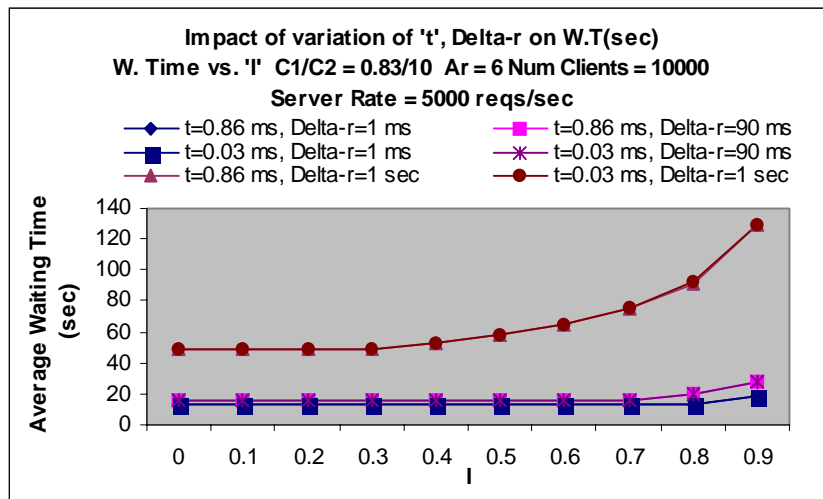


Figure 4.20: Average Waiting Time vs. l

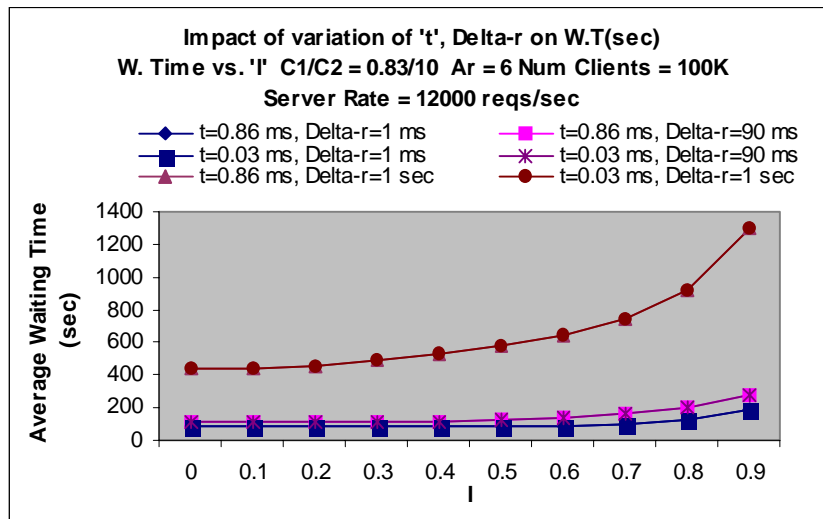


Figure 4.21: Average Waiting Time vs. l

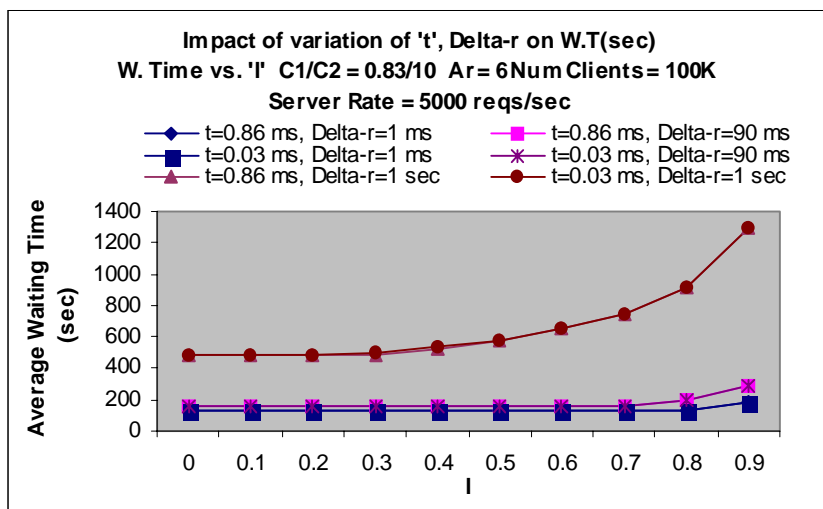


Figure 4.22: Average Waiting Time vs. l

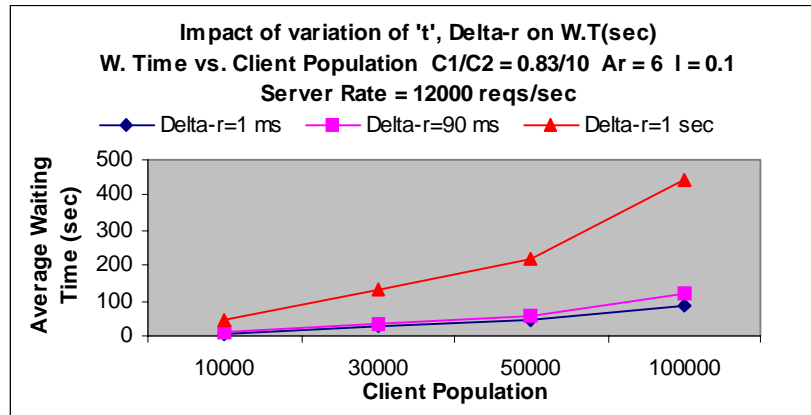


Figure 4.23: Average Waiting Time vs. Client Population

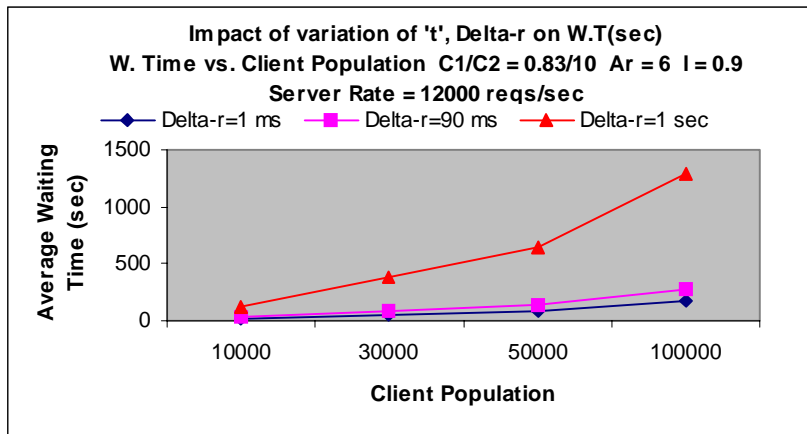


Figure 4.24: Average Waiting Time vs. Client Population

## Chapter 5

### Conclusions and Future Work

With the ever expanding gap between the network line rate, and the server throughput, user-level agreements have to be imposed in order to check on the load imposed on the server, be it a DNS root name server, or a CDN server. This thesis evaluated the performance of a novel rate control mechanism to control the traffic arrival rate to two types of servers, namely, DNS root name servers, and CDN servers. In particular, the application server utilization, and the client waiting times were studied for variations in the input parameters:  $c1$ ,  $c2$ ,  $Ar$ ,  $l$ ,  $\delta_r$ .

The results obtained from the simulations were at par with our expectations:

1. Increasing client population led to increase in the per-client waiting time, however, the server utilization remained unaffected.



2. Increasing values of the maximum interrequest time requested by a client,  $\delta_r$ , led to increase in the per-client waiting time, and lower server utilization.

3. Increasing values of the average number of requests used for *wopt* computation,  $A_r$ , led to an increase in the per-client waiting time, and lower server utilization.

The following bound is suggested to be placed on the value of  $\delta_r$ , so as to give reasonable waiting time guarantees to the clients.

- CDN Network:  $0 < \delta_r < 3$  sec.
- DNS Network:  $0 < \delta_r < 90$  msec.

The values of the parameters that characterize applications such as the average number of accesses required,  $A_r$ , and the maximum interrequest delay,  $\delta_r$ , and not just the communication and computation delays vary greatly from one network to another. Thus, the system designer must choose appropriate values of these application parameters before deciding on the optimal number of accesses, and time window sizes to be given to individual clients. This is the case because both the client waiting time, as well as the server utilization are affected by these values.

