R&D Project Report On

IIT Bombay Web Traffic Characterization

By

Nirav S. Uchat (06305906)

Under the guidance of

Prof. Purushottam Kulkarni
Computer Science and Engineering Department,
Indian Institute of Technology, Bombay
Mumbai

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai

Table of Contents

Chapter 1: Introduction **6**

1.1. Internet and Web Proxy *6*

1.2. IIT Bombay Web Proxy *7*

Chapter 2: Data Collection and Processing **9**

2.1. Raw Data *9*

2.2. Data Processing *10*

2.3. Summary *10*

Chapter 3: Experiments and Results **11**

3.1. HTML vs. Image Content *11*

3.2. HIT Ratio *11*

3.3. File Size Distribution *12*

3.4. Segment wise usage *13*

3.5. Department wise Traffic *14*

From previous stats we conclude that EE department is responsible for 20% of total academic traffic. *14*

3.6. Traffic by type (GET, POST and CONNECT) *14*

3.7. Daily average object size *15*

3.8. Load on each Proxy Servers *16*

3.9. Distinct user count *16*

3.10. Number of requests per user on single day *17*

3.11. Usage in every 2 hour interval on a single day *17*

3.12. Number of user having specific usage per day *18*

3.13. Requests With/Without Proxy Authentication *18*

3.14. Top domain on a given day *19*

3.15. Requests in 1 min interval for 24 hours *20*

3.16. Number of proxy hit by user sorted by frequency *20*

3.17. Usage of user sorted by usage *21*

3.18. Concentration Graph *22*

3.19. Distribution by HTTP response code *22*

3.20. Distinct Request type on a given day *23*

3.21. Distribution by content Type *24*

3.22. Summary *24*

Chapter 4: Cache Behavior **25**

4.1. Introduction *25*

4.2. IIT Bombay Setup 25

4.3. Experiment and Results 26

4.3.1. Single Server LRU 26

4.3.2. Merge Server LRU 27

4.4. Summary 27

5.1. What is Bombardment? 28

5.2. Detecting Bombardment 29

5.3.1. Algorithm 29

5.3.2. Implementation 29

5.3. Summary 31

Chapter 6 - Conclusion and Future Work 32

Bibliography 33

APPENDIX A 34

A.1. Code for data processing 34

A.2. Code for cache simulator 35

A.3. Code for bombardment detection 40

Acknowledgements

I would like to thank Prof. Purushottam Kulkarni for his constant support and guidance during the course of this project. I would also like to thank Computer Center, IIT Bombay for providing high end blade servers to carry out intensive log processing.

Abstract

The growth of internet is phenomenal and is expanding at rapid rate. Starting from early static web pages to current popular site such as Youtube[3], Liveleak, it is evident that more and more data is flowing through already congested internet pipe. In order to fulfill user's requests in timely manner, ISP uses large web proxy server to cache user data. The performance of web proxy highly depends on usage pattern and cache behavior which includes cache replacement protocol. This project is aimed at characterizing IIT Bobmay's web traffic and coming up with various invariant such as traffic distribution which holds irrespective of time.

We will also study the effect of shared cache on hit ratio with varying cache size. Finally we characterize proxy bombardment in IIT Bombay network and give working prototype to prevent it.

Chapter 1: Introduction

1.1. Internet and Web Proxy

In early days of 80's, Internet was catering academia and defense installation with limited number of user count. As personal computer became common, more and more user started joining the network and large amount of traffic started flowing across globe. As technology advanced, came the web browser which allowed user to request pages which are geographically away. To fulfill user request in timely manner, ISP's[1] started using web proxy to cache user data.

Web Browser uses client server model where client[2] request a object from server[3] and server sends back answer in reply which is a generally index page of requested site. However response of the server might be slow especially for servers that are far away from the client or connected through a slow network link or busy network. One way to tackle it is using proxy server which act as intermediate server for client and client for actual server. Following is the sequence of message passing,

1. Original client send request to proxy server
2. Proxy server processes the request
3. If requested object reside in proxy cache and it is not modified since it was in cache then proxy server send object to client, if not then proxy server send request to actual server
4. When proxy gets the new requested object it first stores it in cache[4] and then send it to client.

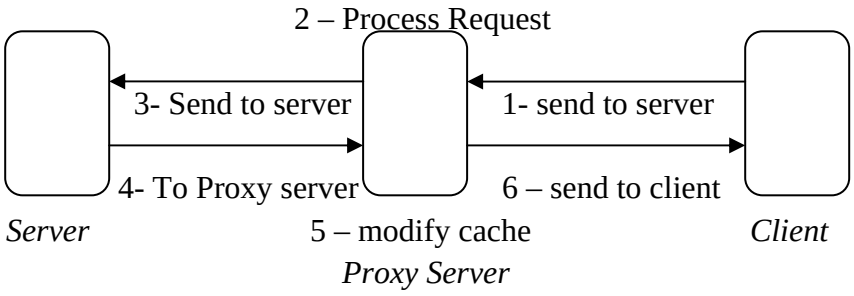


Figure 1 – Web Proxy

If client request follows a specific pattern then tuning proxy server might increase the overall response time.

[1] ISP is acronym for Internet Service provider
[2] client in this case is user requesting a specific object such at www.news.com
[3] Server is hosting www.news.com on web server like Apache, IIS etc.
[4] If cache is full then cache replacement algorithm such as LRU is used to make place for new object in cache

1.2. IIT Bombay Web Proxy

At present five thousand users uses IIT Bombay internet service. In order to fulfill large number of request it uses ultra monkey[5] for load balancing. The architecture of web proxy is as follows,

1) There are two load balancer one is active while other is in standby mode; if active server fails, load balancer service is migrated to standby server.
2) There are four proxy server each having its own cache.
   a) IP range for proxy server is 10.XXX.XX.1 to 10.XXX.XX.4
3) One virtual interface on load balancer to give one entry point to proxy servers.
   a) The virtual IP is 10.XXX.XX.20
4) Load balancer manages the virtual interface and scheduling policy for each request
   a) Client send request to 10.XXX.XX.20
   b) It is then distributed to any one of the four proxy server based on scheduling policy
5) From log it is evident that it uses round robin scheduling
6) Open source proxy - squid[6] is used on each server
7) Squid guard[7] is used for blocking access to specific URL

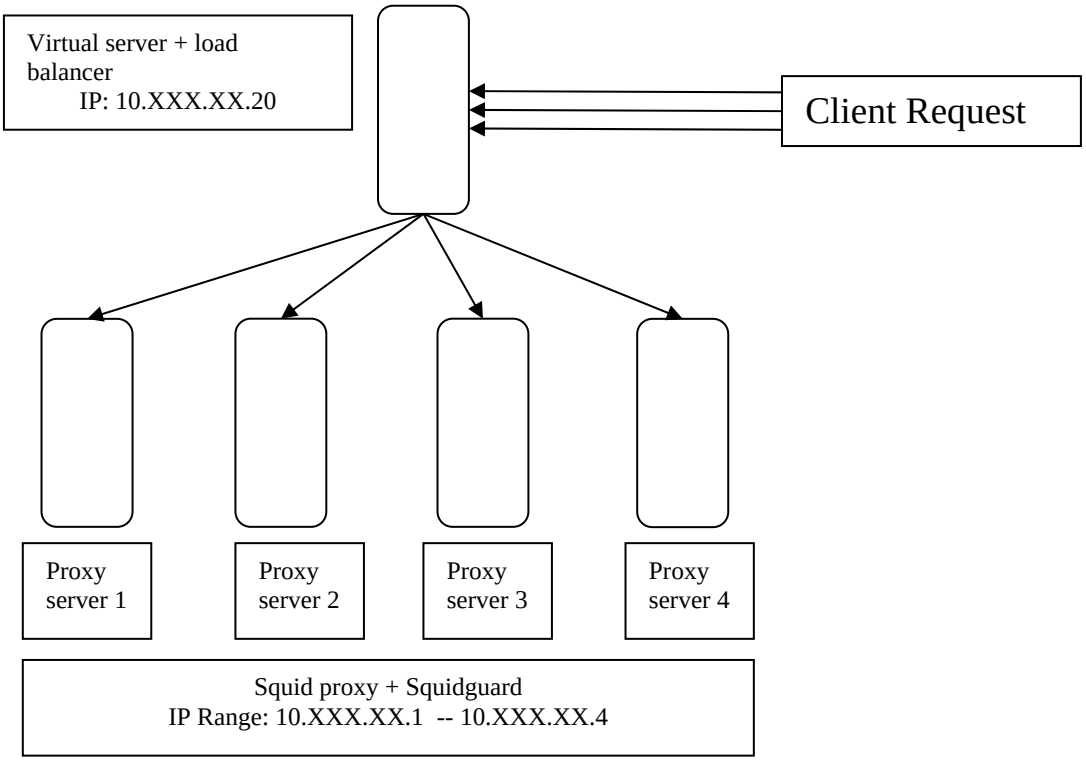


Figure 2 – Proxy Server Architecture

[5] Ultra Monkey is used to create highly scalable and highly available services - www.ultramonkey.org
[6] Squid is a caching proxy for the Web supporting HTTP, HTTPS, FTP etc. http://www.squid-cache.org
[7] SquidGuard is a URL redirector used to use blacklists with the proxy software Squid.

This gives seamless single log file order by access time. Later in chapter 3, we will use this raw file to plot various statistics.

Apart from proxy server, IIT Bombay has 32Mbps lease line from two different ISP. It also has its own DNS and SMTP server. It uses IPTBALE as firewall. There are two different machines handling inbound and outbound traffic.

Chapter 2: Data Collection and Processing

2.1. Raw Data

As discussed in chapter 1, all four proxy servers access log is merged in to single log file and it is ordered by access time. There is one line for each client request. Following is the format of log file,

Year Month Date Time Proxy_Server squid_process_id epoc_timestamp time_ms source_ip tcp_status/tcp_code object size request_type URL user_id object_type object_sub_type

Following is single line from raw log file,

2008 Jan 1 00:00:00 nm1 squid[25879]: 1199125800.015 5 10.163.50.44 TCP_MEM_HIT/200 1864 GET http://www.ndtv.com/playing_caption.jpg XXXX NONE/- image/jpeg

Where XXXX is user_id. For user privacy, in all log processing user_id has been hashed.

Below is the statistics of raw files used in generating various invariant presented in chapter 3.

Log Duration	7 days
Start Date / End Date	1 Jan'08 – 7 Jan'08
Total Raw Log Size	27.3 GB
Number Of Request / Day	16 Million (Approx)
Total Request for 7 Days	110 Million (Approx)

From above raw data some field are not required and are removed before log processing. A small Perl script take original log file as input and gives new truncated file as output. It also hashes the user name and format date which can be handle by MySql database. MySql database is used to store whole log file. Below is the structure of mysql table.

Field Name	**Type**
access_date	Date
access_time	Time
netmon_server	varchar(4)
process_time_ms	int(10) unsigned
source_ip	varchar(15)

tcp_status	Varchar(20)
tcp_status_code	smallint(6)
object_size	int(10) unsigned
request_type	varchar(15)
Domain	varchar(60)
user_id	varchar(35)
server_fetch_type	varchar(25)
server_ip	varchar(15)
object_type	varchar(20)
object_sub_type	varchar(20)

This table is used in chapter 3 to generate required statistics.

Second log file is 407.log, which log the invalid proxy request. To understand it in better way, IIT Bombay has LDAP authentication for user before accessing the proxy. In usual case user enter the username and password as and when asked by the system, but some software like automatic update, antivirus virus signature update tries to fetch data from server without proxy credential. In such proxy server has to deny it i.e. return code 407(TCP_DENIED). Chapter 5 discuss about handling such request. The structure of 407.log is same as access log file discussed above.

2.2. Data Processing

Perl Scripts is used to convert raw data file to file which can be loaded in to MySql table. Furthermore, java code is used to plot concentration graph discussed in chapter 3. Also a cache simulator in JAVA is used to study effect of shared cache on hit ratio with varying cache size[8]. Mixture of Perl and Python script is used for bombardment monitoring and alert mechanism discussed in chapter 5.

2.3. Summary

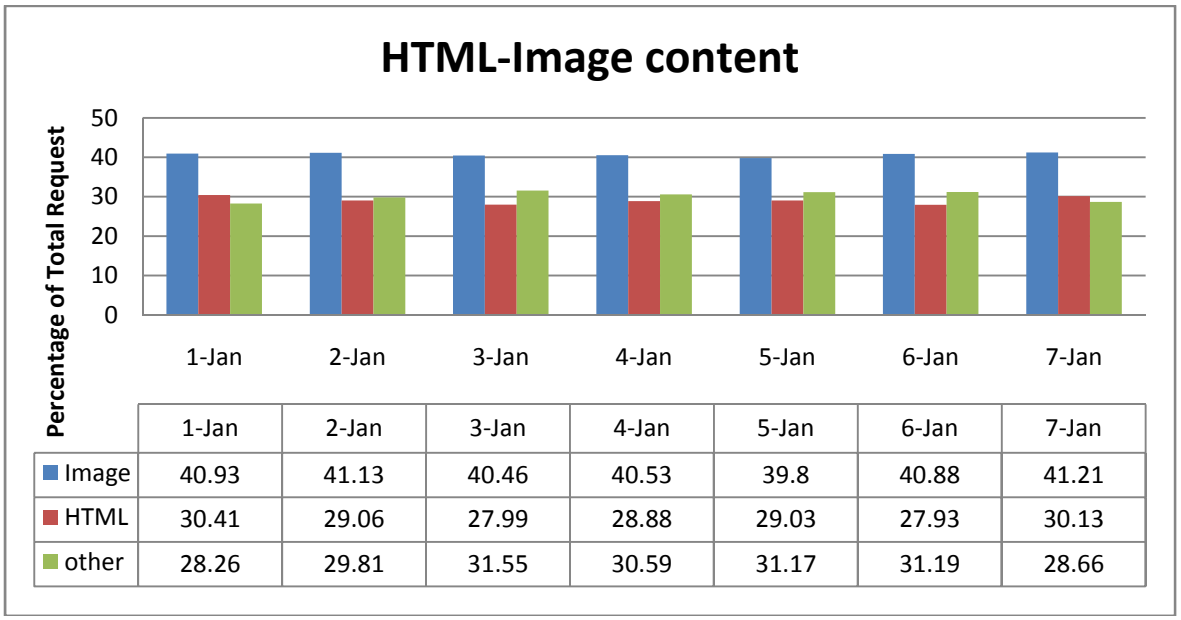
In this chapter we have looked at raw data statistic and how it is converted to required format for further processing. In chapter 3 we will use converted data to get detailed statistics.

