Centralized Caching over OpenFlow

Prasanna Naik , Fengyuan Gong , Payal Godhani [*], Omkar Dalvi [*]

Dept. of Electrical and Computer Engineering, Dept. of Computer Science [*]
North Carolina State University, Raleigh, NC, USA
{pmnaik,fgong,pgodhan[*],okdalvi[*]}@ncsu.edu

Project URL: https://sites.google.com/a/ncsu.edu/573project/

***Abstract*— This document gives the proposal of implementing the centralized caching on OpenFlow. The performance of link delay and throughput are compared between the system on regular individual caches and controller based cognitive cache over OpenFlow.**

I. Introduction

Network bandwidth has become a limited resource owing to an increasing demand in delivery of content over the high speed internet. Huge amount of similar requests for large web pages, images and video files have been congesting the network resulting in longer delay and lower throughput. To reduce the load on web server and for better utilization of available bandwidth, a technique called caching can be used in which the web documents are stored temporarily in local storage [1]. Web caching is usually implemented on the subnet gateway router or switch to reduce the frequent repeated requests to the remote server by holding the content in local storage as shown in Figure 1. Gateway switches run web caching based on HTTP requests received from the respective LANs. Assume that gateway switches are not sharing the local cache with each other. Suppose that there are lots of requests for a large video file from LAN1 which has already been cached in switch2. Since switch1 has no idea of the caching in other switches, it still downloads the file from remote server and then inserts the file into its local cache, which consumes bandwidth. In order to optimize this caching scheme, we propose a centralized caching over OpenFlow by redirecting flows intelligently to the global cache switch, which saves bandwidth by eliminating the redundant requests to the remote web server.

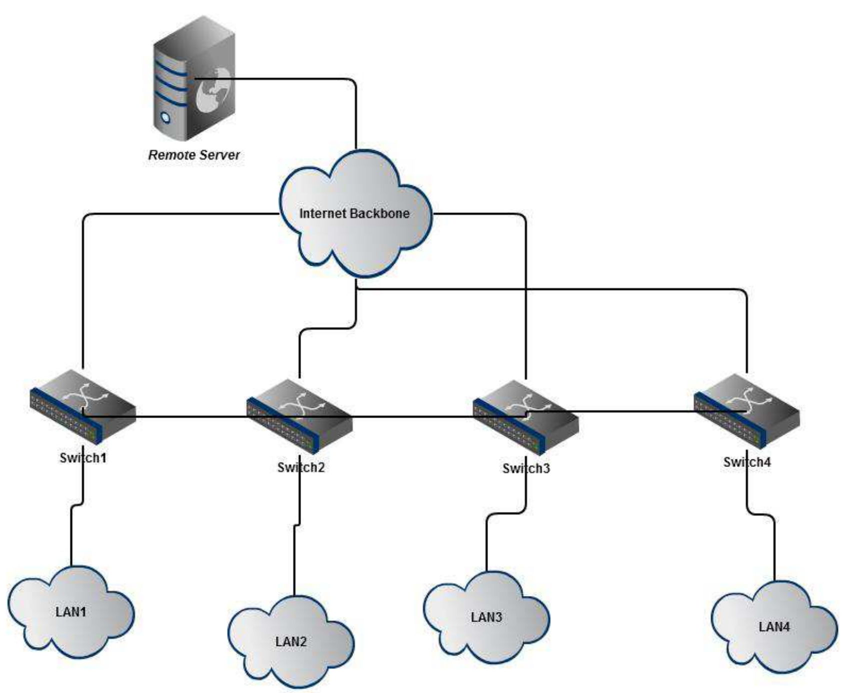


Figure 1. Normal topology with cache on gateway switch

With the help of OpenFlow, traffic can be redirected based on the controller's forwarding policy [2]. In Figure 2,