## **Future Work**

The simulation done in this thesis involved independent non-overlapping time windows, during which clients were expected to place their requests. However, overlapping time windows may be experimented with, and results may be obtained for analysis and comparison with the current results. This would enable us to concretize the bounds we have decided upon for certain parameters.

Considering the rising usage of wireless networks, the application of the rate control scheme simulated in this thesis to wireless networks has to be studied with extensive simulations. The results obtained will allow us to study the impact this scheme would have on the behavior of wireless network servers, as well as the client waiting time guarantees.

As all resources that become very common, wireless networks will face the same problems in security, as faced by wireline networks today, including DDoS attacks. Thus, it is essential to provide solutions to such problems based on empirical analysis already done, including work done in this thesis.

## Appendix A

### The Simulator

/\* Author: Zubair Baig

This program models the rate control server soln. for the DDoS problem proposed by Dr. V.D.Gligor. The arrival process of the requests from the clients to the Rate Control Server is Poisson, with exponential interarrival times(as per definition). The tickets granted to the clients contain  $t_i$ ,  $t_{i+1}$  values.

$t_i = \Delta + \text{first available time window.}$

$t_{i+1} = t_i + w(\tau + 2\Delta + \Delta_{tar})$

Number of accesses granted ( $w$ ) depends on the value of  $w_{opt}$ .

We are considering the case with single client per window.

The value of ' $\tau$ ' will decide the arrival of requests at the Verifier(Appl. Server), and thus the utilization.

The service time at the server is exponential with average time equal to some number.

We will record the total time the client has to wait before getting the service. We will also check the utilization of the app. server

We will also record the avg. number of visits per client to the RCS.

\*/

```
#include <stdio.h>
```

```
#include <math.h>
```

```
#include <stdlib.h>
```

```
#include <time.h>
```

```
#define NUM_CLIENTS 10000
```

```
#define APP_SERVER_RATE 600
```

```
/* ALWAYS USE MILLISECONDS AS UNITS.....!!!!!!!!!!!!!!!!!!!!*/
```

```
const double AVG_INTERARRIVAL_TIME= 0.01;
```

```
/* Interarrival time at RCS (msecs)*/
```

```
const double AVG_SERVICE_TIME= 100.0; /* Service time at the  
appl. server (msecs)*/
```

```
const double MAC_PROC = 0.0;
```

```

const int AR_MAX = 6; /* Ar upper limit */
const int AR_MIN = 1; /* Ar lower limit */

typedef struct RCS_ticket{
int client_id;
double arrival_time; /* Arrival time to the TGS */
double ti; /* Tx */
double ti1; /* Ty */
int num_reqs; /* # of Requests for tickets made by client */
double ctr; /* Max. number of requests given to client */
double deltar; /* Max. time between 2 consecutive reqs to the app.
server */
int come_again;
} RCS_ticket;

RCS_ticket rcs_ticket[NUM_CLIENTS];

double interarrival_time[NUM_CLIENTS];
double simulation_time = 0.0;

const double DELTAR = 30000.0; /* Max. gap between 2 consecutive
requests.(in msec) */
const double DELTA = 200.0; /* Network delay (in msec) */

const double Ar = 6.0;
const double C1 = 200.0;
const double C2 = 1000.0;

```

```

const int L=1024;

int TAU;

FILE *fd1;

/* Computing the value of Wopt for different values of paramters */
/* This function computes the interarrival time between consecutive
requests
to the RCS */

double expon_interarrival(void)
{
int value;
double random_Y;
double arrival_time;
double result;
int cnt;
value = rand() % 100;

random_Y = (double)value/100.0;

if (random_Y == 0.0)
random_Y = 0.01;

arrival_time = -(log(random_Y)/(1.0/AVG_INTERARRIVAL_TIME));

```

```

result = arrival_time ;

if (result == 0.0)
result = 0.0;

/*if (result > 100.0)
result = rand() % 100;

printf("Interarrival time is %f\n\n",result);
*/
return (result);
} /* End of function */

/* This function computes the interarrival time between consecutive
requests
to the RCS */

double expon_service(void)
{
int value;
double random_Y;
double service_time;
double result;
int cnt;

value = rand() % 100;