[8] More on this is discussed in chapter 4. Code is in Appendix A.3

Chapter 3: Experiments and Results

3.1. HTML vs. Image Content

As we know there are various file formats which are requested by user from web. In this experiment we find large numbers of requests are for Image and HTML pages. More precisely we found that @40% and @30% of total request is for Image and HTML respectively. This behavior is more or less same for 7 day period.



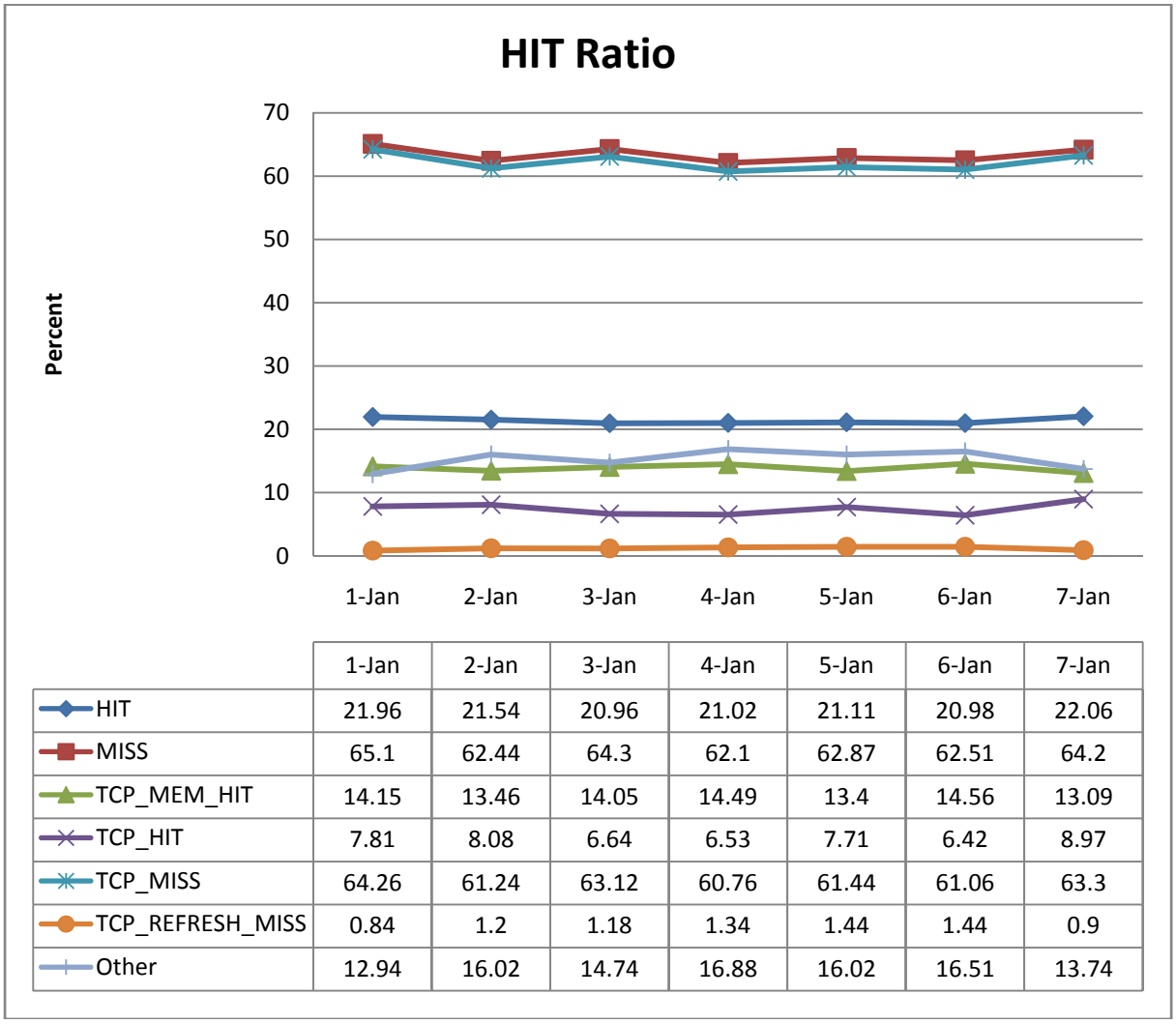
	1-Jan	2-Jan	3-Jan	4-Jan	5-Jan	6-Jan	7-Jan
Image	40.93	41.13	40.46	40.53	39.8	40.88	41.21
HTML	30.41	29.06	27.99	28.88	29.03	27.93	30.13
other	28.26	29.81	31.55	30.59	31.17	31.19	28.66

In above graph x-axis represent day and y-axis represent percent of total requests. Green bar in above graph represent other file format such as pdf, audio, video and so on.

3.2. HIT Ratio

HIT ratio is defined as number client request fulfilled by proxy cache to the total number of requests received by proxy. It implies that higher HIT ratio is favorable as it reduces bandwidth usage and improves client response time. Below graph shows the hit ratio observed on seven day period.

TCP_HIT	Objects found in proxy cache (on secondary storage)
TCP_MEM_HIT	Object found in proxy cache RAM
TCP_MISS	Object not found in cache
TCP_REFRESH_MISS	IF_MODIFY_SINCE returns false
other	Other proxy response
HIT	Sum of TCP_HIT and TCP_MEM_HIT
MISS	Sum of TCP_MISS and TCP_REFRESH_MISS

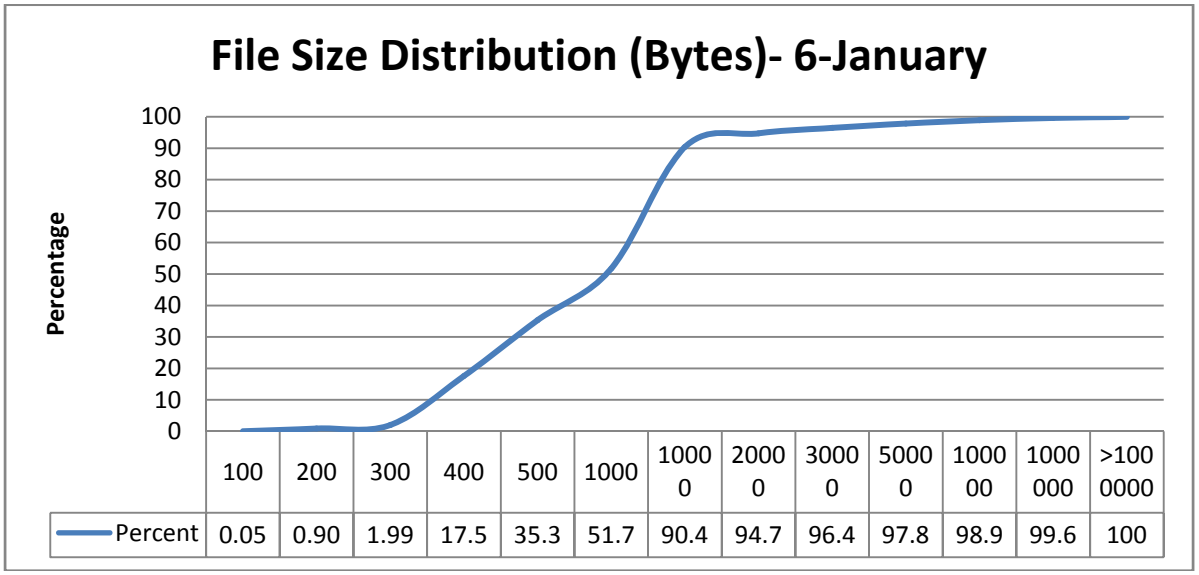


	1-Jan	2-Jan	3-Jan	4-Jan	5-Jan	6-Jan	7-Jan
HIT	21.96	21.54	20.96	21.02	21.11	20.98	22.06
MISS	65.1	62.44	64.3	62.1	62.87	62.51	64.2
TCP_MEM_HIT	14.15	13.46	14.05	14.49	13.4	14.56	13.09
TCP_HIT	7.81	8.08	6.64	6.53	7.71	6.42	8.97
TCP_MISS	64.26	61.24	63.12	60.76	61.44	61.06	63.3
TCP_REFRESH_MISS	0.84	1.2	1.18	1.34	1.44	1.44	0.9
Other	12.94	16.02	14.74	16.88	16.02	16.51	13.74

We find that HIT rate is @ 22% and MISS rate is @ 65% on a given day. One reason could be due to number of distinct domain. We found that there are @ 7.5 million distinct domain on given day which might reduce hit rate.

3.3. File Size Distribution

In this experiment we try to find file size distribution. In below plot, x-axis is percentage of file less then given value. For example we find that 40% of total object has size between 1000 bytes to 10000 bytes, which is evident from section 3.7. Another example, 90% of total object has size less than 10000 bytes.

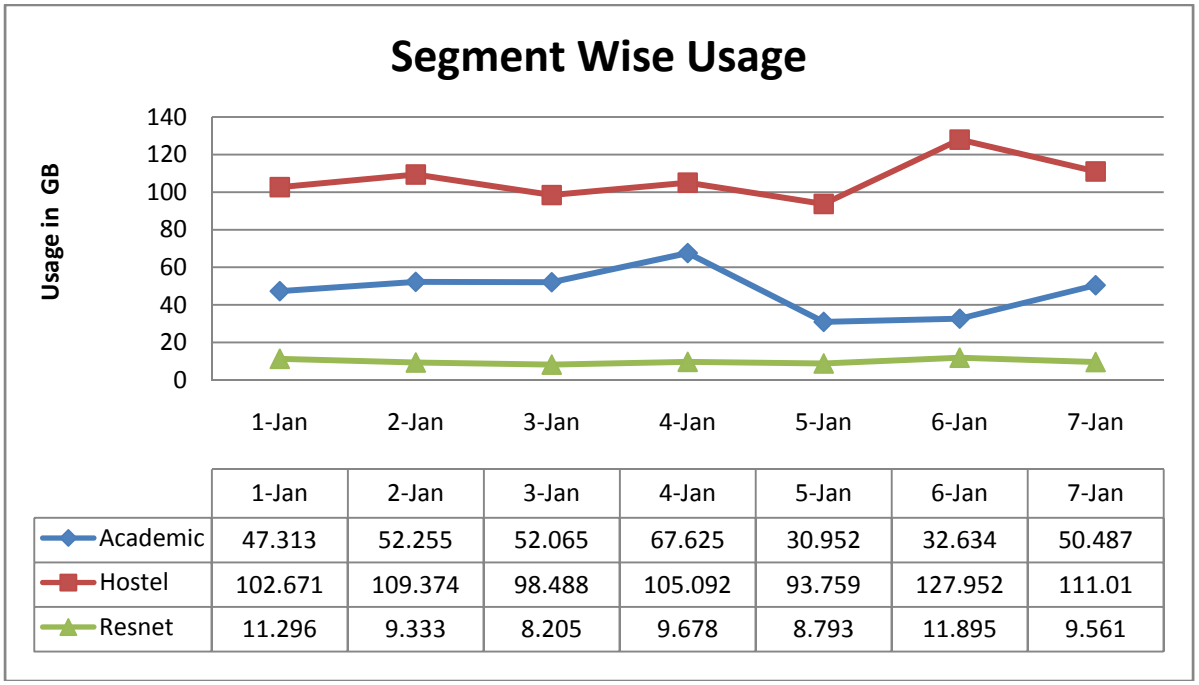


3.4. Segment wise usage

IIT Bombay has three main network segments namely hostel network, academic network and residential network.