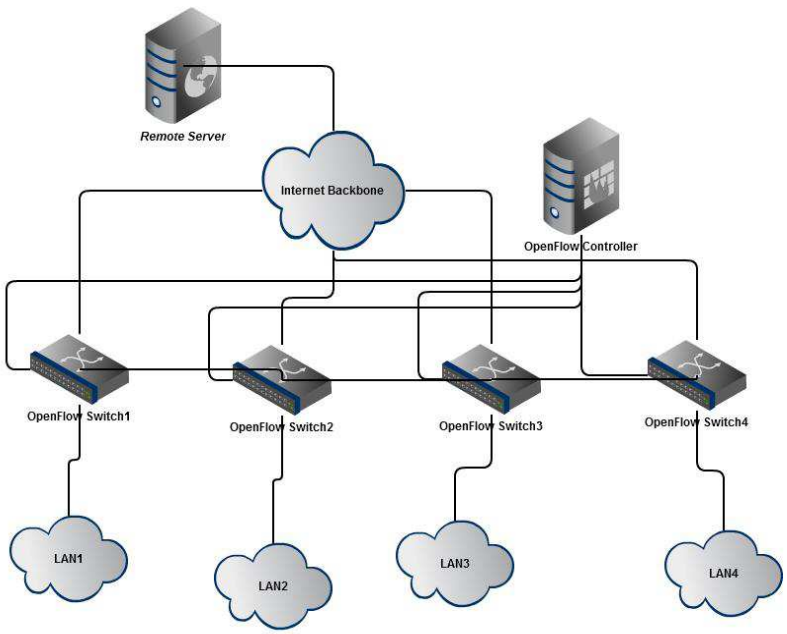


Figure 2: Centralized cache on OpenFlow

OpenFlow controller communicates with each switch to regulate the incoming and outgoing flows. Assume switch2 is the global cache switch and other switches do not run local cache. Whenever the switch receives an incoming request, it looks up the local forwarding flow table. If there is no policy for the request, the switch will send the request to the controller. The controller will keep an entry for the request and inform global cache switch to download the requested file and save it into local cache. Next time if switch4 receives the same request, it will contact the controller which will redirect the request to the global cache switch for the first request. After the first GET request, the flow table of switch4 is populated with the forwarding hop as the local switch for the particular website. Henceforth requests will be directly sent to cache switch and no need of controller intervention. In this way, the centralized cache reduces bandwidth consumption by communicating between OpenFlow switches.

II. SYSTEM MODEL

This section illustrates the system composition.

A. *Major Component Decomposition*

1) OpenFlow Controller Design

OpenFlow controller and OpenFlow switch will be implemented and tested on a GENI test bed [3] [5]. To design the controller, NOX platform will be used which is a C++ based platform that gives the ability to developers to implement new controllers by writing NOX modules. [4]

When a GET request is received for a particular webpage, the switch will check its flow table for any entry of the particular website. If a flow table entry is found, then the request is redirected to the local switch containing cache. The cache will be holding the contents of the requested website, and website will be downloaded to the requesting switch from the local switch with cache.

If a flow table entry is not found for a particular website, then the request if forwarded to the OpenFlow controller which will initiate the local switch cache to download the particular website and store in its cache. Since the website has been stored in local cache, the requesting switch is instructed by the OpenFlow controller to populate its flow table with the latest GET request entry. And any subsequent requests to the same website will be redirected to the local switch with cache without any intervention needed from OpenFlow controller.

2) Forwarding among OpenFlow switches

Depending upon the entry of a GET request in a switch's flow table, forwarding the GET request will be performed either to the local switch with cache or the actual web server. If a flow table entry is found for a given request, the GET request will be sent to the switch with cache and data transfer will be initiated from the local switch. If a flow table entry is not available, then the GET request is forwarded to the OpenFlow controller, which will take the decision of either redirecting the GET request to local switch cache or the actual web server.

3) Cache implementation in centralized switch

Upon receiving GET request for each of the web page, the page is cached and its entry is made into the cache table maintained by the controller process. Once the cache is full, the least popular content is deleted from the cache (Least Popular First out Strategy is used for cache eviction). Every content stored in the cache has a content-id associated with it which is used as a reference by the controller process and open flow switches. Whenever a GET request for a particular content arrives at a switch it checks its own flow table for content-id matching.

a. If an entry is found, the requested content is present in the local switch cache so the switch redirects the flow to the switch having the cache.

b. If entry is not found the switch redirects the flow to the OpenFlow controller which takes an action of caching the requested content and assigning content-id to it. At the same time an entry is also made into the flow table of requesting switch with content-id field and cache switch identifier on which the content is present.