```

```

random_Y = (double)value/100.0;

if (random_Y == 0.0)
random_Y = 0.01;
service_time = -(log(random_Y)/(1.0/AVG_SERVICE_TIME));

result = service_time ;

if (result == 0.0)
result = 0.0;

/*if (result > 100.0)
result = rand() % 100;
*/
printf("Service time is %f\n\n",result);

return (result);
} /* End of function */

/*****/
/* This function emulates a normal web server with no Rate Control
Mechanism implemented */

void normal_server(void)
{

```



```

double total_ctrs;
double total_time;

}/* End of function */
/*****/

/* This function computes the utilization at the appl. server, # of visits on
avg. per client
to the RCS, as well as the Avg. Waiting Time for each client before get-
ting the service. */

void compute(double l, double Wopt)
{
double total_time=0.0, total_ctrs=0.0;
double legitimate_clients;
int cnt,cnt2;
double waiting_time=0.0;
double waiting[NUM_CLIENTS];
double num_visits = 0.0;

TAU = L/APP_SERVER_RATE;

legitimate_clients = 1 * (double) NUM_CLIENTS;

/* Computing the Utilization at the Server */
for (cnt=0;cnt<(int)legitimate_clients;cnt++)
{

```

```

    total_time += (rcs_ticket[cnt].ti1 - rcs_ticket[cnt].ti);
    total_ctrs += rcs_ticket[cnt].ctr;
}

if (total_time == 0.0)
{printf("0\t");
fprintf(fd1,"0\t");
}
else
{fprintf(fd1,"%0.3f\t",total_ctrs*100000.0/((double)APP_SERVER_RATE*total_time));
}

/* Computing the Avg. Waiting Time per client */
for (cnt=0;cnt <NUM_CLIENTS;cnt++)
{
    waiting_time += (rcs_ticket[cnt].ti - rcs_ticket[cnt].arrival_time);
    waiting[cnt] = waiting_time;
}
fprintf(fd1," %0.3f\t",waiting_time/(1000.0*(double)NUM_CLIENTS));
/* Computing the Average number of visits to the RCS per client */

for (cnt=0;cnt<NUM_CLIENTS;cnt++)
{
    if((double)rcs_ticket[cnt].num_reqs <= rcs_ticket[cnt].ctr)
        num_visits +=1.0;
    else
        num_visits += (double)rcs_ticket[cnt].num_reqs/rcs_ticket[cnt].ctr;
}

```

```

printf(" %d\n", (int) num_visits/NUM_CLIENTS);
fprintf(fd1, " %d\n", (int) num_visits/NUM_CLIENTS);

printf("\n\n");

/*****

/***** Repeating the simulations 100 times */

for (cnt2=0; cnt2<10; cnt2++)
{
printf("\n\n");
printf("\n l\tWopt\tUtil(%%)\tW.T(msec)\t#\tvisits\n");
fprintf(fd1, "\n l\tWopt\tUtil(%%)\t W.T(msec)\t#\tvisits\n");
fprintf(fd1, " %0.2f\t%0.2f\t", l, Wopt);

for (cnt=0; cnt<NUM_CLIENTS; cnt++)
{
interarrival_time[cnt] = expon_interarrival(); /* Computing the inter-
arrival time for the clients*/

rcs_ticket[cnt].arrival_time = simulation_time + interarrival_time[cnt];

simulation_time += interarrival_time[cnt]; /* Total time for simula-
tion */

```

```

    rcs_ticket[cnt].num_reqs = rand() % AR_MAX;    /* Each client
makes a req. for random # of w */

    while(rcs_ticket[cnt].num_reqs <AR_MIN)
rcs_ticket[cnt].num_reqs++;
}    /* End of for loop */
/* Providing the ti..ti+1 values to the clients */

for (cnt=0;cnt<NUM_CLIENTS;cnt++)
{
    rcs_ticket[cnt].ti = rcs_ticket[cnt].arrival_time + MAC_PROC +
DELTA;
}    /* End of for */

rcs_ticket[cnt].ti1 = rcs_ticket[cnt].ti + rcs_ticket[cnt].ctr*((double)TAU
+ 2.0*DELTA + DELTAR);
/* Checking for overlap */
if (cnt != 0)
{
    if (rcs_ticket[cnt].ti < rcs_ticket[cnt-1].ti1)
rcs_ticket[cnt].ti = rcs_ticket[cnt-1].ti1;
}

/* Computing the Utilization at the Server */
for (cnt=0;cnt<(int)legitimate_clients;cnt++)
{
    total_time += (rcs_ticket[cnt].ti1 - rcs_ticket[cnt].ti);
}