Hostel	10.1.XXX.XXX – 10.13.XXX.XXX
Academic	10.101.XXX.XXX – 10.150.XXX.XXX
Residential	10.161.XXX.XXX – 10.165.XXX.XXX

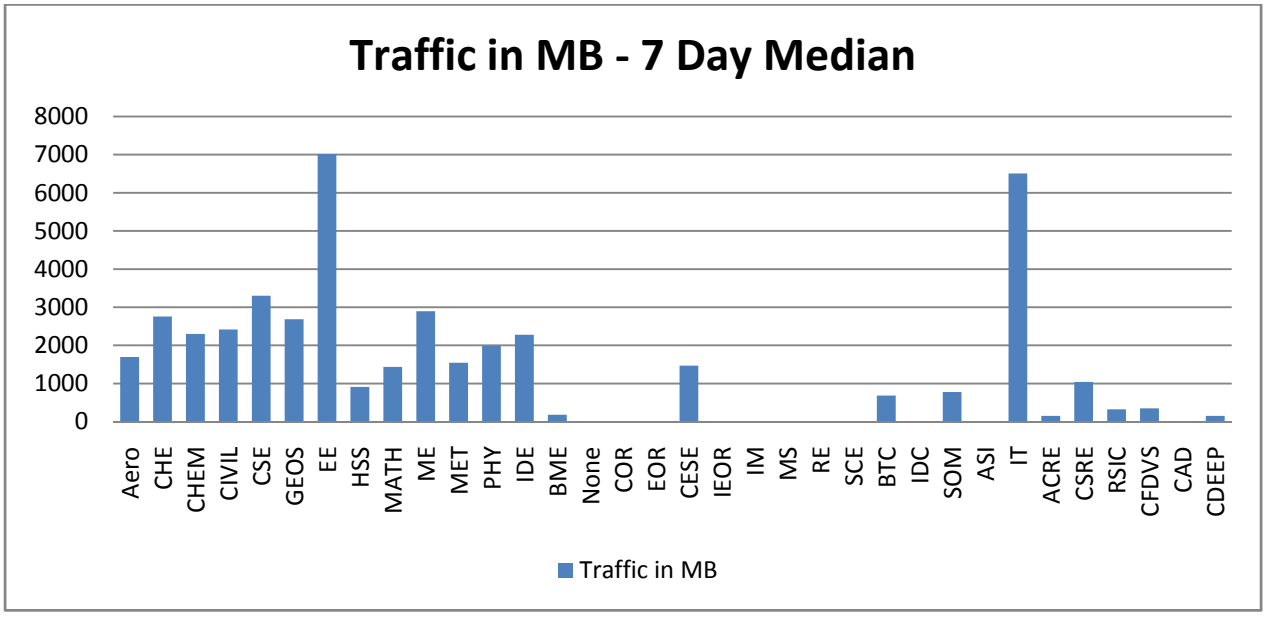
In this experiment we try to find out which segment is responsible for high net usage. Below graph shows maximum traffic on a given day is from Hostel followed by Academic and residential network. In below graph x-axis is a day and y-axis is total usage in GB.



The total usage on a given day is @ 170 GB. We find that hostel network is responsible for 65% of total net traffic followed by academic network which is 30%.

3.5. Department wise Traffic

In this experiment we try to find department which is responsible for high net usage. Below plot shows median of 7 day usage of individual department. X-axis represent department and Y-axis represent size in MB. We find Electrical and Information technology department are top users of proxy server.



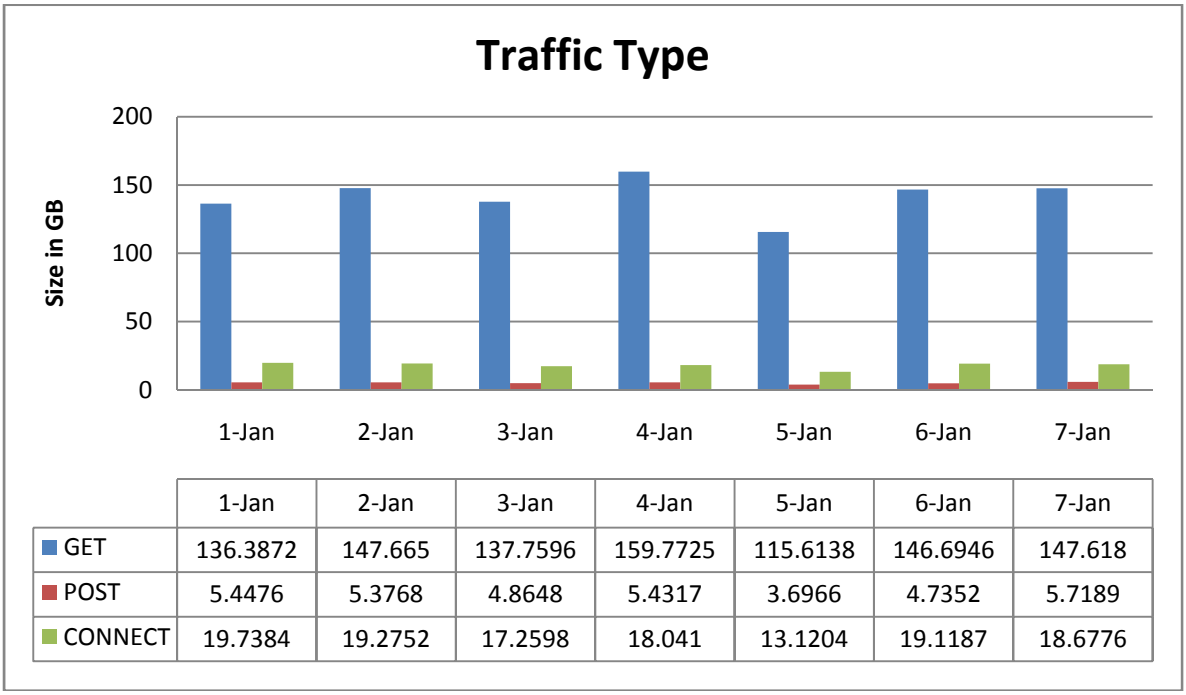
From previous stats we conclude that EE department is responsible for 20% of total academic traffic.

3.6. Traffic by type (GET, POST and CONNECT)

In this experiment we try to classify traffic based on HTTP response code. Below is the definition response code

GET	Get from server (Incoming)
POST	Send to server (Outgoing)
CONNECT	Bidirectional channel

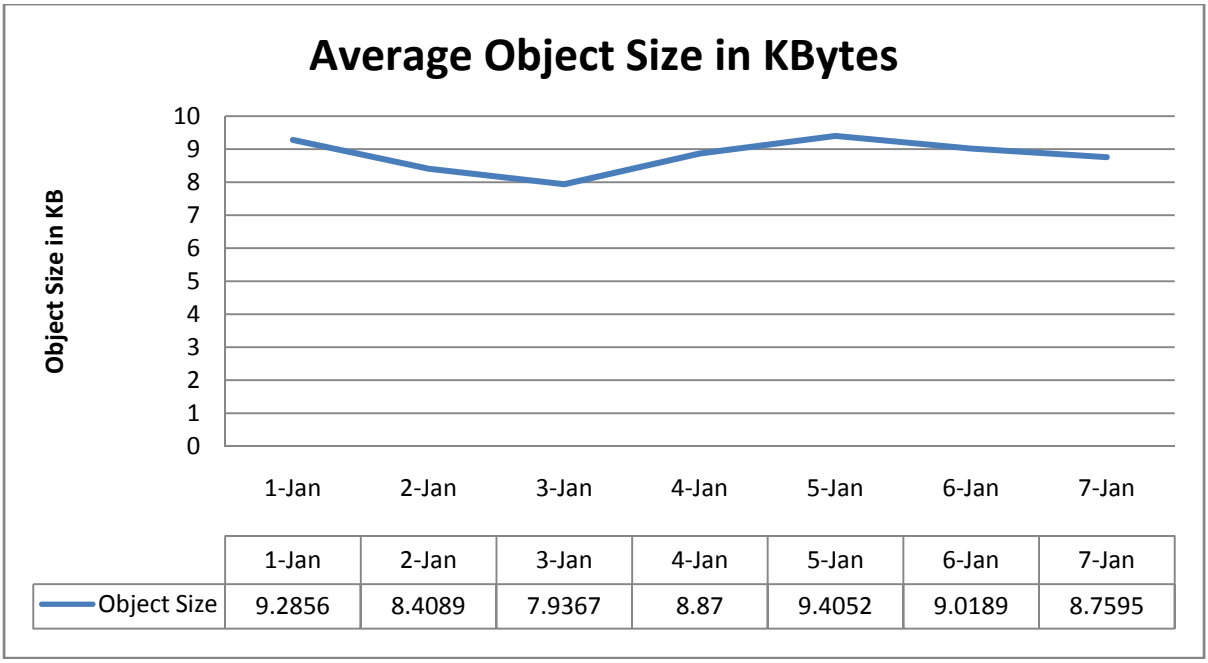
From below graph, GET request is account for maximum traffic. GET request is the request made from IIT Bombay network to outside world. X-axis is day and y-axis is traffic in GB.



	1-Jan	2-Jan	3-Jan	4-Jan	5-Jan	6-Jan	7-Jan
GET	136.3872	147.665	137.7596	159.7725	115.6138	146.6946	147.618
POST	5.4476	5.3768	4.8648	5.4317	3.6966	4.7352	5.7189
CONNECT	19.7384	19.2752	17.2598	18.041	13.1204	19.1187	18.6776

3.7. Daily average object size

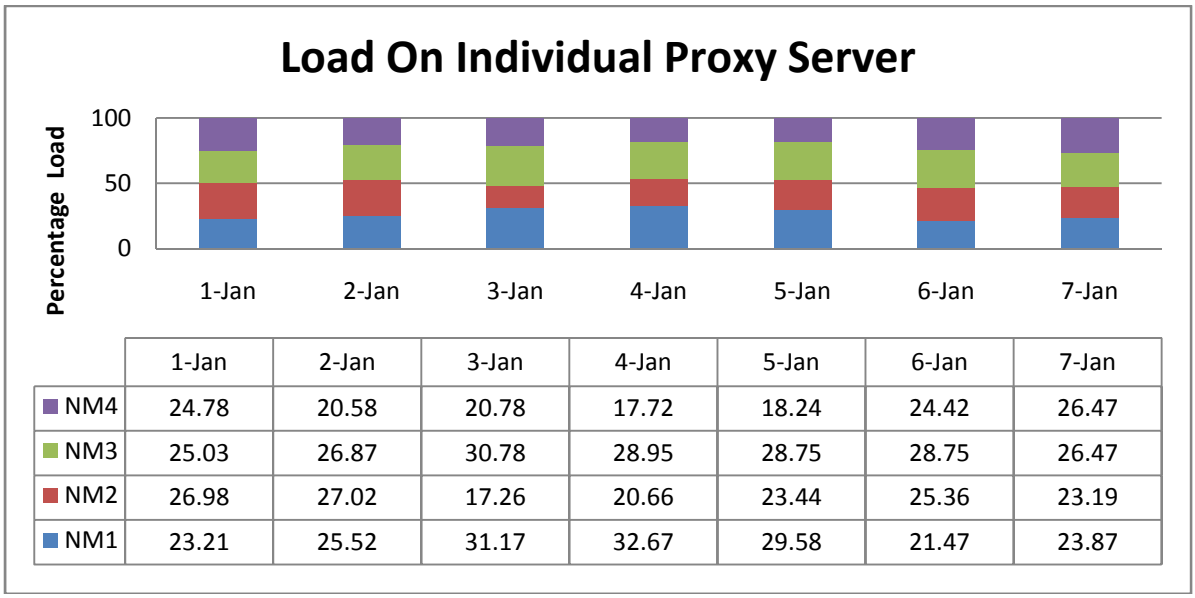
Below graph shows the average object size in KB. We find that, average object size is around 9 KB. X-axis is day and Y-axis is size in KB.



	1-Jan	2-Jan	3-Jan	4-Jan	5-Jan	6-Jan	7-Jan
Object Size	9.2856	8.4089	7.9367	8.87	9.4052	9.0189	8.7595

3.8. Load on each Proxy Servers

As we saw in previous section, IIT Bombay has 4 proxy servers. We find that load on each proxy server is equally divided with each server serving around 25 % of total request. Which implies that proxy server is using round robin scheduling for load balancing. On 5th JAN we find that proxy server 4 has served 18% of request, which might be because of persistent connection configuration in proxy server. Persistent connection says that the TCP connection from some user should select last used proxy server if it occurs within specified time.

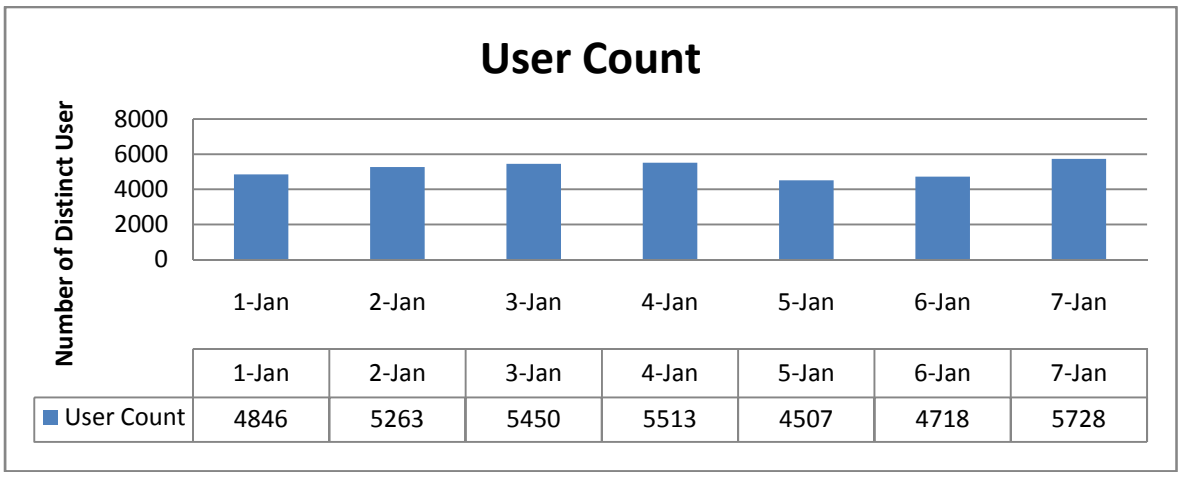


	1-Jan	2-Jan	3-Jan	4-Jan	5-Jan	6-Jan	7-Jan
NM4	24.78	20.58	20.78	17.72	18.24	24.42	26.47
NM3	25.03	26.87	30.78	28.95	28.75	28.75	26.47
NM2	26.98	27.02	17.26	20.66	23.44	25.36	23.19
NM1	23.21	25.52	31.17	32.67	29.58	21.47	23.87

X-axis is day and y-axis is percent of total request served by each proxy server.

