Using a time synchronization protocol to co-ordinate a low rate denial of service (DoS) attack in PlanetLab.

DIPAK BOYED
MARK IHIMOYAN

1. ABSTRACT:

This report summarizes our assignment on using Cristian‟s probabilistic time synchronization algorithm[1] to co-ordinate periodic, low-rate denial of service attacks[2] on our home computer from PlanetLab[3] nodes. We were able to verify denial of service on a simple website hosted on our home computer with as few as 10 PlanetLab slave nodes and bring the attacked computer‟s CPU utilization to 100% with 50 PlanetLab nodes.

2. INTRODUCTION:

Clock synchronization problem in computer science deals with the idea of bringing together or harmonizing the internal clocks of several computers that may differ from each other. In this assignment, we implemented a time synchronization protocol based on Cristian‟s algorithm to synchronize the clocks of several computers. The synchronized computers were then used to carry out a Denial of Service (DoS) attack on a victimized end host.

DoS attacks consume resources in networks, server clusters, or end hosts with the malicious objective of preventing or severely degrading performance to legitimate users. Resources that are typically consumed in such attacks include network bandwidth, server or router CPU cycles, server interrupt processing capacity, and specific protocol data structures.

In this report, the time synchronization protocol, denial of service attacks and results of the protocol on PlanetLab nodes are presented. To this effect, the paper has been divided into the following sections: Section 3 describes the methodology we employed in tackling the home-work problem and also details an explanation of the source code. In Section 4 the time synchronization protocol is discussed. In section 5 the denial of service (DoS) attack is explained. Our results and data from running the home-work on PlanetLab nodes are discussed in Section 6. We also prove why our work was successful in Section 6.

3. METHODOLOGY:

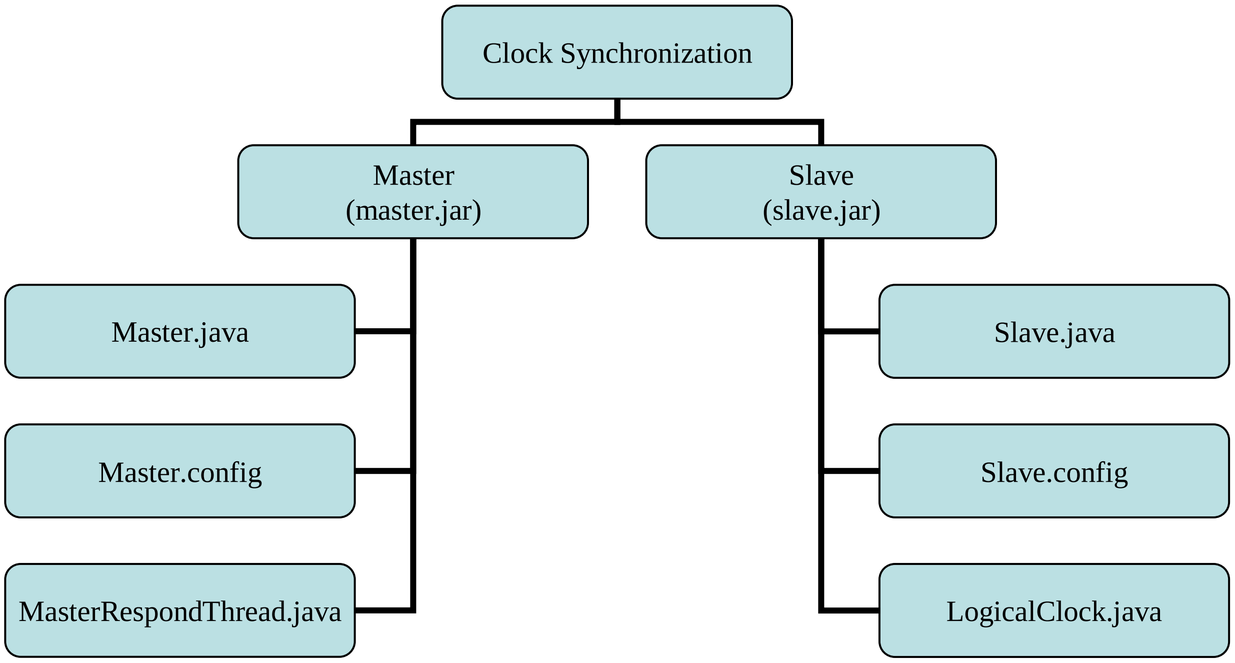
This assignment was roughly divided into two major components, (a) time synchronization, and (b) denial of service attack. The time synchronization component was responsible for ensuring that all attacking machines („slaves") repeatedly synchronize their clocks with a given „master" machine. The goal was to ensure that all the slaves synchronized their clocks with respect to the master"s clock and could then simultaneously send low-rate DoS packets every 500 milliseconds.

3.1. Environment

The project was written in Java and compiled in the CS department Linux boxes with the Java 2 Runtime Environment, version 1.5.0_06-b05. We did not verify that the code complies on other platforms.

The java binaries were converted into jar files and exported to the PlanetLab nodes, which is where the experiments were conducted.

The source code was roughly structured in the following way:



The project consisted of two applications (jar files) with distinct roles:

(a) **Master**: The Master machine was deployed on one PlanetLab node and was responsible for receiving time synchronization requests from slaves and sending them back its own time.

(b) **Slave(s)**: Slaves were deployed to more than one PlanetLab nodes and were responsible for synchronizing their clocks to the master‟s time as well as sending the DoS packets to the targeted end host.

3.2. List of source files

The Master side of the project was bundled into the master.jar file which consists of the following:

File	Description
Master.java	Entry point class for the master that listens for new slave connections and spawns a response thread for every slave connection request for time synchronization.
Master.config	Configuration file for Master.java (more details below).
MasterRespondThread.java	Response thread that processes synchronization requests from one particular instance of a time synchronization request from a slave.
Manifest.txt	Manifest information for the jar file.

The Slave side of the project was bundled into the slave.jar file which consists of the following:

File	Description
Slave.java	Entry point class for the slave that sends/receives time synchronization requests/replies to/from the Master and transmits DoS packets every 500 milliseconds.
Slave.config	Configuration file for Slave.java (more details below).
LogicalClock.java	Class representing the logical clock that adjusts the slave‟s hardware clock with the master‟s clock information.
Manifest.txt	Manifest information for the jar file.

3.3 Configuration

Both the master and slave were designed to be configurable so that the input parameters and algorithm variables could be modified without a code re-compile. This was achieved by using the *.config files. The config files are text files containing fixed and known input parameters in a given format that the master and slave could read to initialize themselves.

The Master.config file consists of the following information:

Property Name	Description
MasterRcvPort	Port number on which the Master should listen for slave connections and receive time synchronization requests.
SlaveSendPort	Port number on which the Master should send the time synchronization replies to the slaves.