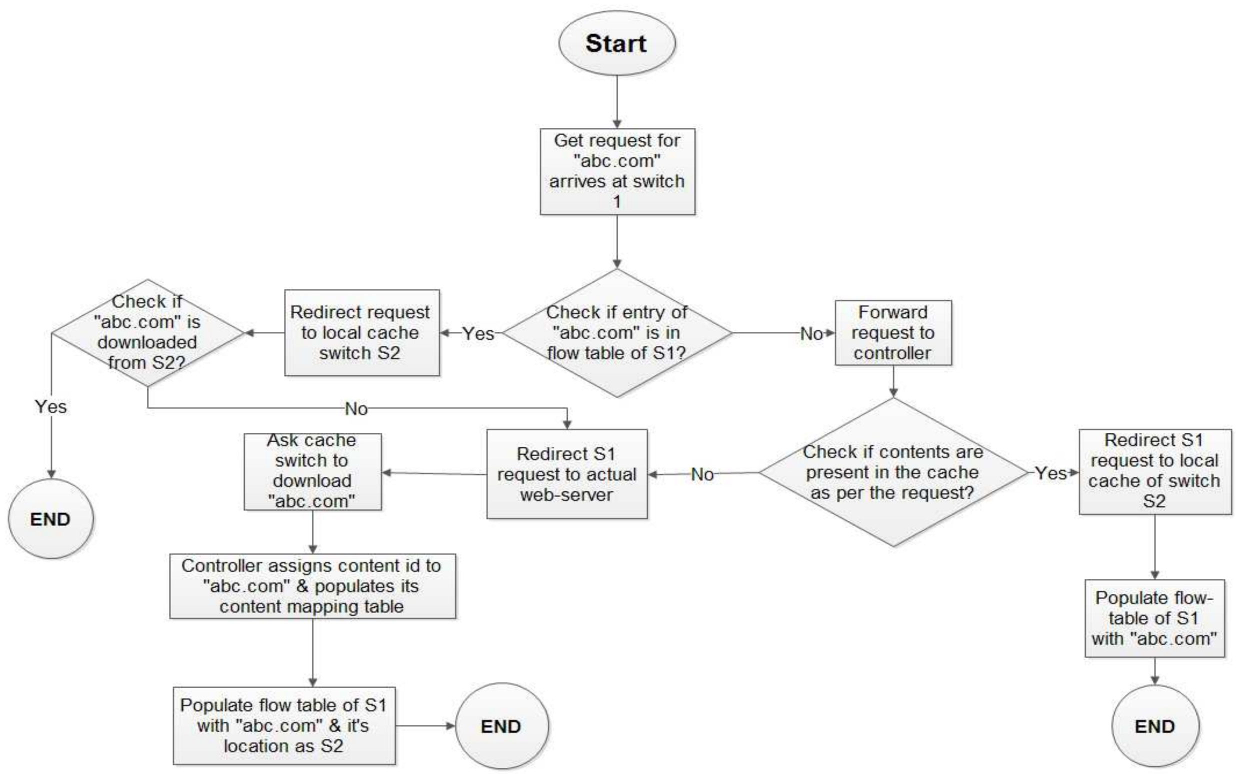
B. Platform Used

Platforms: GENI, OpenFlow, NOX-d

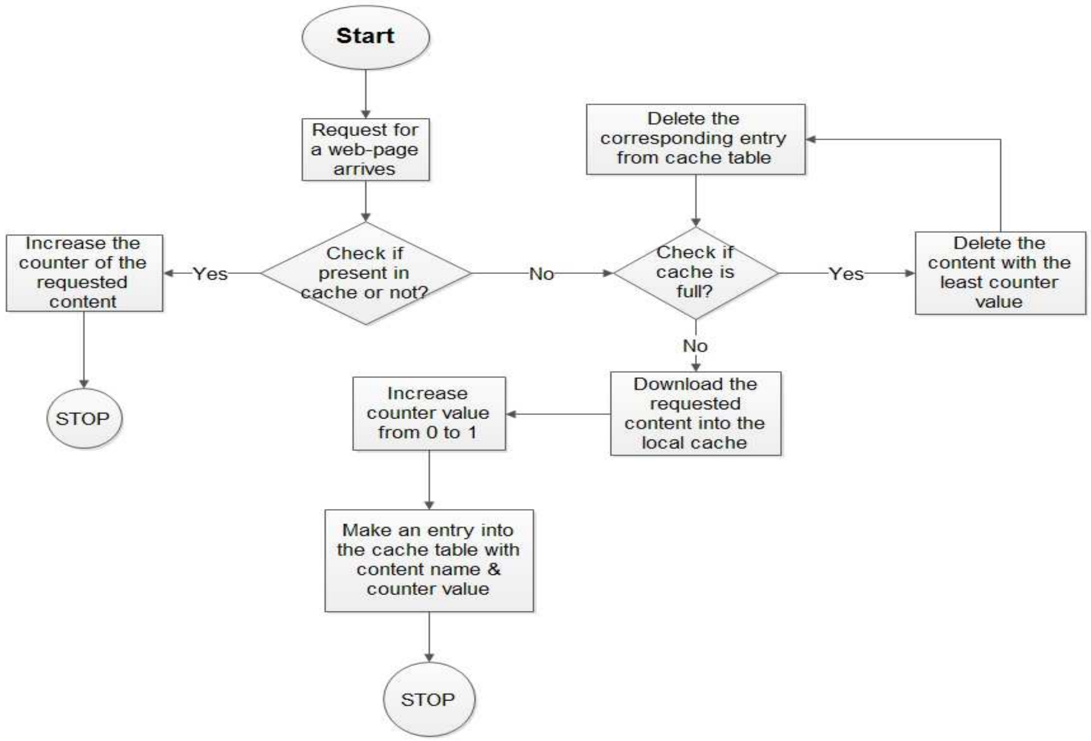
Programming Language: C/C++, Python, Perl

Basically, testbed is set up on GENI VMs. OpenFlow is installed on four switches and one controller.