3.9. Distinct user count

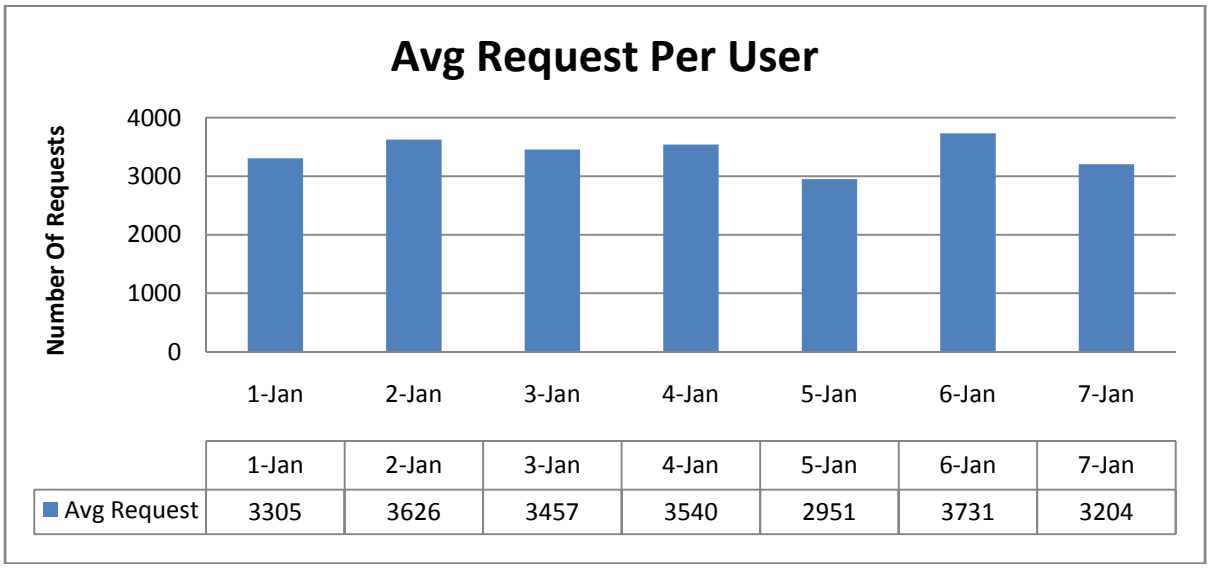
On a given day around 5100 distinct users sends requests to proxy server. In chapter 4 we will see the effect of usage pattern of large number users on cache hit rate. In below graph x- axis represent day and y-axis is distinct user count.



	1-Jan	2-Jan	3-Jan	4-Jan	5-Jan	6-Jan	7-Jan
User Count	4846	5263	5450	5513	4507	4718	5728

3.10. Number of requests per user on single day

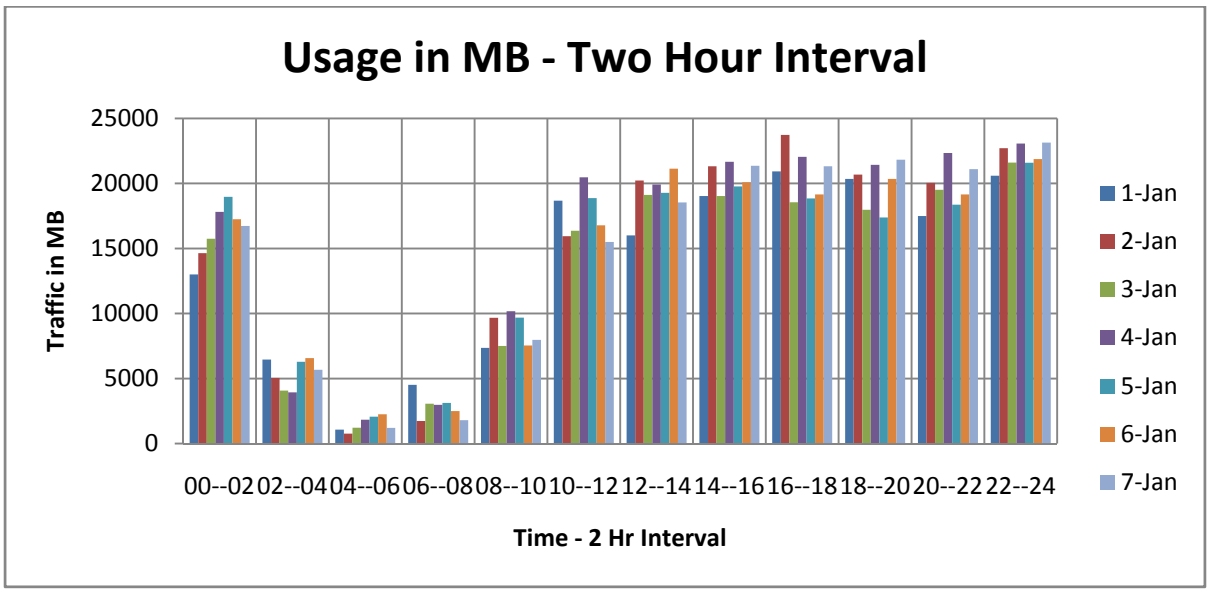
On a given day, a user sends around 3400 requests to proxy server. We will also look at how 407 requests are handled to reduce network bombardment in chapter 5. In below graph x-axis represent day and y-axis represent average number of request per user.



	1-Jan	2-Jan	3-Jan	4-Jan	5-Jan	6-Jan	7-Jan
Avg Request	3305	3626	3457	3540	2951	3731	3204

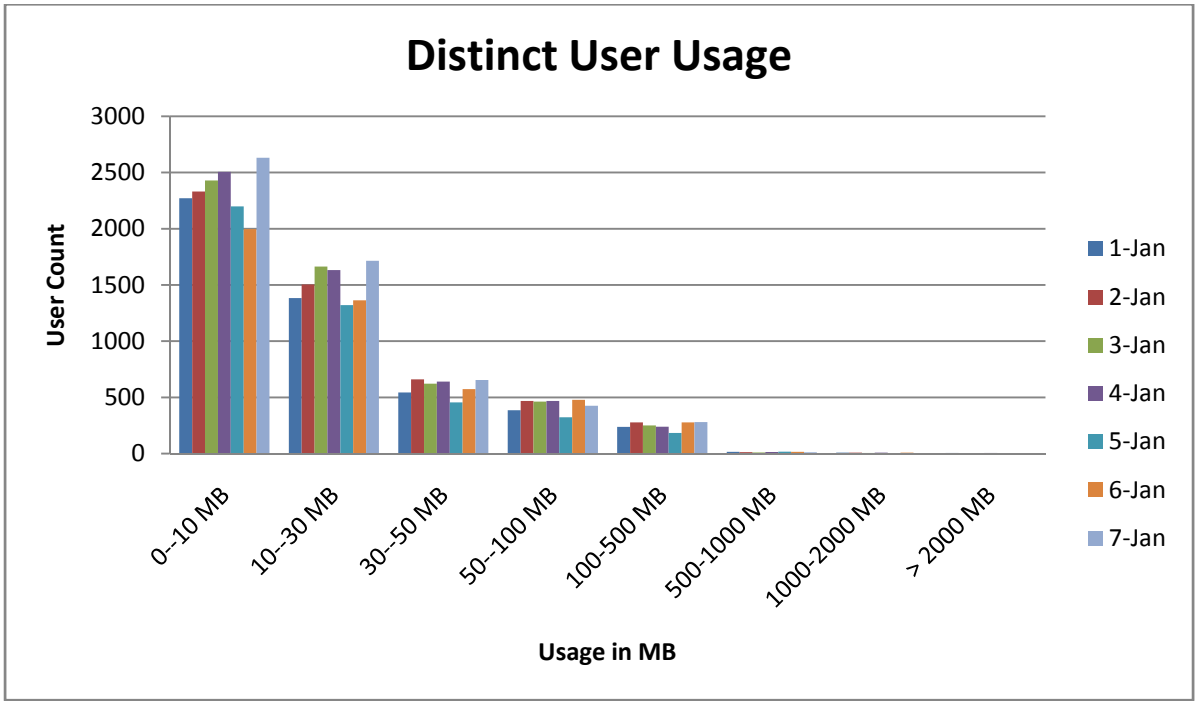
3.11. Usage in every 2 hour interval on a single day

In this experiment we try plot net usage in 2 hr interval for whole day. We find that, from 10:00 to 02:00 usage is @ 15 GB/2 hr. While from 02:00 to 10:00 usage is around 5 GB and it is lowest at @ 2 GB from 04:00 to 06:00. This reflect LAN ban period in hostel in night hours. Again we find that usage is high in hostel during day time. In below plot x-axis is time and y-axis is usage in MB.



3.12. Number of user having specific usage per day

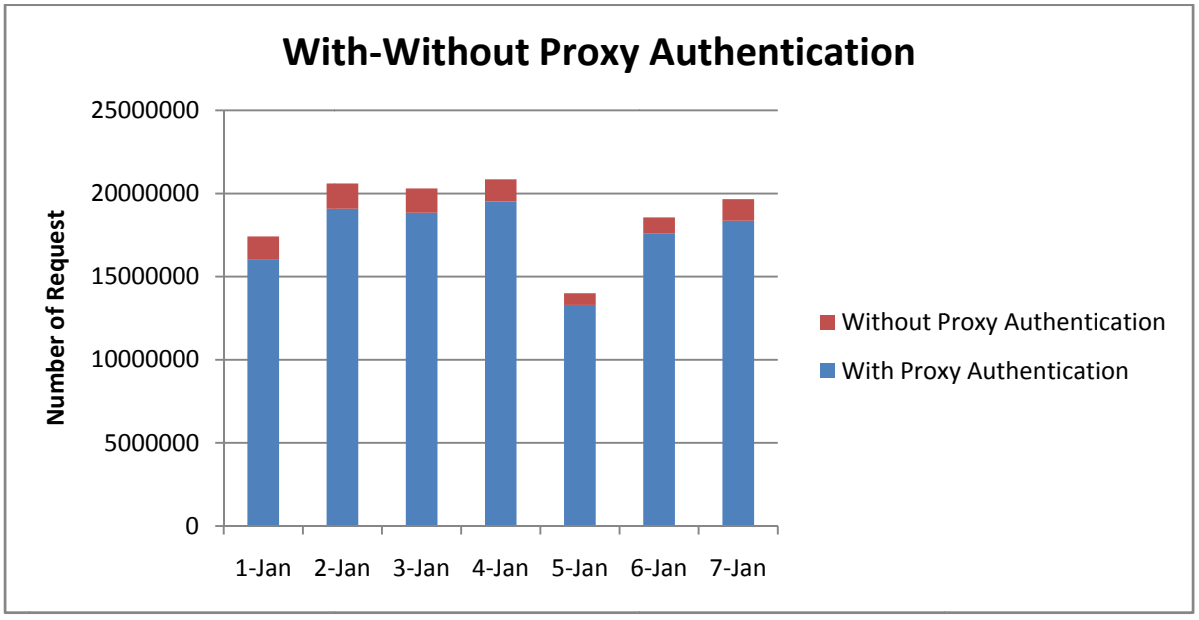
In this experiment we try to find the usage of user by their transfer size. We will plot number of user having transfer size given below. For example, plot says that @ 2500 user on 4th Jan has usage less than 10 MB. In general out of 5100 distinct user @ 2300 users are having usage < 10 MB on a given day. While @ 6 users have usage > 1 GB and @ 3 user has usage > 2 GB. We find that usage pattern follows 80-20 rule.



From above observation we find that there are very few users who account for high usage. On average per user usage is low. In above graph x-axis is usage in MB and y-axis is user count.

3.13. Requests With/Without Proxy Authentication

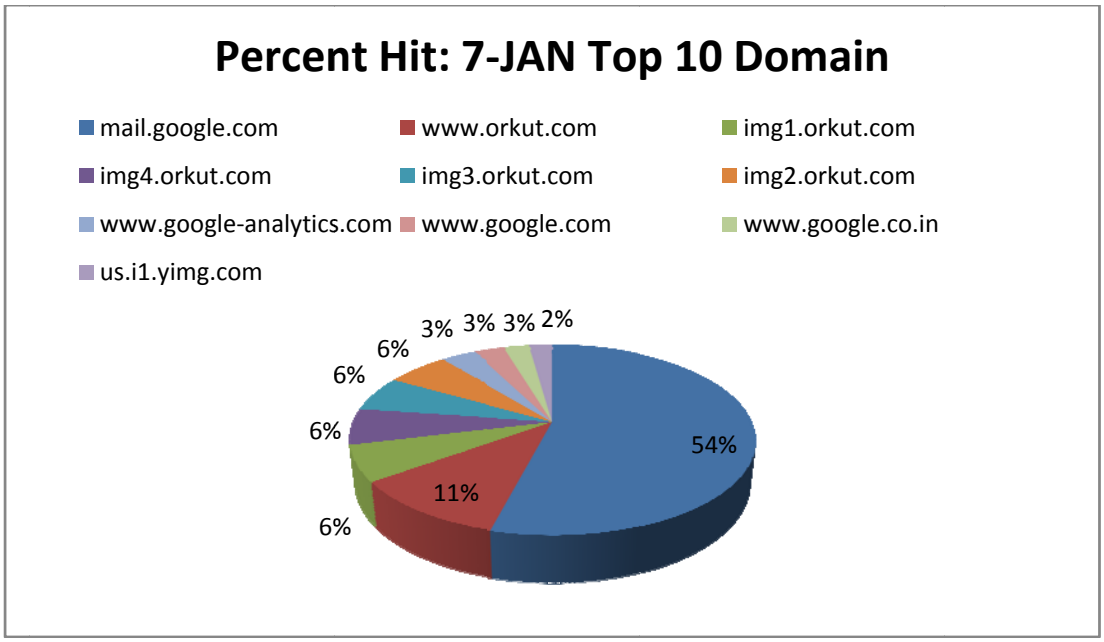
IIT Bombay uses LDAP as authentication for net access. If user send request without proper authentication, it will mark as 407. I.e. HTTP response for “require proxy authentication”. From proxy log we find that there are some users who send 407 requests in burst for specific time. This is mostly due to automatic antivirus update, open office update which are not configured properly in this experiment we try to find number of valid and invalid requests. In below plot, x-axis is day and y-axis is number or request with proxy and without proxy authentication.