The Slave.config file consists of the following information:

Property Name	Description
MasterIP	IP address or host name where the Master is running.
MasterSendPort	Port number on which the Slave should send synchronization requests to the Master. This property‟s value should be identical to MasterRcvPort‟s value in Master.config.
SlaveRcvPort	Port number on which the Slave should receive synchronization replies from the Master. This property‟s value should be identical to SlaveSendPort‟s value in Master.config.
U	Input to Cristian‟s algorithm representing round--trip acceptability threshold in milliseconds.
K	Input to Cristian‟s algorithm representing the maximum number of successive synchronization attempts
W	Input to Cristian‟s algorithm representing the waiting time between synchronization attempts in milliseconds.
SynchFrequency	Input to Cristian‟s algorithm representing the time to wait before sending a new synchronization request in milliseconds.
DoSTargetIP	IP address or host name of the target machine for the DoS attack.
DoSTargetPort	Port number of the target machine where DoS attacks will be sent.
AttackLifetime	Time (in minutes) for which the Slave application should be running.

	Note: We did not make the Master‟s lifetime time-bound.
AttackMode	Which DoS attack mode to use (TCP or UDP).

The properties and their values defined in the config files are specified in the fixed format where each property and its value is „=‟ separated without any whitespaces and contained in a separate line. For simplicity, the format is fixed and all configuration input is assumed to be valid (i.e. we do not handle error checking and validation).

The project submission also has a „SampleOutput‟ folder which contains actual slave output logs from runs in PlanetLab. These logs are verification tools and are a supplement to our Results section (Section 6). More details on their organization are provided in Section 6.

3.4. Execution:

The Master can be executed on one designated machine by running the following command line:

- *"java –jar ~/ClockSynch/master/master.jar ~/ClockSynch/master/Master.config"*

Assuming the above command is executed from the home directory and the home directory contains the „ClockSynch‟ folder which in turn contains the „master‟ folder.

The Slave can be executed on machines by running the following command line:

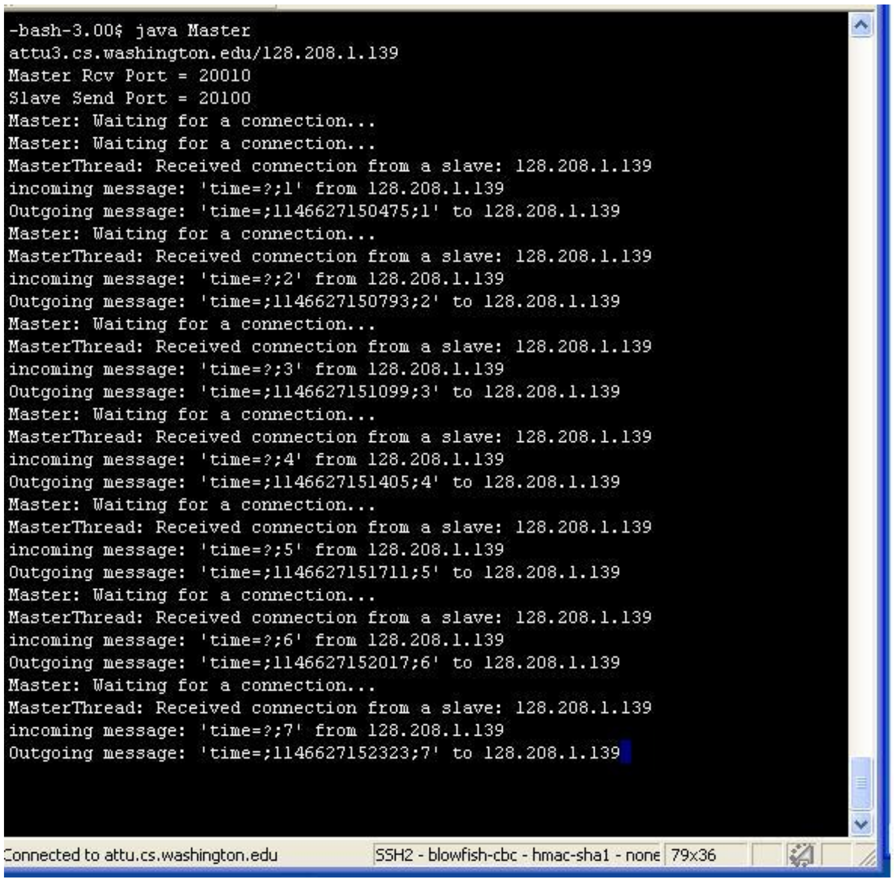
- *"java –jar ~/ClockSynch/slave/slave.jar ~/ClockSynch/slave/Slave.config"*

Assuming the above command is executed from the home directory and the home directory contains the „ClockSynch‟ folder which in turn contains the „slave‟ folder.

3.5. Output:

A screenshot of the sample outputs of the master and a slave is shown below. Both the master and slave(s) print out information about the configuration parameters and print details of receiving and sending every time synchronization message. Along with the message, the time stamp is also printed (which turned out be very helpful in debugging).