```

```

    total_ctrs += rcs_ticket[cnt].ctr;
}

if (total_time == 0.0)
{printf("0\t");
fprintf(fd1,"0\t");
}
else
{printf("%0.3f\t",total_ctrs*100.0/((double)APP_SERVER_RATE*total_time));
fprintf(fd1,"%0.3f\t",total_ctrs*100000.0/((double)APP_SERVER_RATE*total_time));
}
/* Computing the Avg. Waiting Time per client */
for (cnt=0;cnt <NUM_CLIENTS;cnt++)
{
    waiting_time += (rcs_ticket[cnt].ti - rcs_ticket[cnt].arrival_time);
    waiting[cnt] = waiting_time;
}
printf(" %0.3f\t",waiting_time/(double)NUM_CLIENTS);
fprintf(fd1," %0.3f\t",waiting_time/(1000.0*(double)NUM_CLIENTS));

/* Computing the Average number of visits to the RCS per client */

for (cnt=0;cnt<NUM_CLIENTS;cnt++)
{
    if((double)rcs_ticket[cnt].num_reqs <= rcs_ticket[cnt].ctr)
        num_visits +=1.0;
    else
        num_visits += (double)rcs_ticket[cnt].num_reqs/rcs_ticket[cnt].ctr;
}

```

```

}

printf(" %d\n",(int)num_visits/NUM_CLIENTS);
fprintf(fd1," %d\n",(int)num_visits/NUM_CLIENTS);
printf("\n\n\n");
}
} /* End of function Compute */
/*****/

/* Function Main */
int main (void)
{ int cnt,cnt2;
  double l,Wopt;

fd1 = fopen("rcs_results_12_02_deltar.txt","aw");
srand(time(0));

for (cnt=0;cnt<NUM_CLIENTS;cnt++)
{
  rcs_ticket[cnt].client_id = cnt;

  rcs_ticket[cnt].deltar = DELTAR;          /* Constant value */
  interarrival_time[cnt] = expon_interarrival(); /* Computing the interar-
rival time for the clients*/

  rcs_ticket[cnt].arrival_time = simulation_time + interarrival_time[cnt];
  simulation_time += interarrival_time[cnt]; /* Total time for simula-
tion */
}
}

```

```

        rcs_ticket[cnt].num_reqs = rand() % AR_MAX;    /* Each client
makes a req. for random # of w */
        while(rcs_ticket[cnt].num_reqs <AR_MIN)
rcs_ticket[cnt].num_reqs++;
    }    /* End of for loop */
/* Providing the ti..ti+1 values to the clients */

    for (cnt=0;cnt<NUM_CLIENTS;cnt++)
    {
        rcs_ticket[cnt].ti = rcs_ticket[cnt].arrival_time + MAC_PROC +
DELTA;
    }    /* End of for */
TAU = L*1000/APP_SERVER_RATE;

printf("\n");
printf("C1      = %0.1f msec\n", C1);
printf("C2      = %0.1f msec\n", C2);
printf("Ar      = %0.1f\n", Ar);
printf("t       = %0.3f msec\n", AVG_INTERARRIVAL_TIME);
printf("Server Rate = %d reqs/sec\n",APP_SERVER_RATE);
printf("Queue length= %d reqs\n",L);
printf("TAU      = %d msec\n",TAU);

fprintf(fd1,"/*****\n");
fprintf(fd1,"\n");
fprintf(fd1,"C1      = %0.1f msec\n", C1);
fprintf(fd1,"C2      = %0.1f msec\n", C2);