We find that on average there are 8 - 9% of total requests which are without proxy authentication. This number in small but they occur in burst, which is overhead for proxy server. In section 5 we will look at prototype system to prevent this bombardment.

3.14. Top domain on a given day

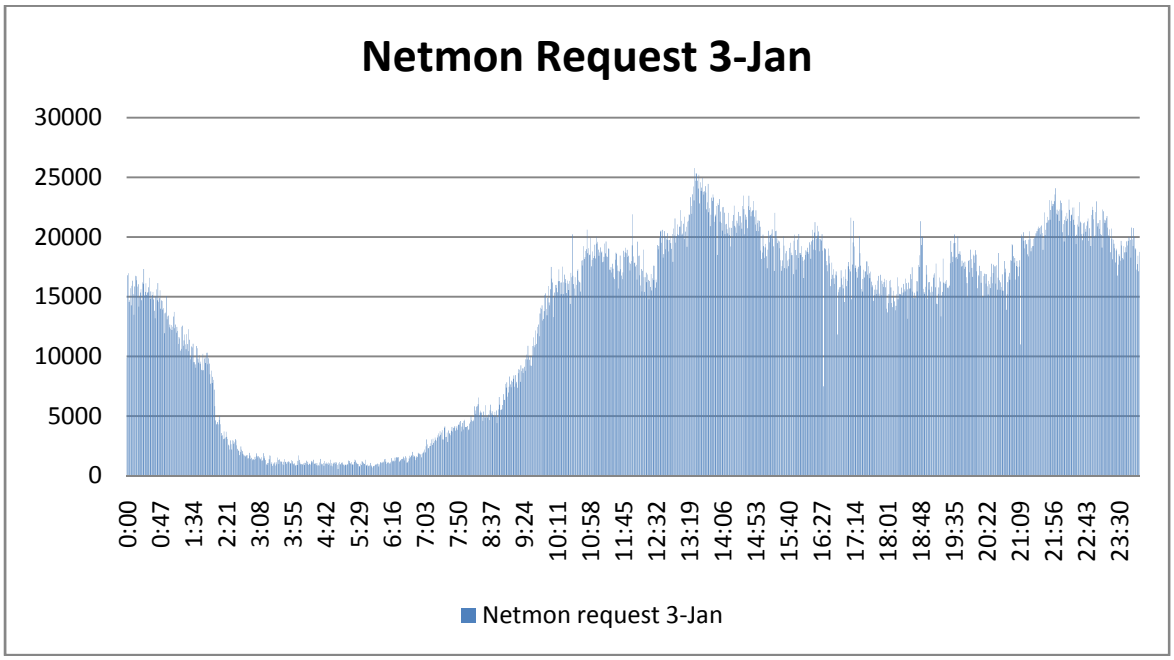
This experiment shows top domain accessed from IIT Bombay. On a given day google.com domain consumes 50% of total traffic followed by orkut.com and yimg.com. (It depends on specific day, but by observation usually “google.com” and “orkut.com” domain consumes bulk of the request)



We also found some unusual domain like iitbfreedom.com, which was in top list on 6th Jan.

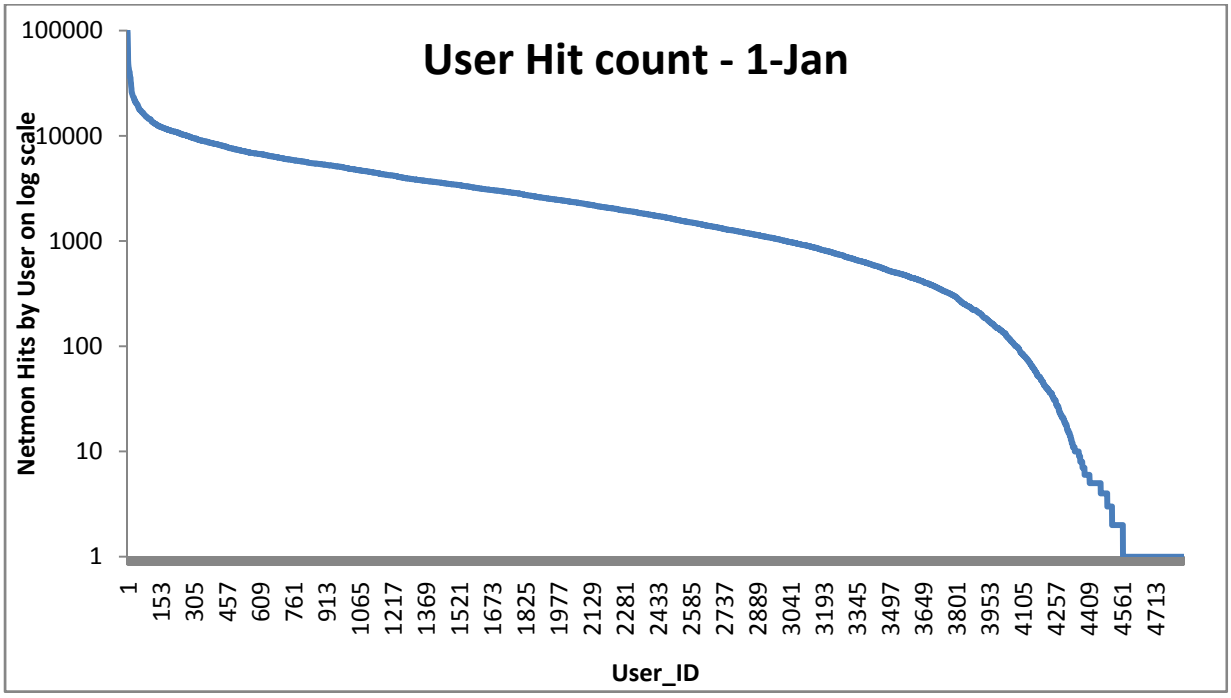
3.15. Requests in 1 min interval for 24 hours

Below plot shows number of request in 1 min interval. From 12.00 AM to 7 AM Hostel network is turned off in IIT Bombay which is evident from fig. on average, all proxy receives @ 20000 request/min. In plot, X-axis is time in min and Y-axis is number of requests received by proxy server.



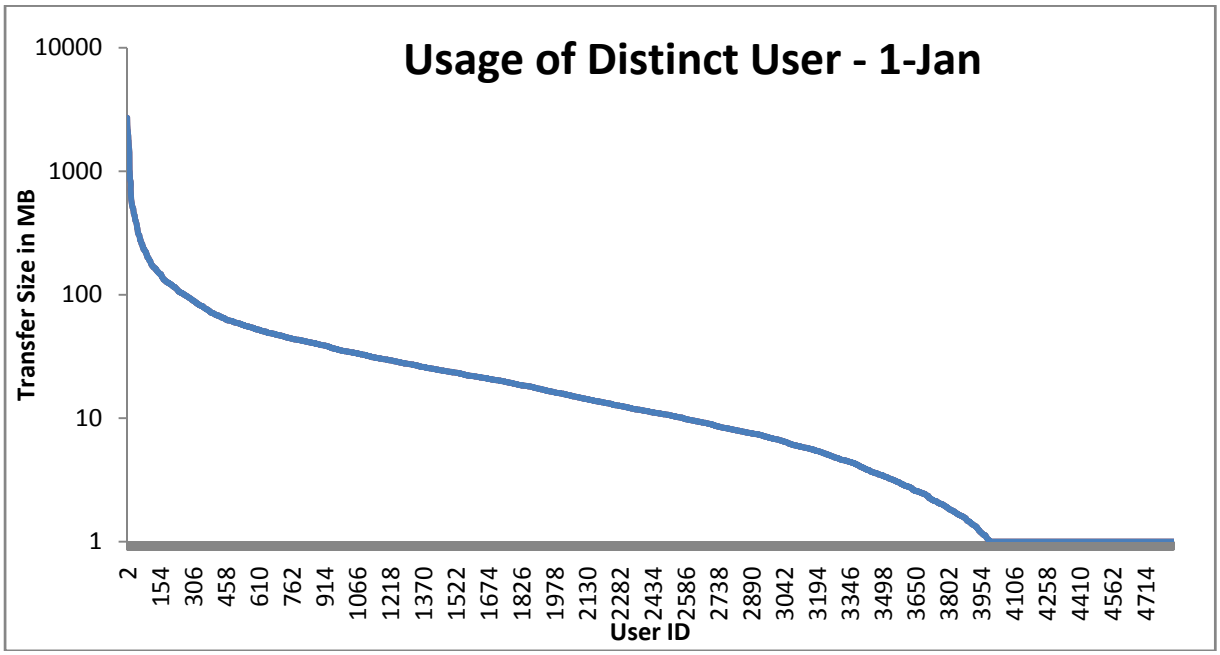
3.16. Number of proxy hit by user sorted by frequency

In this experiment we plot the request received by proxy server from user sorted by frequency. I.e. plot shows there are few users who are responsible for higher request rate. The plot is tail heavy and obeys 80-20 rule. X- axis is hashed user ID and y-axis is frequency of request on log scale.



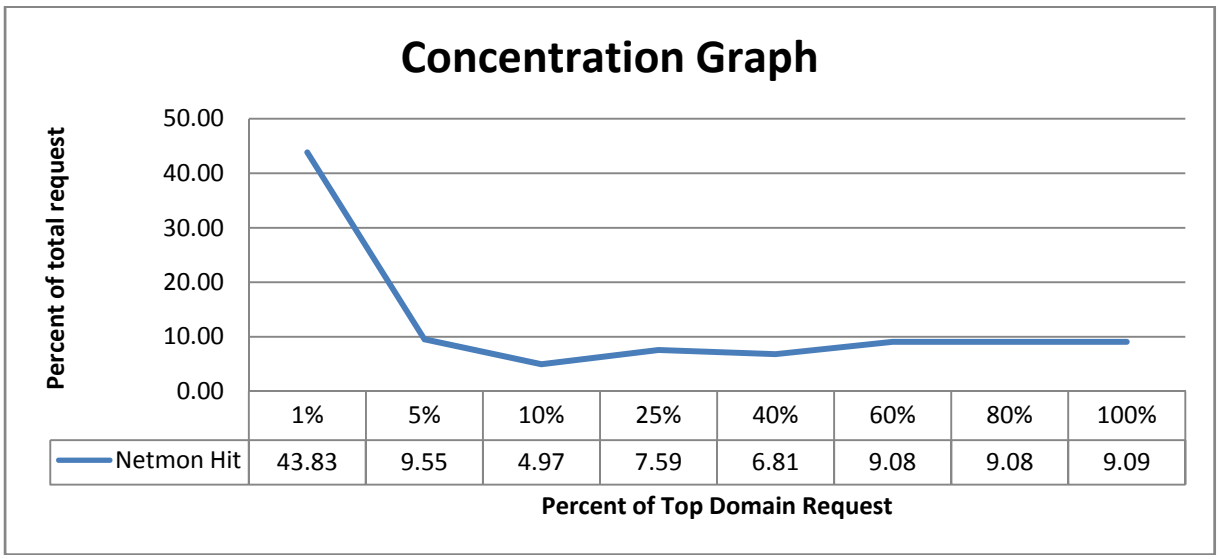
3.17. Usage of user sorted by usage

Below graph plots the usage of user sorted by transfer size in descending order. This show, very few users are having large usage. Graph is tail heavy which adhere to 80-20 rule. This is evident from result from 3.16. Here x-axis is user ID and y-axis is usage in MB.