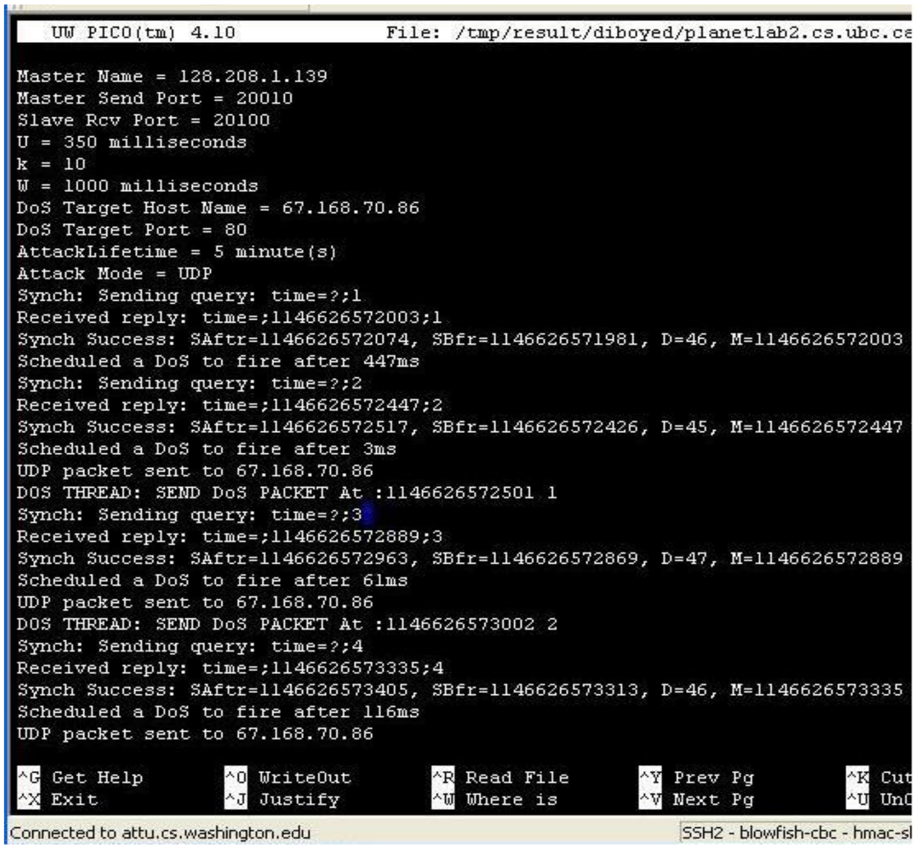
The slave also prints information about the denial of service attack and at the end of execution it summarizes a list of useful synchronization and denial of service statistics.



```
-bash-3.00$ java Master
attu3.cs.washington.edu/128.208.1.139
Master Rcv Port = 20010
Slave Send Port = 20100
Master: Waiting for a connection...
Master: Waiting for a connection...
MasterThread: Received connection from a slave: 128.208.1.139
incoming message: 'time=?;1' from 128.208.1.139
Outgoing message: 'time=;1146627150475;1' to 128.208.1.139
Master: Waiting for a connection...
MasterThread: Received connection from a slave: 128.208.1.139
incoming message: 'time=?;2' from 128.208.1.139
Outgoing message: 'time=;1146627150793;2' to 128.208.1.139
Master: Waiting for a connection...
MasterThread: Received connection from a slave: 128.208.1.139
incoming message: 'time=?;3' from 128.208.1.139
Outgoing message: 'time=;1146627151099;3' to 128.208.1.139
Master: Waiting for a connection...
MasterThread: Received connection from a slave: 128.208.1.139
incoming message: 'time=?;4' from 128.208.1.139
Outgoing message: 'time=;1146627151405;4' to 128.208.1.139
Master: Waiting for a connection...
MasterThread: Received connection from a slave: 128.208.1.139
incoming message: 'time=?;5' from 128.208.1.139
Outgoing message: 'time=;1146627151711;5' to 128.208.1.139
Master: Waiting for a connection...
MasterThread: Received connection from a slave: 128.208.1.139
incoming message: 'time=?;6' from 128.208.1.139
Outgoing message: 'time=;1146627152017;6' to 128.208.1.139
Master: Waiting for a connection...
MasterThread: Received connection from a slave: 128.208.1.139
incoming message: 'time=?;7' from 128.208.1.139
Outgoing message: 'time=;1146627152323;7' to 128.208.1.139
```

Connected to attu.cs.washington.edu | SSH2 - blowfish-cbc - hmac-sha1 - none | 79x36

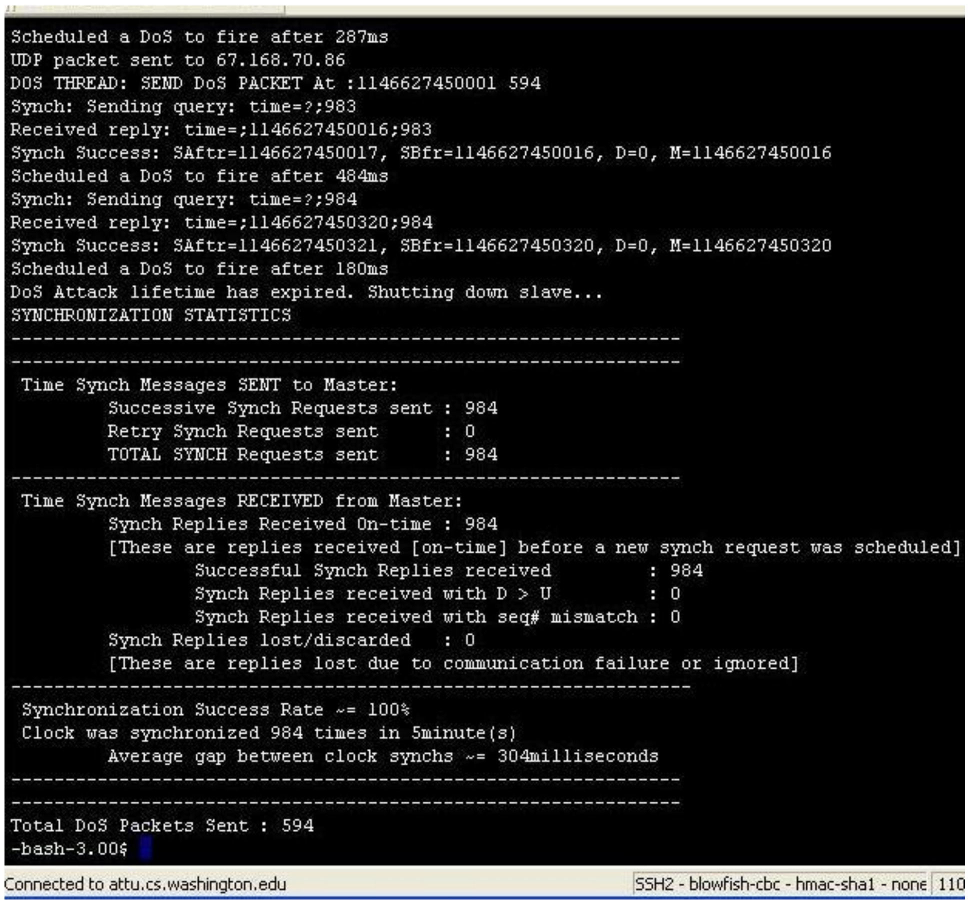
Fig 3.1: Sample Master‟s output when started



```
  UW PICO(tm) 4.10          File: /tmp/result/diboyed/planetlab2.cs.ubc.ca

Master Name = 128.208.1.139
Master Send Port = 20010
Slave Rcv Port = 20100
U = 350 milliseconds
k = 10
W = 1000 milliseconds
DoS Target Host Name = 67.168.70.86
DoS Target Port = 80
AttackLifetime = 5 minute(s)
Attack Mode = UDP
Synch: Sending query: time=?;1
Received reply: time=;1146626572003;1
Synch Success: SAftr=1146626572074, SBfr=1146626571981, D=46, M=1146626572003
Scheduled a DoS to fire after 447ms
Synch: Sending query: time=?;2
Received reply: time=;1146626572447;2
Synch Success: SAftr=1146626572517, SBfr=1146626572426, D=45, M=1146626572447
Scheduled a DoS to fire after 3ms
UDP packet sent to 67.168.70.86
DOS THREAD: SEND DoS PACKET At :1146626572501 1
Synch: Sending query: time=?;3
Received reply: time=;1146626572889;3
Synch Success: SAftr=1146626572963, SBfr=1146626572869, D=47, M=1146626572889
Scheduled a DoS to fire after 61ms
UDP packet sent to 67.168.70.86
DOS THREAD: SEND DoS PACKET At :1146626573002 2
Synch: Sending query: time=?;4
Received reply: time=;1146626573335;4
Synch Success: SAftr=1146626573405, SBfr=1146626573313, D=46, M=1146626573335
Scheduled a DoS to fire after 116ms
UDP packet sent to 67.168.70.86

^G Get Help     ^O WriteOut     ^R Read File    ^Y Prev Pg      ^K Cut
^X Exit         ^J Justify      ^W Where is     ^V Next Pg      ^U UnC
```