```

```

fprintf(fd1,"Ar      = %0.1f\n", Ar);
fprintf(fd1,"t      = %0.3f msecs\n", AVG_INTERARRIVAL_TIME);
fprintf(fd1,"Server Rate = %d reqs/sec\n",APP_SERVER_RATE);
fprintf(fd1,"Queue length= %d reqs\n",L);
fprintf(fd1,"TAU      = %d msecs\n",TAU);
/* This for loop will execute for diff. values of l (% of legitimate clients)
*/
/*****/
printf("\n l\tWopt\tUtil(%%)\t W.T(msec)\t# visits\n");
  fprintf(fd1,"\n l\tWopt\tUtil(%%)\t W.T(msec)\t# visits\n");

  for (l = 0.0; l < 0.9; l += 0.1)
  {
Wopt = sqrt ( (Ar * C1)/(C2 * (1.0-l)));

/*Assigning the value of ctr to the individual clients */

for (cnt=0; cnt<NUM_CLIENTS; cnt++)
{
  if(rcs_ticket[cnt].num_reqs < (int)Wopt)
    rcs_ticket[cnt].ctr = (double)rcs_ticket[cnt].num_reqs;
    else
    rcs_ticket[cnt].ctr = Wopt;
rcs_ticket[cnt].ti1 = rcs_ticket[cnt].ti + rcs_ticket[cnt].ctr*((double)TAU
+ 2.0*DELTA + DELTA);

/* Checking for overlap */

```



```

if (cnt != 0)
{
    if (rcs_ticket[cnt].ti < rcs_ticket[cnt-1].ti1)
rcs_ticket[cnt].ti = rcs_ticket[cnt-1].ti1;
    }
}

rcs_ticket[cnt].ti1 = rcs_ticket[cnt].ti + rcs_ticket[cnt].ctr*((double)TAU + 2.0*DELTA + DELTAR);
printf(" %0.2f\t%0.2f\t",l,Wopt);
fprintf(fd1, " %0.2f\t%0.2f\t",l,Wopt);

    /* Calll the function 'compute' to find out the utilization, #of visits,
waiting time */
    /******
compute(l,Wopt);
    }
fclose(fd1);

    /******
/* Checking the behaviour of a normal server with M/M/1 */
normal_server();
} /* End of Main */

```

## BIBLIOGRAPHY

- [1] L. von Ahn, M. Blum, N. Hopper, and J. Langford, "CAPTCHA: Using Hard AI Problems for Security," *Advances in Cryptography - EUROCRYPT 2003*, Warsaw, Poland, May 2003.
- [2] T. Aura, P. Nikander, and J. Leiwo, "DOS-Resistant Authentication with Client Puzzles," *Proc. of the 8th International Security Protocols Workshop*, Cambridge, U.K., April 2000, LNCS vol. 2133, Springer Verlag, pp. 170-178.
- [3] N. Brownlee, K. Claffy, and E. Nemeth, "DNS Root/g TLD Performance Measurements", *USENIX LISA*, San Diego, CA. December 2001.
- [4] N. Brownlee, K. Claffy, and E. Nemeth, "DNS Measurements at the Root Server", *Proc. of Globecom*, San Antonio, TX. November 2001.
- [5] J. Cabrera, L. Lewis, X. Qin, W. Lee, R. Prasanth, B. Ravichandran, and R. Mehra, "Proactive Detection of Distributed Denial of Service Attacks using MIB Traffic Variables - A Feasibility Study," *Proc. of the 7th*

IFIP/IEEE International Symposium on Integrated Network Management, Seattle, WA, May 2001.

[6] R. K. C Chang, "Defending against Flooding-Based Distributed Denial-of-Service Attacks: A Tutorial," *IEEE Communications Magazine*, pp. 42-51, October 2002.

[7] J. Elliott, "Distributed Denial of Service Attacks and the Zombie Ant Effect," *IT Pro*, pp. 55-57, March/April 2000.

[8] M. Fomenkov, K. Claffy, B. Huffaker, and D. Moore, "Macroscopic Internet Topology and Performance Measurements from the DNS Root Name Servers", USENIX LISA, San Diego, CA. December 2001.

[9] V. D. Gligor, "Guaranteeing Access in Spite of Distributed Service-Flooding Attacks", Proc. of the International Workshop on Security Protocols, Sidney Sussex College, Cambridge, U.K., April 2003.

[10] M. Kaeo, "Designing Network Security," Cisco Press, 1999.

[11] D. Kashiwa, E. Chen, and H. Fuji, "Active Shaping: A Countermeasure against DDoS Attacks," Proc. of the 2nd European Conference on Universal Multiservice Networks, Colmar, France, April 2002.

[12] C. Kaufman, R. Perlman, and M. Speciner, "Network Security: Private Communication in a Public World," Prentice Hall, 2002.