3.18. Concentration Graph

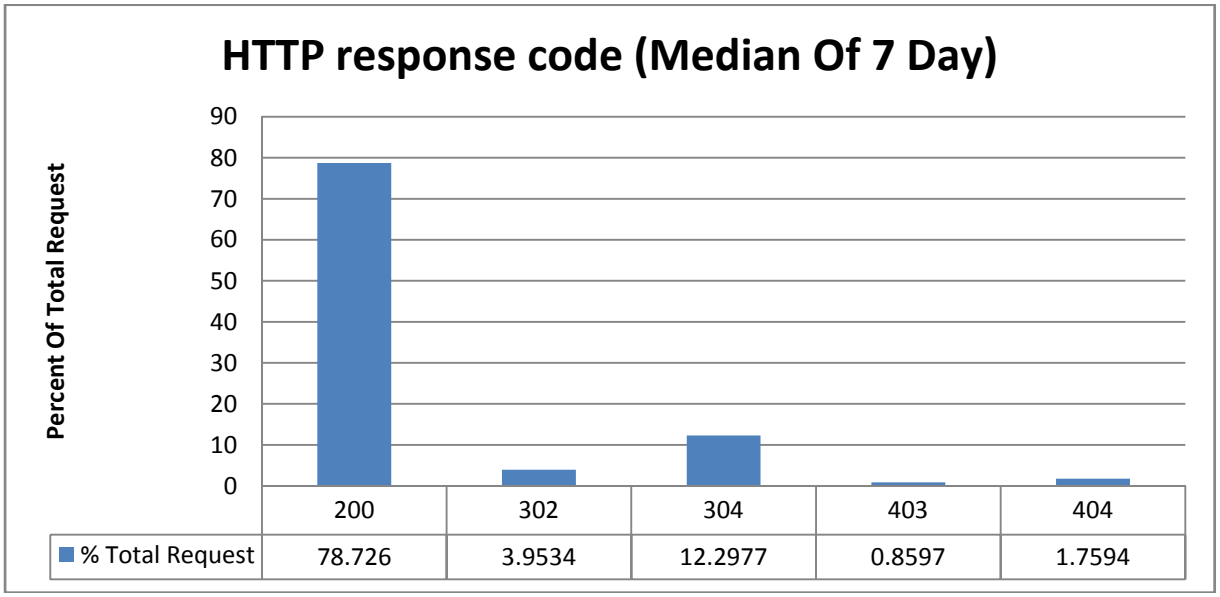
Below graph plots the concentration graph depending on number of request made by top accessed domain. For example, it says that 1% of top domain is responsible for 43% of total request. We saw that google.com and orkut.com domain is top accessed domain. From this result we say that google.com and orkut.com and some other domain are responsible for 43 % of total request. Rest of the domain is on average responsible for 10% of total request as show in below plot.



3.19. Distribution by HTTP response code

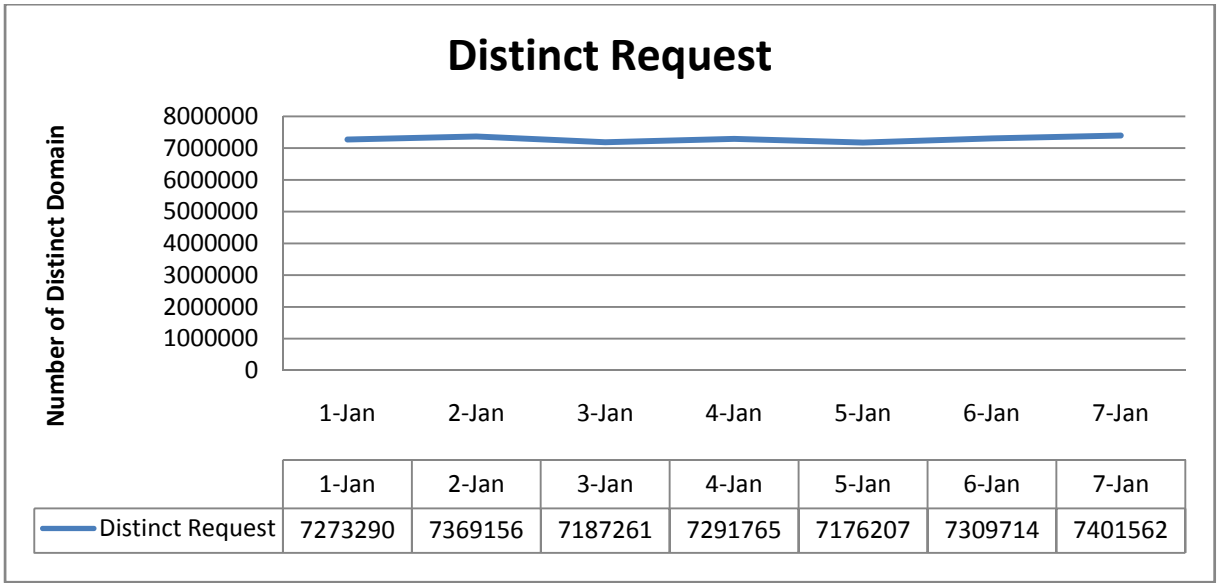
From below plot we find that around 78% of total request are valid request. Also from access log we find that in most cases 304 and 302 are generated from *orkut.com* domain. below is the definition of response code

HTTP Response Code	What it means!
200	OK - Successful HTTP request
302	Found - Gives location of redirection
304	Not-Modified since last request (Used during caching)
403	Forbidden - Not allowed to access (Authentication required)
404	Not-Found - Document does not exists
407	Proxy Authentication required



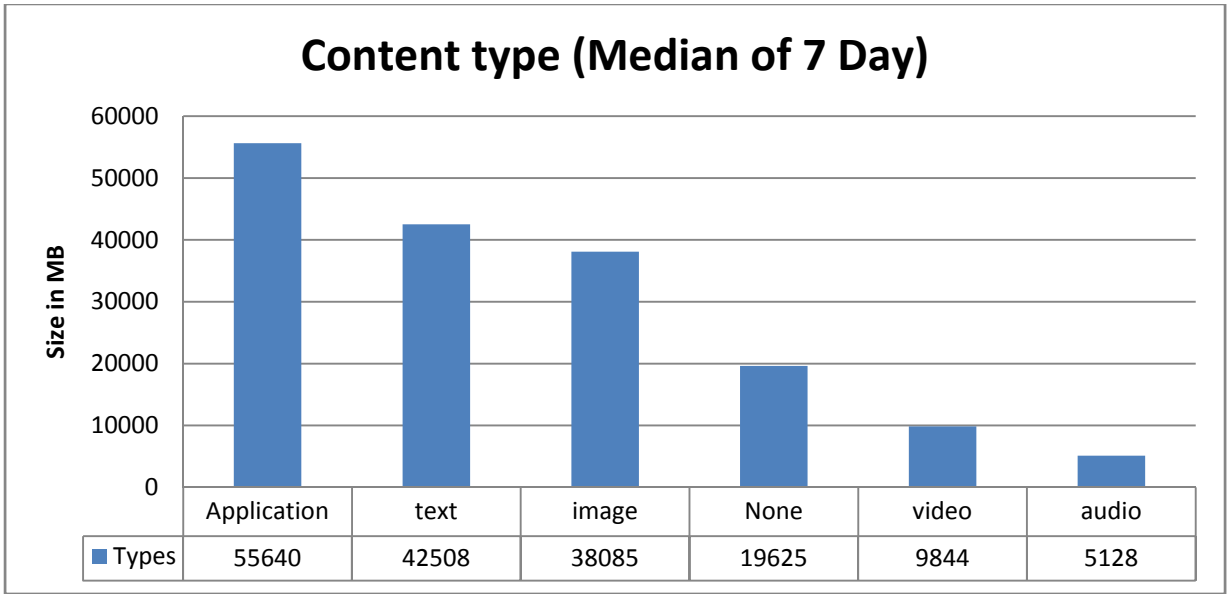
3.20. Distinct Request type on a given day

From below plot, we find that on average 44% of total request on a given day are unique. Hit rate and distinct domain has direct correspondence. Due to large number of distinct domain, hit rate might be low as user are requesting different object. In plot x-axis is day and y-axis is number of distinct domain.



3.21. Distribution by content Type

From below graph we say that Application tops the usage chart followed by text and image content. Application consists of x-javascript, x-msn-messenger, octet-stream, x-shockwave-flash, xml, x-fcs, pdf, zip etc.



3.22. Summary

In this chapter we have looked at various invariants found in proxy access log. Below is the snippet of statistics

- 40% and 30% of total request is for Image and HTML respectively
- Cache hit is @ 21% and cache miss is @ 63%
- 40% of total object requested has size between 1000 bytes to 10000 bytes
- Average object size is around 9 KB.
- Proxy server is using round robin scheduling for load balancing.
- We find that usage pattern follows 80-20 rule (Power Law).
- Very few user accounts for > 2000 hit on a given day
- 1% of top domain is responsible for 43% of total request.
- 78% of total request are valid request resulting in http response code 200
- On average 44% of total request on a given day are unique.

Now in next chapter we will look at cache replacement protocol behavior with varying cache size and also considering unified cache for all proxy servers.

Chapter 4: Cache Behavior

4.1. Introduction

Cache is used to reduce response time to the client and the core component which ensures it is its replacement protocol and its size. The cache replacement protocol is used to make room for new unseen object in cache when it is full. In this chapter we will look at how the performance of cache replacement protocol directly affects the cache hit rate[9] and a way to enhance the cache performance. We will also look at effect of varying cache size on hit rate.

4.2. IIT Bombay Setup

IIT Bombay uses open source squid software as proxy server. From the documentation we found that it uses variant of LRU as cache replacement protocol. The setup of proxy server in IIT Bombay is such that all four proxy servers have its own isolated cache. In such case if user request object and if it is not on requesting servers cache it will results in cache miss and will try to fetch if from original source even if it is residing on some other proxy server. This behavior might reduce the cache hit rate which could be possible to achieve if we use unified cache for all server.

This observation leads us to study the effect of unified cache on hit rate with varying cache size. In order to study the cache behavior I have written small JAVA simulator which has tunable parameter such as cache size and number of proxy servers to simulate cache replacement protocol.

In next section we will look detailed result obtained from simulator. Before we move forward, below is the basic setup that has been done for simulator.

1. Access log file is used for processing
2. User ID has been hashed for user privacy
3. Access log file has been truncated and contain simulator relevant information. Size reduction is 4:1. i.e. each truncated file is about 1 GB compared to 4 GB of original access log file
4. Each proxy server has identifier in each line of access log i.e. ‘nm1’ for first proxy server and so on
5. Specific file format has been flagged as cacheable object and we can change it as and when need arise.
6. Maximum allowable object size for caching can be set prior to simulation.

Appendix A.2 contains the cache simulator code with some explanation. Now we will look at simulator statistics.

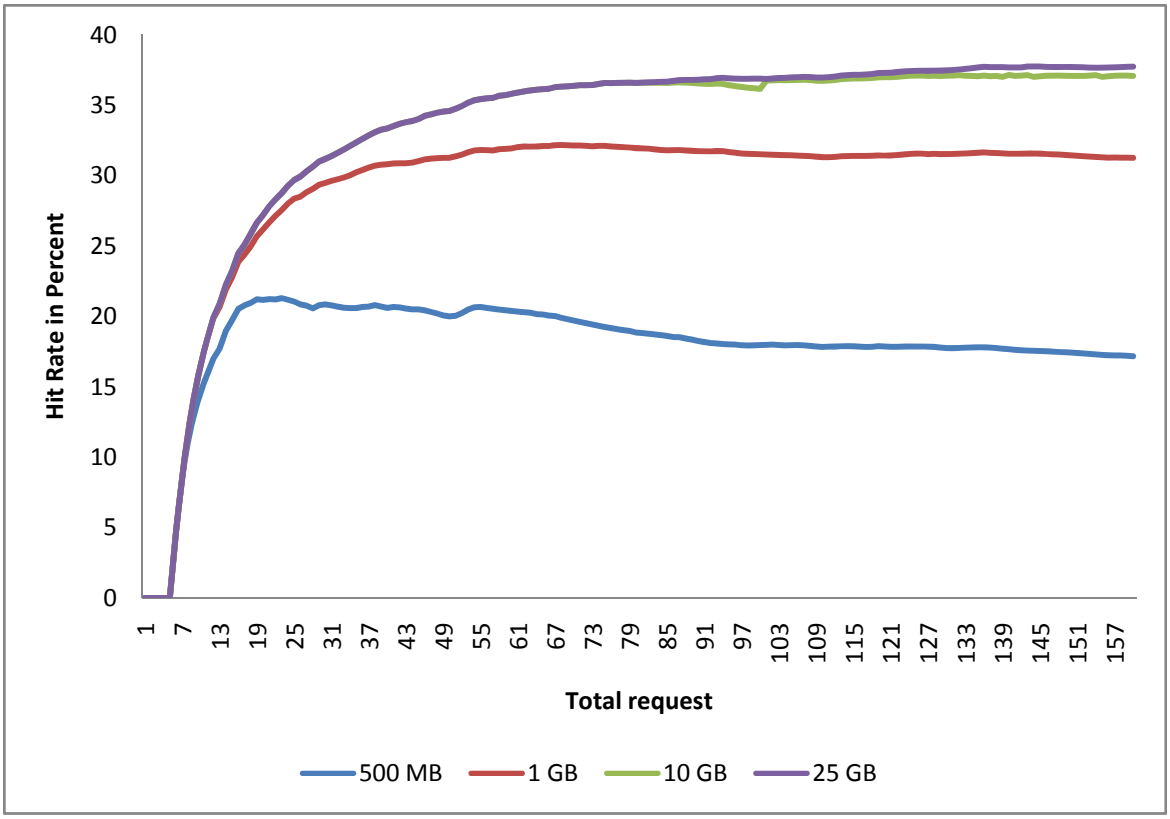
[9] Hit rate is defined as the ratio of cache hit i.e. object found in cache by total number of request.

4.3. Experiment and Results

In this section we will simulate single server and merge server scenario. In single server we will consider single proxy server trace[10] and try to simulate cache hit rate with varying cache size. While in merger server we will construct unified cache for complete trace.

In simulation, initially 0.1 million lines are used for training cache. Once there is sufficient data in cache we start counting cache hit.