Connected to attu.cs.washington.edu | SSH2 - blowfish-cbc - hmac-sl

Fig 3.2: Sample Slave‟s output when started



```
Scheduled a DoS to fire after 287ms
UDP packet sent to 67.168.70.86
DOS THREAD: SEND DoS PACKET At :1146627450001 594
Synch: Sending query: time=?;983
Received reply: time=;1146627450016;983
Synch Success: SAftr=1146627450017, SBfr=1146627450016, D=0, M=1146627450016
Scheduled a DoS to fire after 484ms
Synch: Sending query: time=?;984
Received reply: time=;1146627450320;984
Synch Success: SAftr=1146627450321, SBfr=1146627450320, D=0, M=1146627450320
Scheduled a DoS to fire after 180ms
DoS Attack lifetime has expired. Shutting down slave...
SYNCHRONIZATION STATISTICS
-------------------------------------------------------------
-------------------------------------------------------------
 Time Synch Messages SENT to Master:
         Successive Synch Requests sent : 984
         Retry Synch Requests sent      : 0
         TOTAL SYNCH Requests sent      : 984
-------------------------------------------------------------
 Time Synch Messages RECEIVED from Master:
         Synch Replies Received On-time : 984
         [These are replies received [on-time] before a new synch request was scheduled]
                 Successful Synch Replies received         : 984
                 Synch Replies received with D > U         : 0
                 Synch Replies received with seq# mismatch : 0
         Synch Replies lost/discarded   : 0
         [These are replies lost due to communication failure or ignored]
--------------------------------------------------------------
 Synchronization Success Rate ~= 100%
 Clock was synchronized 984 times in 5minute(s)
         Average gap between clock synchs ~= 304milliseconds
-------------------------------------------------------------
-------------------------------------------------------------
Total DoS Packets Sent : 594
-bash-3.00$
```

Connected to attu.cs.washington.edu SSH2 - blowfish-cbc - hmac-sha1 - none 110

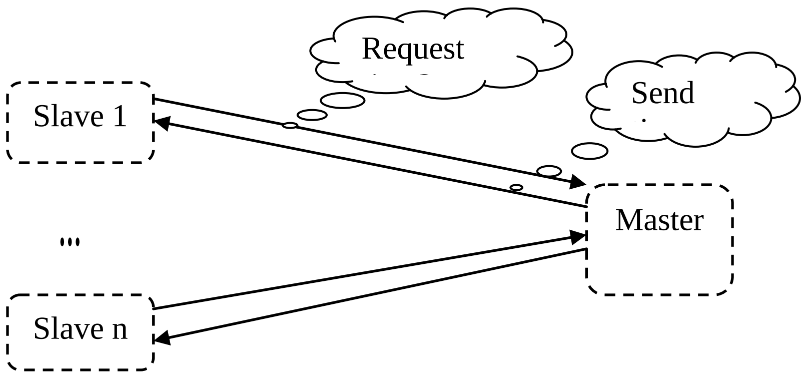
Fig 3.3: Sample Slave‟s output when finished

4. TIME SYNCHRONIZATION:

This section describes the time synchronization part of the project. We decided to implement time synchronization using Cristian‟s algorithm based on probabilistic modeling. The synchronization protocol is directly derived from the description in the Appendix of the Cristian paper.

For simplicity the amortization rate and maximum deviation variables described in the paper are ignored and the time synchronization requests are sent at fixed intervals (*synchFrequency*). We did not modify the system clock but maintained our own Logical clock that was computed using the slave‟s system clock and an adjustment parameter derived from the master‟s clock reading and round-trip message delay. This also meant that the logical clock reading represented a discontinuous function that was updated during each successful synchronization. Sending a DoS packet successfully and precisely every 500ms meant that we would need to synchronize the slave‟s clock successfully as often as possible. Essentially the slave(s) ensure that they send time requests to the master often enough to ignore the drift between the clocks and adjust their hardware clock reading with the master‟s clock reading and the delay in receiving the response since it left the master.

Our synchronization protocol is „internal clock synchronization‟ where all machines (slaves) attempt to synchronize their clocks with an internal (master) machine as shown in the figure below.



We assume that our distributed system has static membership of the nodes and do not handle dynamic membership changes, site failures in the slave or master nodes and cases where failed nodes come back up and automatically want to re-join the distributed system.

4.1 Algorithm

The synchronization algorithm requires slaves to frequently send time synchronization requests to the master and adjust their time accordingly on receiving a successful and timely reply. The time adjustment is implemented using our Logical Clock class which contains the local hardware clock reading and the adjustment function.

The following variables definitions are used in describing Cristian‟s algorithm :

U	=	Round-trip acceptability threshold
k	=	Max. number of successive synchronization attempts
W	=	Wait time between synchronization re-try attempts
synchPeriod	=	Wait time from current successful synchronization to sending the next synchronization request
bfrT	=	Slave‟s time before sending synchronization request to master.
aftrT	=	Slave‟s time when receiving synchronization reply from master.
D	=	(aftrT – bfrT)/2

ET	=	Master‟s time sent in the synchronization reply.
HC	=	Hardware clock
N	=	Adjustment parameter = (ET + D) – aftrT
m	=	Adjustment parameter = 0 (since amortization is ignored).
A(t)	=	Adjustment function = m*HC + N
LC	=	Logical Clock = HC + A(t) = HC + (m*HC + N)

For brevity I will only present a quick summary of the synchronization protocol as the detailed explanation can be found in the Appendix section of Cristian‟s paper.

A slave requests for time synchronization by executing the following steps:

Record bfrT

Send synchronization request to master " time=?;< seq no.> "

Wait for synchronization reply from master " time= ;<ET> ;< seq no.> "

Record aftrT once reply arrives.

Calculate D and adjust N = (ET + D) – aftrT;

Schedule next DoS attack information and next synchronization request

Synchronization is considered to have failed when D > U or the sequence number of the reply doesn‟t match the sequence number of the request. After sending the request, the slave waits for W and if it hasn‟t received a reply until then, it goes into retry attempts up to k times. If it hasn‟t been synchronized even after the maximum successive attempts, the slave pretends to leave the system and re-join it by re-initializing itself.