[13] H. Kobayashi, "Modeling and Analysis: An Introduction to System Performance Evaluation Methodology," Addison-Wesley, 1978.

- [14] A. Kobsa, J. Fink, “Performance Evaluation of User Modeling Servers Under Real World Workload Conditions,” Proc. of the 9th International Conference on User Modeling, Johnstown, PA, 2003.
- [15] D. Moore, G. Voelker, and S. Savage, “Inferring Internet Denial of Service Activity,” Proc. of 2001 USENIX Security Symposium, Washington D.C, August 2001.
- [16] J. H. Saltzer, D. P. Reed, and D. D. Clark, “End-to-End Arguments in System Design,” *ACM Transactions on Computer Systems*, Vol.2, Nov. 1984.
- [17] W. F. Slater, “The Internet Outage and Attacks of October 2002”, Available: <http://www.ISOC-Chicago.org>, Nov. 7, 2002.
- [18] W. Stallings, “High-Speed Networks: TCP/IP and ATM Design Principles,” Prentice Hall, 1998.
- [19] A. S. Tanenbaum, “Computer Networks,” Prentice Hall, 2003.
- [20] L. Wang, V. Pai, and L. Peterson, “The Effectiveness of Request Redirection on CDN Robustness”, Proc. of the 5th Symposium on OS Design and Implementation (OSDI), Boston, Mass. December 2002.
- [21] X. Wang and M. Reiter, “Defending Against Denial-of-Service Attacks with Puzzle Auctions,” Proc. of IEEE Symposium on Security and Privacy, Berkeley, CA, May 2003.

- [22] N. Weiler, "Honeypots for Distributed Denial of Service Attacks," Proc. of the 11th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2002.
- [23] J. Xu, R. Lipton, and I. Essa, "Hello, Are You Human," Technical Report, Georgia Institute of Technology, November 2000.
- [24] [http://www.netapp.com/ftp/netcache\\_siebel7.pdf](http://www.netapp.com/ftp/netcache_siebel7.pdf);  
Verified on: 11/17/03.
- [25] <http://www.cs.cmu.edu/~srini/15-744/F02/readings/PM95.ps.gz>;  
Verified on: 11/17/03.
- [26] <http://research.microsoft.com/~lilic/papers/pub/dimacs.ps>;  
Verified on: 11/17/03.
- [27] <http://npmv5.solarwinds.net/NetPerfMon/NodeDetails.Asp?NodeID=32>; Verified on: 11/17/03.
- [28] <http://www.webhostselect.com>; Verified on: 11/17/03.
- [29] <http://www.cs.brandeis.edu/~mfc/cs120/asst2.txt>;  
Verified on: 11/17/03.
- [30] <http://www.cyberspace.org/stats/httpd030621.html>;  
Verified on: 11/17/03.
- [31] <http://stats.gseis.ucla.edu/gseis>; Verified on: 11/17/03.
- [32] <http://www.dent.ucla.edu/stats/0203.html>; Verified on: 11/17/03.
- [33] <http://www.utas.edu.au/stats/results.html>; Verified on: 11/17/03.

[34] <http://iwi.com/Pubs/dos.html>; Verified on: 11/17/03.

[35] <http://downloads.securityfocus.com/library/oliver.pdf>;

Verified on: 11/17/03.

[36] <http://www.caida.org/~kkeys/dns/2002-08-14/>; Verified on: 11/17/03.

[37] <http://www.caida.org/~kkeys/dns/2002-10-21/>; Verified on: 11/17/03.

[38] [http://www-1.ibm.com/servers/eserver/pseries/news/pressreleases/2000/apr/network\\_solutions.html](http://www-1.ibm.com/servers/eserver/pseries/news/pressreleases/2000/apr/network_solutions.html); Verified on: 11/17/03.

[39] <http://www.caida.org/outreach/presentations/ietf0112/dns.damage.htm>

Verified on: 11/17/03.

[40] <http://www.cymru.com/DNS/dns01.html>; Verified on: 11/17/03.

[41] [http://www.usenix.org/publications/library/proceedings/usits99/full\\_papers/lefebvre/lefebvre\\_html/](http://www.usenix.org/publications/library/proceedings/usits99/full_papers/lefebvre/lefebvre_html/); Verified on: 11/17/03.