4.3.1. Single Server LRU



Simulator parameters

Learning cut off	1 Lac Line
Proxy Server	nm1
Cache Size	500MB,1GB,10GB and 25GB
Maximum size of cacheable object	200 MB

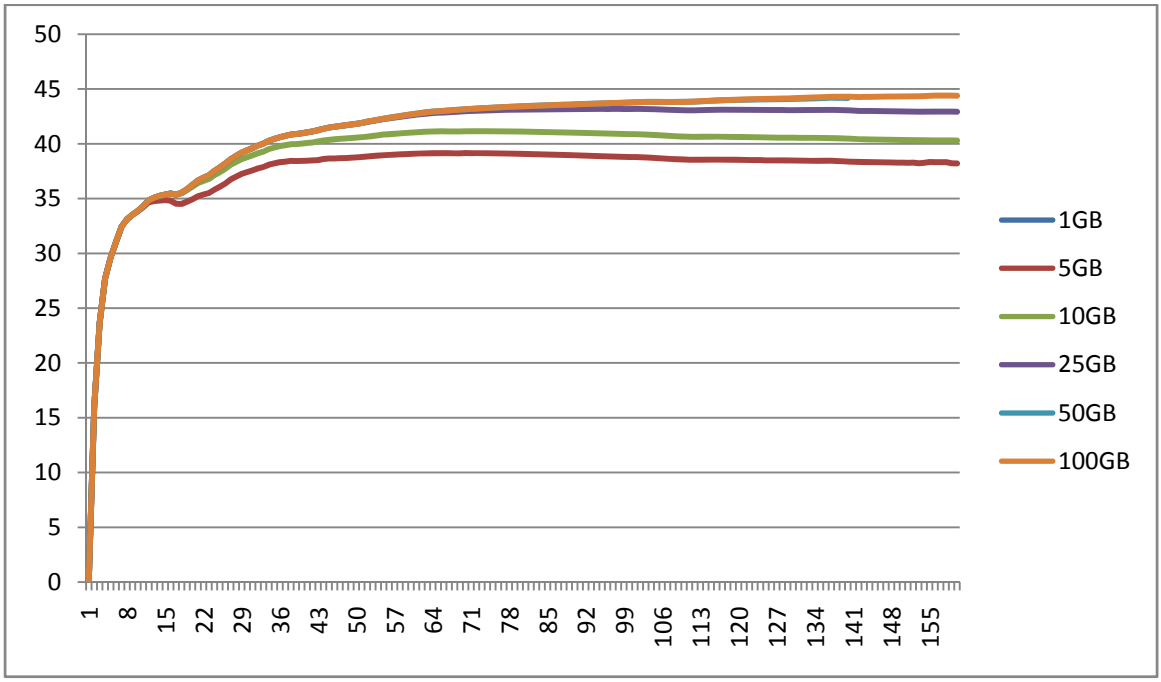
With LRU as cache replacement protocol and with above simulation parameter we get cache hit rate as shown in above figure. Due to distinct request type and user access pattern hit rate

[10] We have four proxy server nm1 to nm4- we will consider nm1 trace from access log file

is around 38 percent. This example is for isolated single server scenario. Next we will look at unified scenario.

4.3.2. Merge Server LRU

In merge scenario, the maximum hit rate is around 45 percent with cache size of 50 GB. This is due to fact that usage pattern across individual server does have share resources and due to which cache miss in isolated scenario can be a cache hit merger scenario. Graph also shows cache hit with varying size. As cache size increases hit ratio also increases but after some time it levels off. One reason could be due to usage pattern. As we saw in section 3.20 which says 45 % of request are distinct, it might be case that there are no more cacheable objects after certain limit.



Simulator parameters

Learning cut off	1 Lac Line
Proxy Server	All
Cache Size	1GB,5GB,10GB,25GB,50GB and 100GB
Maximum size of cacheable object	200 MB

4.4. Summary

In this chapter we looked at isolated and unified cache scenario and saw effect of unified cache on hit rate. This guides us to use unified cache to improve cache performance thereby reducing bandwidth usage.

Chapter 5: Proxy Bombardment

5.1. What is Bombardment?

On an average day proxy server receive various types of request which can be classified depending on HTTP response code. Following are the common response code found in IIT Bombay access log,

HTTP Response Code	What it means!
200	OK - Successful HTTP request
302	Found - Gives location of redirection
304	Not-Modified since last request (Used during caching)
403	Forbidden - Not allowed to access (Authentication required)
404	Not-Found - Document does not exists
407	Proxy Authentication required

From above response code, 407 code constituent proxy bombardment which is explained next.

IIT Bombay use LDAP[11] authentication for various intranet services such as user forum, course management software – moodle and for internet access. User uses proxy server with LDAP ID as authentication parameter for access internet. All will work well if user provides proper credential. For example, when user open web browser, it asks for user name and password. This behavior is due to proper HTTP response handling in web browser. As soon as browser send first request for web page without authentication, proxy server will send 407. Now on other side browser will handle this by asking user credential. Once user enter valid id and password connection go thorough and he can use internet.

There are some softwares which do not handle 407 as web browser do. In fact such software bombard proxy server with high request rate. On studying access log we found Adobe Auto updates, ESET Antivirus, Windows update were among the bombarding application. There is sharp variation in bombardment rate and normal usage pattern. More precisely, bombarding user has around 140 req/sec and this behavior continues till such application is active where as this not the case with normal user.

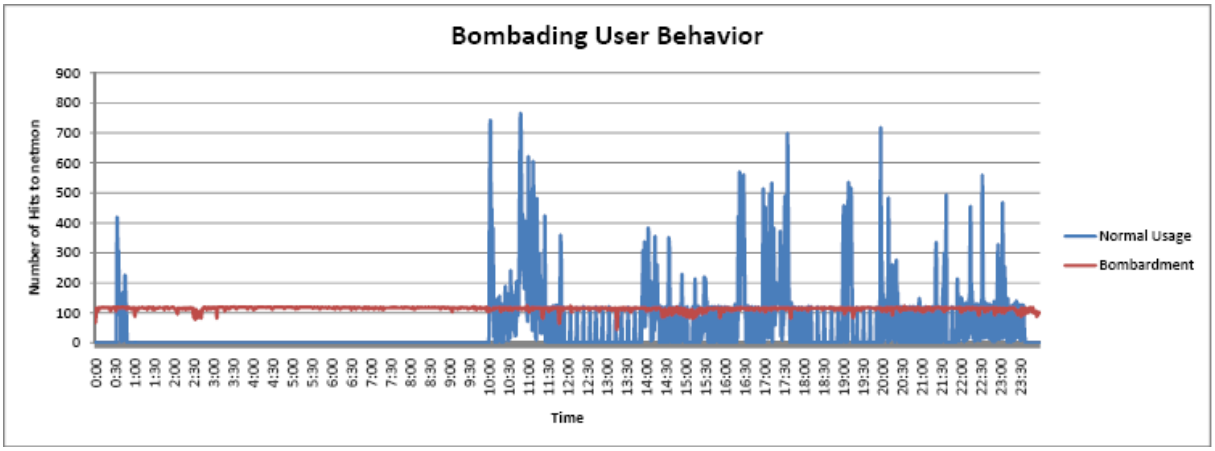
Clearly, proxy server is doing nothing except denying such request and wasting CPU in processing such request.

Below is the statistics of 407 log file.

Log duration	1-Jan'08 – 7 Jan'08
Log Size	2.1 GB
# of line per day	1.4 million

[11] LDAP is a light weight directory access protocol for user management http://www.openldap.org/

Following figure explains the normal usage and the one which is bombarding proxy server.



In next section we will look at prototype system to prevent such request from reaching IIT Bombay's proxy server.

5.2. Detecting Bombardment

We start with method to detect bombardment in IIT Bombay's network scenario. Once bombardment is detected alerts are generated and email is send to the administrator giving IP of the bombarding machine.

5.3.1. Algorithm

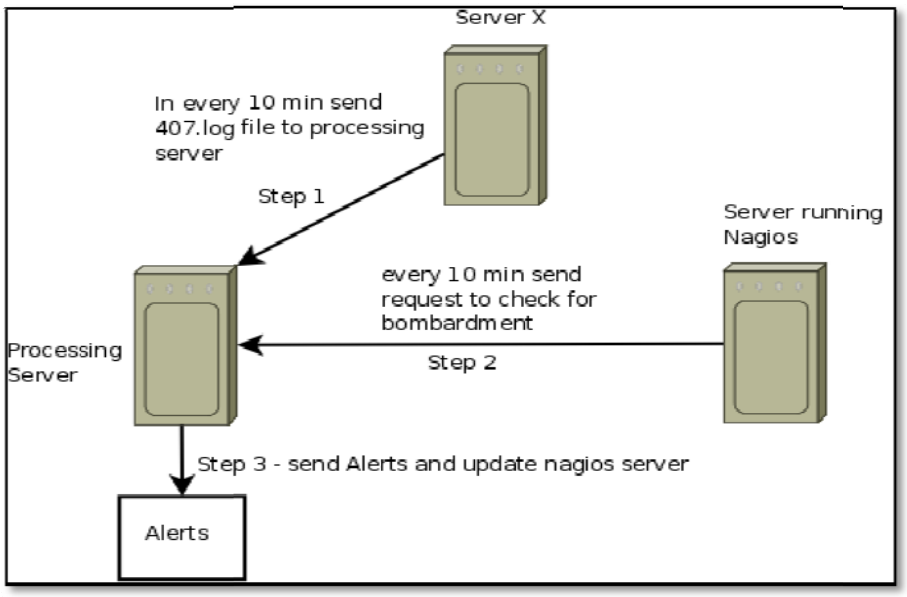
Input for the system is raw file containing one line for each 407 request. The structure of file is same as that of access.log file discussed in chapter 2. Below outline the algorithm for bombardment detection.

1. Periodically check 407 log file
2. Find unique IP which are bombarding proxy server in last x minute (let say 10 min)
3. If any IP is found send then alert to Administrator
4. Block such unique IP-MAC pair with guarantee that no further request from such pair ever reach proxy server.

5.3.2. Implementation

First step is to get 407 log file for processing. In IIT Bombay network, there is single 407 log file for all 4 proxy server as discussed in chapter 2. We want heavy processing to be done on different server than logging server so Cron job is written on log server to send 407 log file to processing server every 10 min. It also sends alerts if necessary.

Below is the representation of prototype system, which consists of 3 machine each responsible for specific purpose and they work together for bombardment detection. Each machine has specific task to do. One is sending log file for processing, other is for processing that file upon receiving trigger from third machine.



Steps:

1. Logging server log 407 request (Server X)
2. It sends 407 log file to processing server every 10 min. (Using Cron job)
3. A server running Nagios[12] sends request to processing server for bombardment checking.
    a. Here the processing server runs a program which listen for the request from Nagios Server. Upon receiving the request it process the log file and find bombarding IP.
    b. Threshold for bombardment is chosen to be 1000 request in last 10 min. This can vary depending on the request pattern.
4. Once the bombardment is detected, processing server send mail to administrator with IP of the bombarding machine.
5. Following is the message content

```
NETMON Bombardment in Last 10 minutes of span
IP ADDRESS   - Netmon Hits
10.9.1.40    - 1834
10.107.170.41 - 1580
```

Code is written is Perl and Python and it uses socket programming. For more information refer Appendix A.3. Once the IP has been found, ACL is updates in specific router to block future request from bombarding IP.

[12] Nagios is a network monitoring software which can be configured to monitor custom service www.**nagios**.org/

5.3. Summary

In this chapter we looked at prototype method to prevent proxy bombardment in IIT Bombay network. It does depend on actual network topology and hence might not work on other network. The threshold is set to 1000 request per minute and it come from the observation. One need to fine tune it depending on nature of bombardment.

Chapter 6 - Conclusion and Future Work

We have seen various invariants observed in IIT Bombay web traffic. Most interesting of all are usage difference between Hostel and Academic segment and 407 bombardments. We looked at method to prevent 407 requests in chapter 5, which identifies the bombarding IP. In current prototype, blocking of such IP is done manually. We can extend it to block such IP automatically.

We find that there is sticking difference between hostel and academic segment and IIT Bombay internet lines are experiencing chocking due to excessive net usage from hostel segment. To counter this problem we propose to use different proxy server for hostel and academic segment. We can cap the bandwidth of hostel proxy to give sufficient speed to academic segment.

IIT Bombay has blocked the access to majority of video sharing and file sharing site, which they found were clogging the network. We propose to allocate separate proxy for such access with download limit on each user. This will give sufficient bandwidth to genuine user in academic section.

In all solution to improved net experience is to increase the bandwidth. IIT Bombay has 32 Mbps of ISP line which very low compared to world class institute. In course of experiment we found some site were not globally known but had very high access rate from IIT Bombay. One should look in to such domain.

IIT Bombay has MRTG for all ISP line and in case of one ISP, which is supposed to provide 16 Mbps plot never gone above 12 Mbps. In contrast IIT Bombay has usage more than 32 Mbps at any given moment. One needs to look at it and be vigilant about it.

From cache experiment we found, unified cache gives high hit rate compared to isolated cache, which is current configuration at IIT Bombay. We can use cluster file system such as GFS and implement unified cache. But this needs some careful configuration such as simultaneous access to file system and proxy server should handle this.

Implementing above finding will definitely improve overall net experience in currently clogged network in IIT Bombay.