On every successful synchronization attempt, the adjustment parameter, N is updated and the time left for the next DoS attack is computed. The goal is to ensure that the slaves synchronize their clocks with the master often enough to send DoS attack packets every 500ms at the same time.

The synchronization messages (requests and replies) were specified in the fixed format:

Message Type	Format
Synchronization request message	time=?;<seq no.>
Synchronization reply message	time=;<ET>;<seq no.>

For simplicity, the format is fixed and all messages were assumed to be valid and we did not incorporate error checking, validation and message corruption handling.

4.2 Roles

The Master‟s role was to listen for time synchronization requests and spawn response threads for each new request. The entire implementation was event-based with the help of timers and socket listeners (ServerSocket in Java). This project containing such time-sensitive components, there was a concerted effort to ensure that at any time there were no threads waiting in an idle-loop and consuming system resource while another thread was ready to do useful execution. The Master response thread would live to serve a single synchronization request and reply back to the request.

The Slave‟s role was to periodically synchronize time with the master by sending it requests. Upon successful synchronization, the slave would update the adjustment parameter of its logical clock. At this time the slave had the latest time information, so it would also re-configure the timers and adjust the period to send the next batch of DoS attacks. On each successful synchronization reply, the schedule of the DoS attacks was recalculated. The Slave used the SynchTimerTask, AttemptTimerTask, SendDosTimerTask and TerminateSlaveTimerTask classes to schedule and fire time-based events. Details on the function of each class can be found in the comments of the source code.

5. DENIAL OF SERVICE:

In this homework, our team attacked an end host in 2 different ways. Once we got the synchronization protocol working we proceeded to attack the end host by sending network packets at the same time from all attacking computers to the victimized end host using:

i. User Datagram Protocol

   UDP packets were sent to fixed ports by each attacking slave on the victimized end host at the same time every 500ms. As a result, the victimized host checks for the application listening at that port and sees that no application listens at that port and subsequently replies with an ICMP Destination Unreachable packet. Thus, for a large number of UDP packets, the victimized system will be forced into sending many ICMP packets, eventually leading it to be unreachable.[4]

ii. Transmission Control Protocol

The DoS attack consisted of getting a number of computers to send valid HTTP 1.1 requests for a simple page (index.htm) over TCP to the end host every 500ms. The end host becomes unusable after some time because every TCP connection establishment requires an allocation of significant memory resources and as such if the attacking computers are synchronized and send all these TCP packets at the same time, the attacked end host‟s performance is severely degraded as it would not be able to handle all the incoming TCP connections.

6. RESULTS AND OBSERVATIONS:

The goal of this section is to present data from our runs in PlanetLab and verify our results. We want to able to answer two main questions:

1. Was synchronization correctly implemented and successful?
2. Was the DoS correctly implemented and did it work?

In addition to the above questions, we also attempt to look at scalability issues, precision issues and performance and system utilization issues.

To supplement our results and help us prove our success, the submission also contains the „*SampleOutput*‟ folder. This folder consists of actual slave log files from multiple PlanetLab runs. The „SampleOutput‟ folder has been organized into two sections:

1. TimeSynchVerification: This folder deals with logs used to verify that synchronization was correctly implemented and was successful.
2. DoSVerification: This folder deals with logs used to verify that DoS was correctly implemented and was successful.

Each of the folders contains log files from multiple PlanetLab executions with varying number of slave nodes (min 1, max 50).

6.1 Time Synchronization Verification

The „TimeSynchVerification‟ folder consists of log from running our distributed system on 1, 5, 10, 24 and 29 slaves in PlanetLab. The purpose of these log files is to prove the correct implementation of the time synchronization protocol. For simplicity the above mentioned runs were executed by disabling the DoS attack port on the target machine so that the DoS related logs could be ignored and we could focus on time synchronization alone.

We answer the following questions:
Please note that this section references variables names defined in section 4.1 and class and data structure names defined in the source code.

Did the slaves send time synchronization requests?

Yes. All time synchronization requests sent by the slaves were logged in the following format:

<Synch | Attempt>: Sending query: time=?;<seq no.>

Where "Synch" meant a new request was being sent and „Attempt" meant a retry attempt was being made as we had timed out waiting for a response for the original synch request.

How often did the slaves send synchronization requests and how many were successful?

On average we had one successful synchronization completed every 350~500ms (for the parameters U=350ms, k=10, W=1000ms, synchFrequency=300ms and AttackLifetime=5mins). We computed the total number of synchronization requests sent (both "Synch" and „Attempts") and then calculated the number of those requests that succeeded. This consistently gave us a synchronization success rate above 90%. We divided the number of successful synchronization requests over the attack lifetime to get the above number (average gap between time synchronization).

How many of the time synchronization succeeded?

Consistently around and above 90% as suggested in the description of the synchronization success rate above. We attribute this high rate to selecting a high value for U and synchFrequency. Selecting lower values for U and the synchFrequency would cause messages to be sent more often and expected to be received earlier. This would lead to more synchronization failures especially when scaling to higher number of slaves. In order to keep our adjustment parameter, N updated often enough, we decided to keep a higher number (yet < 500ms) for our U.

Each synchronization reply received was also logged in the following format:

```
Received Reply: time=;<ET>;<seq no.>
```

Did the master receive and respond to the synchronization requests?

Yes. We did not include logs for the master node but the fact that the slaves received the response implies that the master was able to receive and respond to those synchronization requests.

Each successful synchronization in the slave was also logged in the following format:

```
Synch Success: SAfter= <aftrT>, SBfr=<bfrT>, D=<D>, M=<ET>
```

How well did the master manage scalability issues when contacted from increasing number of slaves?

We saw no significant delay in receiving replies when increasing the number of slaves. The master spawned a new response thread that replied for a single time synchronization request and then terminated itself. There was no significant increase in average values of D when the number of slaves was increased. For example: planetlab2.csail.mit.edu consistently saw average values of D around 70~80ms when run with 5 as well as 29 slaves.

How often were the slave's logical clocks updated?

As described above, we divided the number of successful synchronization requests over the attack lifetime to compute the average gap between each successful time synchronization. Since the adjustment parameter, N was edited upon each successful synchronization, this number was the same as described above, 350~500ms.

Did all the slaves attempt to send the DoS attack every 500ms?

Yes. Each DoS attack was logged in the following format:

```
DOS THREAD: SEND DoS PACKET AT:<local logical Clock time> <DoS packet number>
```

We divided the total number of DoS packets sent over the attack lifetime (5 minutes) and consistently saw values around ~593 (out of a total possible 600). Some of the PlanetLab nodes did take time to start up and the first and last few seconds of the slave execution were used to initialize and terminate the data structures where no packets were sent.

Did all the slaves attempt to send the DoS attacks together at the same logical time?

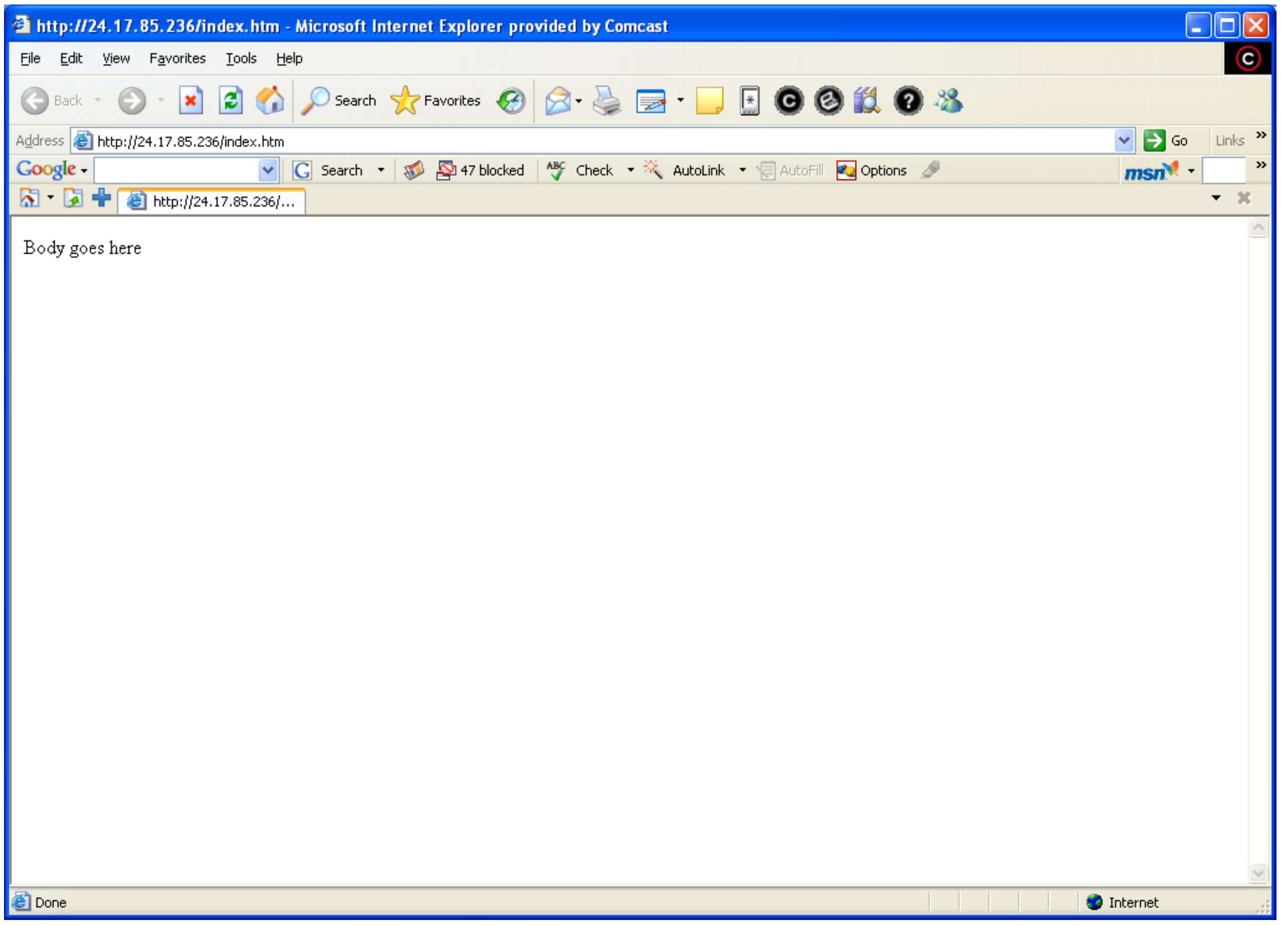
Yes. Each DoS attack contained the local logical clock time during which it was sent. We verified that all slaves consistently sent packets around times that were multiples of 500ms (for e.g.: 1146625740501 and 1146625741002). The slight high number from the multiples of 500ms was attributed to the time spent in the SendDosAttackTimerTask class in actually preparing and sending the packet. Since all slaves sent a packet every 500ms (as verified above) and all the packets send around the same number of packets over the attack lifetime, it can be concluded that they all sent the packets at the same time.

Answering the above questions with log files gives us ample confidence that for our practical purposes, the time synchronization protocol was implemented successfully.

6.2 DoS Verification

The „DoSVerification" folder consists of log from running our distributed system on 10, 25, and 50 slaves in PlanetLab. The purpose of these log files is to prove the successful use of the DoS attack to bring down the attacked machine. We were able to bring down the website with 10 PlanetLab slaves. For simplicity we have only included log files from Attack mode of TCP on port 80 of one of our home machine. More PlanetLab runs were conducted but their logs have not been included.