Bibliography

[1] *Web server workload characterization: the search for invariants.* Martin F. Arlitt, Carey L. Williamson.1996

[2] *The measured access characteristics of world-wide-web client proxy caches***.** Bradley M. Duska, David Marwood, Michael J. Feeley. 1997

[3] *YouTube Traffic Characterization: A View From the Edge*. Phillipa Gill, Martin Arlittz, Zongpeng Li, Anirban Mahanti. 2007

APPENDIX A

A.1. Code for data processing

```
#!/usr/bin/perl -w
use strict;
use IO::Socket;
use File::ReadBackwards;
use Digest::MD5 qw(md5_hex);
my ($hash) = md5_hex("TEST");
open(STDERR, ">$ARGV[2]") or die "Could not open $ARGV[2]: $!\n";
my $nlines = 0;
    my $bw;
    if (! ($bw = File::ReadBackwards->new($ARGV[0]))) {
        print STDERR "Couldn't open $ARGV[0]: $!\n";
        return -1;
    }
    open(OUTFILE,">$ARGV[1]");
    my $log_line;
    my $winex = 0;
    my $ltime = 0;
    my $key;my $value;my $key_t;
   while($log_line = $bw->readline) {
  my
($var1,$var2,$var3,$var4,$var5,$var6,$var7,$var8,$var9,$var10,$var11,$var12,$var13,$var1
4,$var15,$var16)=split(" ",$log_line);
 #Handling Month
  if ($var2=="Jan") {$var2="01";}
  if ($var2=="Feb") {$var2="02";}
  if ($var2=="Mar") {$var2="03";}
  if ($var2=="Apr") {$var2="04";}
  if ($var2=="May") {$var2="05";}
  if ($var2=="Jun") {$var2="06";}
  if ($var2=="Jul") {$var2="07";}
  if ($var2=="Aug") {$var2="08";}
  if ($var2=="Sep") {$var2="09";}
  if ($var2=="Oct") {$var2="10";}
  if ($var2=="Nov") {$var2="11";}
  if ($var2=="Dec") {$var2="12";}
  if ($var3=="1") {$var3="01";}
  if ($var3=="2") {$var3="02";}
  if ($var3=="3") {$var3="03";}
  if ($var3=="4") {$var3="04";}
```

```
  if ($var3=="5") {$var3="05";}
  if ($var3=="6") {$var3="06";}
  if ($var3=="7") {$var3="07";}
  if ($var3=="8") {$var3="08";}
  if ($var3=="9"){$var3="09";}
  my ($access_date)=$var1.'-'.$var2.'-'.$var3;
  my ($hour,$min)=split(":",$var4);
  my ($access_time)=$hour.':'.$min;
  my ($netmon_server)=$var5;
  my ($process_time_ms)=$var8;
  my ($source_ip)=$var9;
  my ($tcp_status,$tcp_status_code)=split("/",$var10);
  my ($object_size)=$var11;
  my ($request_type)=$var12;
  my ($domain) = md5_hex($var13);
  my ($user_id)= md5_hex($var14);
  my ($server_fetch_type,$server_ip)=split("/",$var15);
  my ($object_type,$object_sub_type)=split("/",$var16);
  print OUTFILE
"$access_date\t$access_time\t$netmon_server\t$process_time_ms\t$source_ip\t$tcp_status\t$
tcp_status_code\t$object_size\t$request_type\t$domain\t$user_id\t$server_fetch_type\t$serve
r_ip\t$object_type\t$object_sub_type\n";
}
close(OUTFILE);
```

A.2. Code for cache simulator

Simulator.java

```
import java.io.*;
import java.util.*;

public class simulator{
        public double cacheSize=0;
        public long currentCacheSize=0;
        public double totalObjectSize=0;
        public double hit=0;
        public Hashtable cacheObjectsize = new Hashtable();
        public Hashtable cacheHash = new Hashtable();
        public int cutoff=100000;
        public int count=0;
        public int totalCount=0;
```

```
	public void  sortAndremove(reader obj)
	{
		ArrayList myArrayList=new ArrayList(this.cacheHash.entrySet());

//		Sort the values based on values first and then keys.
	Collections.sort(myArrayList, new MyComparator());

		Iterator itr=myArrayList.iterator();
		String key="";
		//String value;

		while(this.currentCacheSize+obj.object_size >this.cacheSize)
		{
			Map.Entry e=(Map.Entry)itr.next();
			key = (String)e.getKey();
			this.cacheHash.remove(key);
	 this.currentCacheSize-=Long.parseLong(this.cacheObjectsize.get(key).toString());
		}
		itr.remove();
	 }

	public void manageHash(reader obj)
	{
		//sort on time stemp
		this.sortAndremove(obj);
	}

	public void  process(reader obj){
		if ((this.cacheHash.get(obj.domain)==null) &&
(this.currentCacheSize+obj.object_size<=this.cacheSize))
		{
			this.cacheHash.put(obj.domain, obj.access_time);
			this.currentCacheSize+=obj.object_size;
		}
		else
			if ((this.cacheHash.get(obj.domain)==null) &&
(this.currentCacheSize+obj.object_size>this.cacheSize))
			{
	 this.manageHash(obj);  // make space for new object and remove object based on LRU

				this.cacheHash.put(obj.domain, obj.access_time);
				this.currentCacheSize+=obj.object_size;
```

```
            }
            else
                  if (this.cacheHash.get(obj.domain)!=null)
                  {
                        if (this.count>=this.cutoff)
                              this.hit+=1;
                        //update time-stamp in Hash
                        this.cacheHash.remove(obj.domain);
                        this.cacheHash.put(obj.domain,obj.access_time);
                  }
}

public static void main(String[] argv){
int linecount=0;
simulator nm4=new simulator();
nm4.cacheSize=100000000000L;  //1 MBytes
      try
      {
            FileReader kl = new FileReader("/home/data.txt");
            BufferedReader mk = new BufferedReader(kl);
            String sword;
            int i=1;
            while ((sword = mk.readLine()) != null)
            {

                  linecount++;
                  if(linecount==100000)
                  {
               System.out.println(i+"\t"+(nm4.hit*100)/nm4.totalCount);i++;linecount=1;
                  }
                reader test = new reader(sword);

  //    if (test.server.equals("nm4"))    // ** for single
                  nm4.totalCount++;

       if (test.cachable()==true)
                  {// send test object for processing

     nm4.count++;
     nm4.cacheObjectsize.put(test.domain,Long.toString(test.object_size));
     nm4.process(test);
      }
```

```
                    test=null;
                    //test.display();
                }
                mk.close();
System.out.println("Hit in cache :" + nm4.hit);//+"  "+nm2.hit+"  "+nm3.hit+" "+nm4.hit);
System.out.println("Total nm4 line : "+ nm4.totalCount);//+"  "+nm2.count+"
"+nm3.count+"  "+nm4.count);
System.out.println("Total cacheable object : " + nm4.count);
            }
 catch (IOException ex){}
 }

static class MyComparator implements Comparator{
public int compare(Object obj1, Object obj2){

                int result=0;
                Map.Entry e1 = (Map.Entry)obj1 ;
                Map.Entry e2 = (Map.Entry)obj2 ;
                //Sort based on values.
                String value1 = e1.getValue().toString();
                String value2 = e2.getValue().toString();
                result=value1.compareTo(value2);
                return result;
            }
      }
}
```

reader.java - returns true if object is cacheable

```
public class reader{
                public String access_time;
                public String domain;
                public String netmon_server;
                public String request_type;
                public long  object_size;
                public String object_type;
                public String object_sub_type;
                public boolean consider=false;
                static int count=0;
                public String server;
                public reader(String str){
```

```
            String[] data = str.split("\\s+");
            server=data[0];
            //System.out.println(data[1]+" : "+data.length);
        if (data.length>6)
            {
        if
((!data[3].equals("CONNECT"))&&(data.length>6)&&(!data[1].equals("message")))

            {
                consider=true;
                count++;
                netmon_server=data[0];
                access_time=data[1];
                object_size=Long.parseLong(data[2]);
                request_type=data[3];
                domain=data[4];
                object_type=data[5];
                object_sub_type=data[6];
            }} }

        public void display()
        {
            System.out.println(domain+" "+object_sub_type+" "+object_type);
        }
        /**
         * Cheks whether given line is chachable or not
         * @return true if line is chachable else return false
         */
        public boolean cachable(){
            if ((this.consider==true) && ((this.object_type.equals("application")) ||
(this.object_type.equals("audio")) ||
                    (this.object_type.equals("video")) ||
(this.object_type.equals("flv")) ||
                    (this.object_type.equals("text")) ||
(this.object_type.equals("image"))||
                    (this.object_type.equals("binary")))  && (!(this.object_size >=
200000000)))
                return true;
            else
                return false;
        }
```

A.3. Code for bombardment detection

Server.pl : for processing log file and sending alerts

```
#!/usr/bin/perl -w
use strict;
use IO::Socket;
use File::ReadBackwards;

my $conf = '/home/logproc/server.cf';
my ($port, $logfile, $window, $ok, $warning, $errorlog);
my ($sok, $swarning, $scritical);
my %ip_Hash=();

open(FILE, $conf) or die "ERROR: Couldn't read $conf: $!";

while(<FILE>) {
   chomp;
   next if (/^\s*#/);
   next if (/^\s*$/);
   s/^\s+//g;
   s/\s+$//g;
   ($port) = $_ =~ /^PORT:\s+(\d+)$/ if (/^PORT:/);
   ($logfile) = $_ =~ /^FILE:\s+(.+)$/ if (/^FILE:/);
   ($window) = $_ =~ /^WINDOW:\s+(\d+)$/ if (/^WINDOW:/);
   ($ok) = $_ =~ /^OK:\s+(\d+)$/ if (/^OK:/);
   ($warning) = $_ =~ /^WARNING:\s+(\d+)$/ if (/^WARNING:/);
   ($errorlog) = $_ =~ /^ERRORLOG:\s+(.+)$/ if (/^ERRORLOG:/);
   ($sok) = $_ =~ /^SOK:\s+(\d)$/ if (/^SOK:/);
   ($swarning) = $_ =~ /^SWARNING:\s+(\d)$/ if (/^SWARNING:/);
   ($scritical) = $_ =~ /^SCRITICAL:\s+(.+)$/ if (/^SCRITICAL:/);
  }

close(FILE);

open(STDERR, ">$errorlog") or die "Could not open $errorlog: $!\n";

$window *= 60; # convert window in minutes to seconds
#$window = $epoch - $window; # calculate orignal time window
#print "$errorlog $logfile";
#
```

```
# getlines function returns number of lines between specified time
# window. Function takes no arguments. Function will return -1 in
# case of any error and log error message to ERRORLOG file.
#

sub hashValueDescendingNum {
  $ip_Hash{$b} <=> $ip_Hash{$a};
}

sub getlines {

   my $nlines = 0;
   my $bw;
   if (! ($bw = File::ReadBackwards->new($logfile))) {
         print STDERR "Couldn't open $logfile: $!\n";
         return -1;
   }
   my $log_line;
   my $winex = 0;
   my $ltime = 0;
   my $key;my $value;my $key_t;

         my $epoch = time();
         my $window1 = $epoch - $window; # calculate orignal time window
         while($log_line = $bw->readline) {
         my ($ltime) = $log_line =~ /squid\[\d+\]:\s+(\d+)/;
         my ($ip) = $log_line =~ /squid\[\d+\]:\s+\d+\.\d+\s+\d+\s+((\d+)(\.\d+){3})/;
      $ip_Hash{$ip}=$ip_Hash{$ip}+1;

         #print %ip_Hash;
         $nlines++ if ($ltime >= $window1);
         $winex++ if ($ltime < $window1);
         # Read 50 extra lines alter window is exceeded.
         # This is just a precautionary measure because
         # Logs may come out of time.
         last if ($winex >= 50);
   }
         $|++;

      my $send_mail=0;
         open(OUTFILE,">/home/logproc/mail.lst");
         print OUTFILE "NETMON Bombardment in Last 10 minutes of span\n";
```

```
        print OUTFILE "IP ADDRESS   - Netmon Hits\n";
        foreach $key_t(sort hashValueDescendingNum (keys(%ip_Hash))) {
        if ($ip_Hash{$key_t}>1000){
      $send_mail=1;
      #print "$key_t : $ip_Hash{$key_t}\n";
        print OUTFILE "$key_t - $ip_Hash{$key_t} \n";}
      }
        if ($send_mail==1){
      system("/home/logproc/mail.py nirav\@cse.iitb.ac.in NETMON \"`cat
/home/logproc/mail.lst`\"");
        $send_mail=0;
        }
        close(OUTFILE);
        return $nlines;
}


# startserver subroutine creates a TCP socket and listens on port specified in conf file.
# It can send three values to client depending
# upon the value returned by getlines function and values given in conf
# file.

sub startserver {
   my $server = IO::Socket::INET->new( LocalPort => $port,
                                       Type      => SOCK_STREAM,
                                       Reuse     => 1,
                                       Listen    => 1
                                     ) or die "Couldn't be a tcp " .
                                     " server on port $port : $@\n";
   my $client;
   while ($client = $server->accept()) {          #Blocking Call .. will block here till it get
connect request from logproc.pl
        my $nlines = getlines();
        # set default status to critical so that if anything is wrong
        my $status = $scritical;
        if ($nlines >= 0) {
          if ($nlines <= $ok) {
                $status = $sok;
          } elsif ($nlines <= $warning) {
                $status = $swarning;
          } else {
                $status = $scritical;
```

```
            }
        }
        $client->send($status);
        close($client);
   %ip_Hash=();
        }
}

startserver();

# Should never reach here.
exit(0);
```

server running on nagios machine

```
#!/usr/bin/perl -w
use strict;

use IO::Socket;

my ($remote_host, $remote_port) = @ARGV;

my $socket = IO::Socket::INET->new(PeerAddr => $remote_host,
                        PeerPort => $remote_port,
                        Proto    => "tcp",
                        Type     => SOCK_STREAM)
   or die "Couldn't connect to $remote_host:$remote_port : $@\n";

my $answer = <$socket>;

print $answer;
```