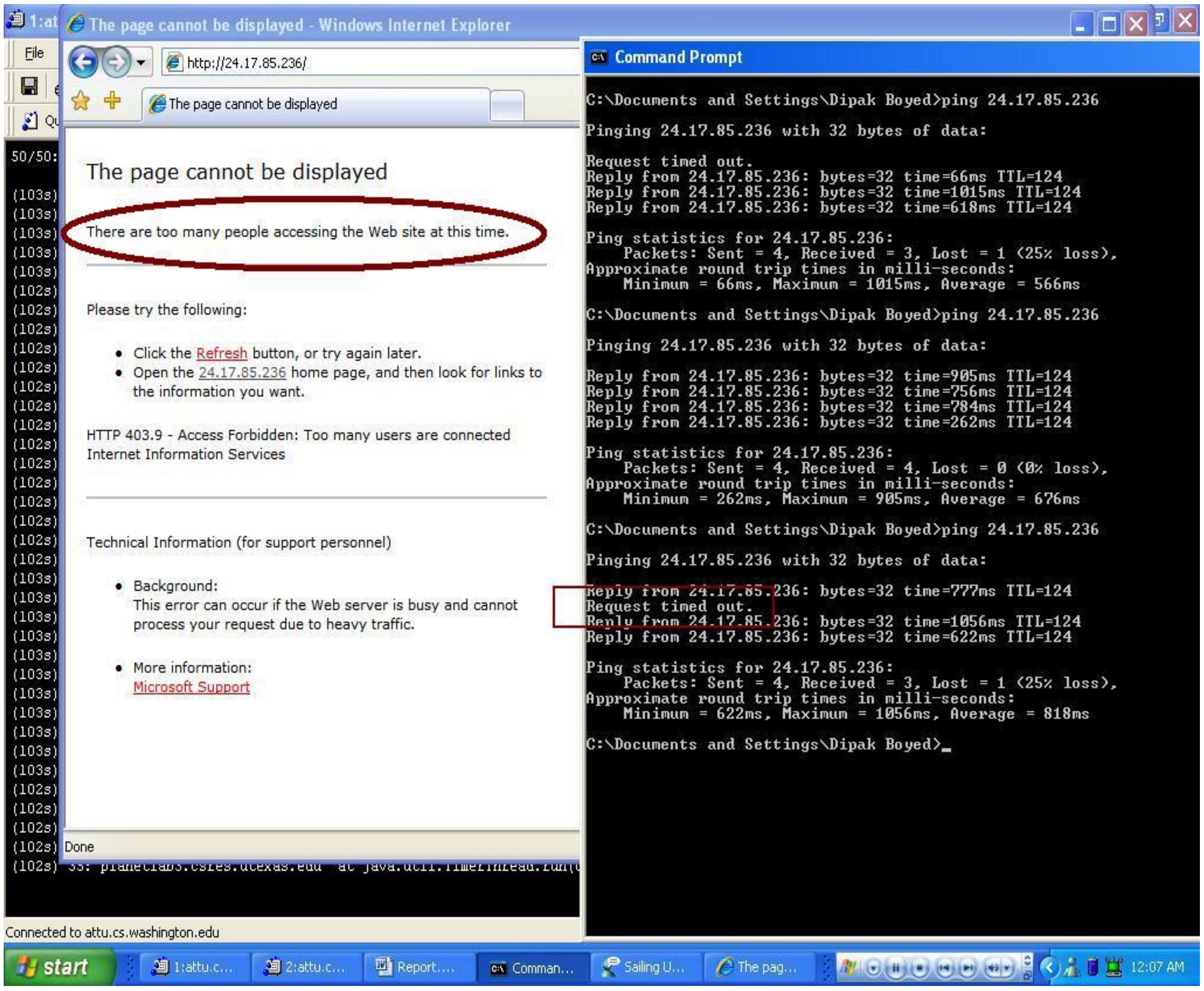
The IP Address of the victimized host was 24.17.85.236. There was a simple web page hosted on the victimized machine. The victimized machine was a Pentium III (1.0 GHz) PC with 384 MB of memory running Windows XP SP2. The URL for the page was http://24.17.85.236/index.htm. Legitimate visitors to the page should be served with the following simple html page. (Figure 6.1)


http://24.17.85.236/index.htm - Microsoft Internet Explorer provided by Comcast
File Edit View Favorites Tools Help
Back Search Favorites
Address http://24.17.85.236/index.htm Go Links
Google Search 47 blocked Check AutoLink AutoFill Options msn
http://24.17.85.236/...
Done
Internet

Body goes here

Figure 6.1

After synchronizing the clocks of the planet lab nodes, we were able to get **10 machines** requesting for the same page every 500ms. This caused the page to be unavailable from other legitimate visitors to the site. There was a HTTP 403.9 error with the message “Too many users are connected”. Pinging the machine also timed out. (see Figure 6.2).



The page cannot be displayed - Windows Internet Explorer

http://24.17.85.236/

The page cannot be displayed

There are too many people accessing the Web site at this time.

Please try the following:

- Click the Refresh button, or try again later.
- Open the 24.17.85.236 home page, and then look for links to the information you want.

HTTP 403.9 - Access Forbidden: Too many users are connected
Internet Information Services

Technical Information (for support personnel)

- Background:
  This error can occur if the Web server is busy and cannot process your request due to heavy traffic.
- More information:
  Microsoft Support

Done

```
50/50:

(103s)
(103s)
(103s)
(103s)
(103s)
(102s)
(102s)
(102s)
(102s)
(102s)
(102s)
(102s)
(102s)
(102s)
(102s)
(102s)
(102s)
(102s)
(102s)
(102s)
(103s)
(103s)
(103s)
(103s)
(103s)
(103s)
(103s)
(103s)
(103s)
(103s)
(103s)
(102s)
(102s)
(102s)
(102s)
(102s) 33: planetlab3.csres.utexas.edu  at java.util.TimerThread.run(
```

Connected to attu.cs.washington.edu

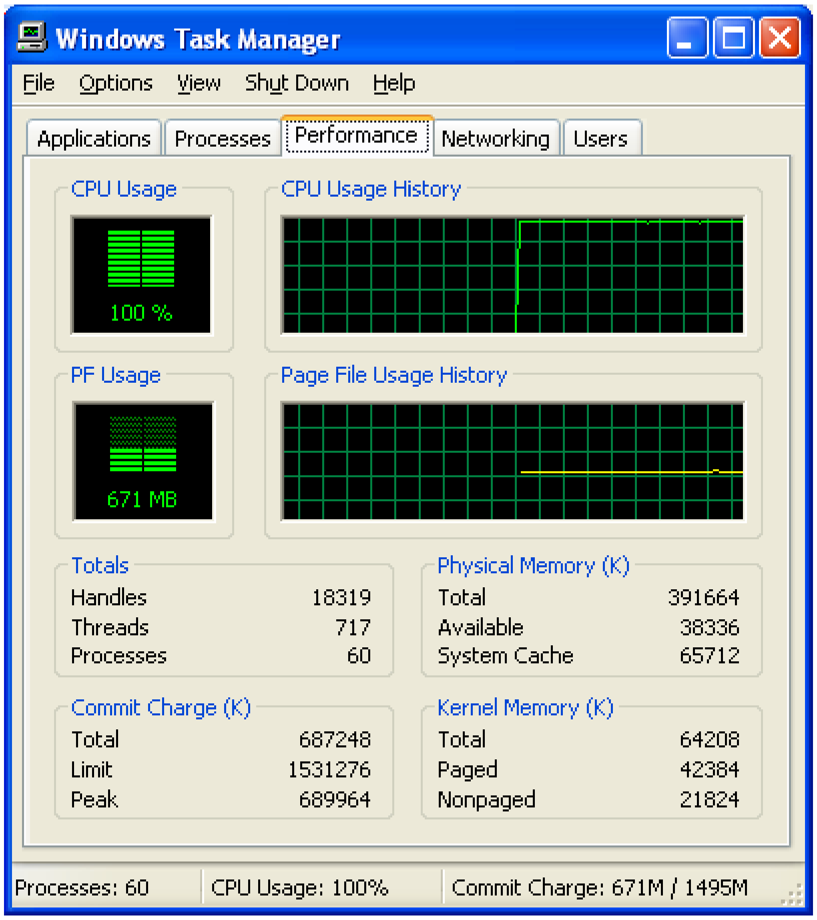
Command Prompt

```
C:\Documents and Settings\Dipak Boyed>ping 24.17.85.236

Pinging 24.17.85.236 with 32 bytes of data:

Request timed out.
Reply from 24.17.85.236: bytes=32 time=66ms TTL=124
Reply from 24.17.85.236: bytes=32 time=1015ms TTL=124
Reply from 24.17.85.236: bytes=32 time=618ms TTL=124

Ping statistics for 24.17.85.236:
    Packets: Sent = 4, Received = 3, Lost = 1 (25% loss),
Approximate round trip times in milli-seconds:
    Minimum = 66ms, Maximum = 1015ms, Average = 566ms

C:\Documents and Settings\Dipak Boyed>ping 24.17.85.236

Pinging 24.17.85.236 with 32 bytes of data:

Reply from 24.17.85.236: bytes=32 time=905ms TTL=124
Reply from 24.17.85.236: bytes=32 time=756ms TTL=124
Reply from 24.17.85.236: bytes=32 time=784ms TTL=124
Reply from 24.17.85.236: bytes=32 time=262ms TTL=124

Ping statistics for 24.17.85.236:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 262ms, Maximum = 905ms, Average = 676ms

C:\Documents and Settings\Dipak Boyed>ping 24.17.85.236

Pinging 24.17.85.236 with 32 bytes of data:

Reply from 24.17.85.236: bytes=32 time=777ms TTL=124
Request timed out.
Reply from 24.17.85.236: bytes=32 time=1056ms TTL=124
Reply from 24.17.85.236: bytes=32 time=622ms TTL=124

Ping statistics for 24.17.85.236:
    Packets: Sent = 4, Received = 3, Lost = 1 (25% loss),
Approximate round trip times in milli-seconds:
    Minimum = 622ms, Maximum = 1056ms, Average = 818ms

C:\Documents and Settings\Dipak Boyed>_
```

Figure 6.2

The *.err log files (in the 10, 25 and 50 slave runs) also indicate that due to excessive overload many of the DoS packets send attempts threw “Connection refused” exceptions. This further shows that the targeted machine was unable to handle such massive amount of requests by the low-rate (shrew) DoS attacks.

Increasing the number of planet lab nodes to 25 slowed down the performance of the host considerably and brought CPU utilization to about 50%. However when the number of planet lab nodes was increased to 50, the computer became quite unusable as CPU utilization peaked to 100% (See figure 6.3). Ethereal (a network protocol analyzer) results showed that 109,857 TCP packets were sent to port 80 in the first 2 minutes of the attack time (see Figure 6.4)



Totals	
Handles	18319
Threads	717
Processes	60

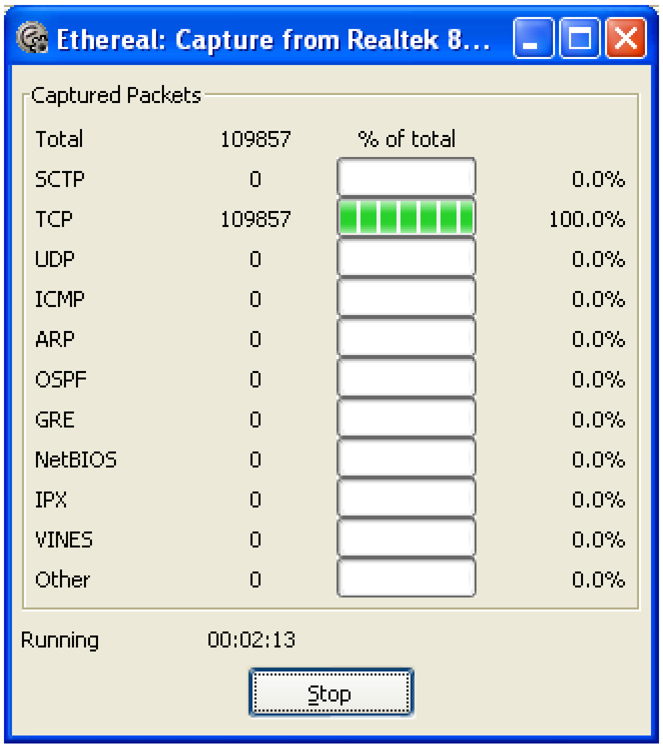
Physical Memory (K)	
Total	391664
Available	38336
System Cache	65712

Commit Charge (K)	
Total	687248
Limit	1531276
Peak	689964

Kernel Memory (K)	
Total	64208
Paged	42384
Nonpaged	21824

Processes: 60 CPU Usage: 100% Commit Charge: 671M / 1495M

Figure 6.3



Captured Packets

Total	109857	% of total	
SCTP	0		0.0%
TCP	109857	🟩	100.0%
UDP	0		0.0%
ICMP	0		0.0%
ARP	0		0.0%
OSPF	0		0.0%
GRE	0		0.0%
NetBIOS	0		0.0%
IPX	0		0.0%
VINES	0		0.0%
Other	0		0.0%

Running 00:02:13

Figure 6.4

To summarize, when carrying out the attack using TCP it took about 50 planet lab nodes to make the victimized host unusable while as few as 10 labs caused the web page served to become unavailable.

We proceed to carry out the UDP portion of the attack. 25 planet labs were able to cause a significant jump in CPU utilization to about 50%. This was because the victimized host had to send ICMP packets to let the senders know that there is no application listening on that port and as such the resources on the machine were eventually depleted. For brevity, the logs from the UDP attack mode have not been included.

7. CONCLUSIONS:

We implemented a simplified version of Cristian‟s probabilistic time synchronization protocol to allow multiple slaves to synchronize their logical clocks with an internal master node. The synchronization protocol was used to co-ordinate a low-rate Denial of service (DoS) attack on a target end host (local machine) with a 500ms periodic attack.

Our distributed system was ported to PlanetLab nodes and was executed for multiple topologies (minimum of 1 slave, maximum of 50 slaves). We listed our assumptions made during the implementation and verified that the time synchronization protocol was correctly implemented for all practical purposes. The PlanetLab executions were used to co-ordinate a DoS attack on our home computer. We found that 10 slaves were enough to bring down our website from servicing legitimate customers over a period of 5 minutes and with 50 slaves the target machine‟s CPU utilization shot up to 100%.

---

[1] F. Cristian, A Probabilistic Approach to Clock Synchronization, Proceedings of the 9th International Conference on Distributed Computing Systems (ICDCS), pages 288-296, Newport Beach, California, June 1989.

[2] A. Kuzmanovic, E. Knightly, Low-Rate TCP-Targeted Denial of Service Attacks, SIGCOMM, Karlsruhe, Germany, August 2003.

[3] Peterson, Anderson, Culler, and Roscoe. A Blueprint for Introducing Disruptive Change into the Internet. HotNets 2002

[4] http://en.wikipedia.org/wiki/UDP_flood_attack