C. Flow Chart for Openflow controller and switch actions.



D. Flow Chart for logic used in Cache implementation.



III. Design and Development plan

A. Per-component Development Phases

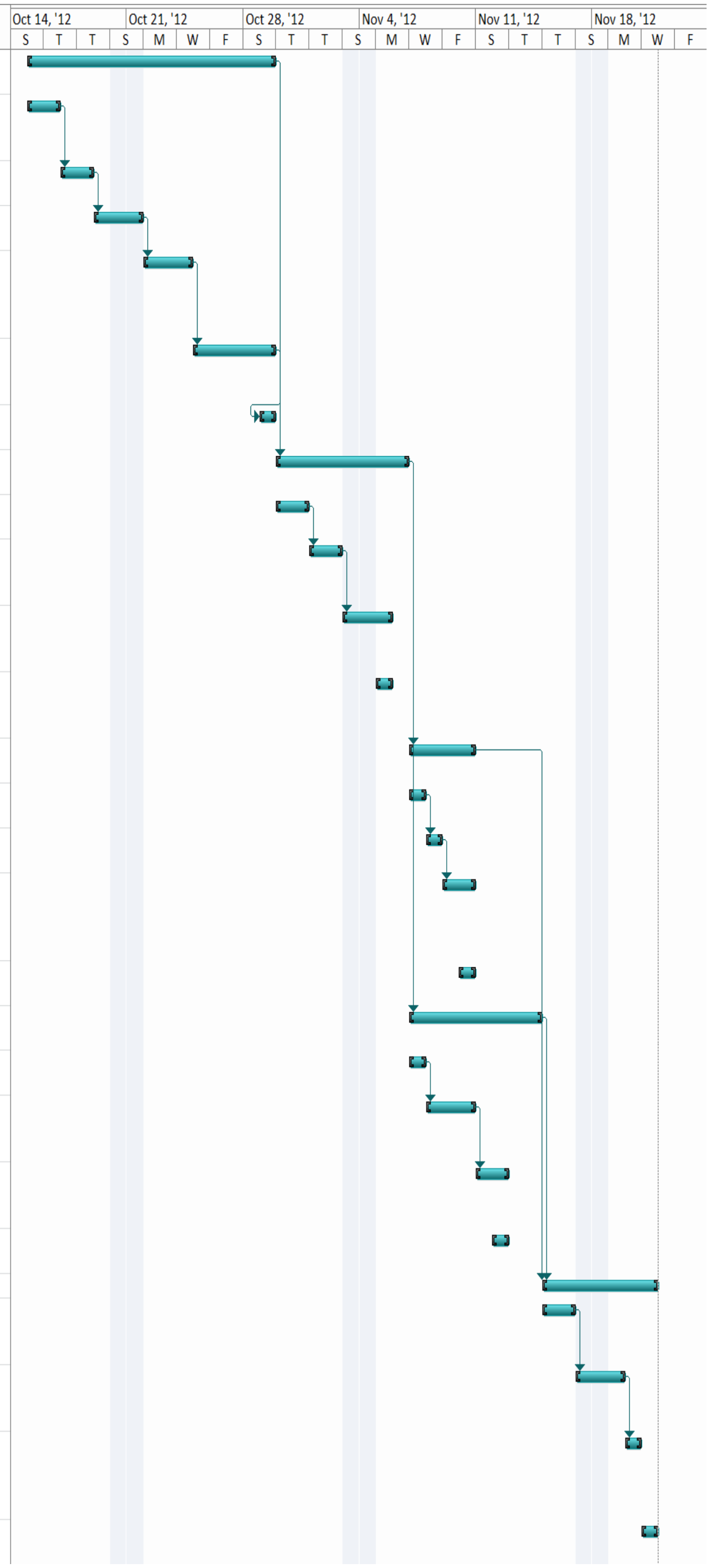
1) OpenFlow Implementation: This part is composed of the detail design of OpenFlow switch and controller. It can be subdivided into the following components:
   a. Installing OpenFlow switch on the VM gateway switches
   b. Installing OpenFlow controller on the VM control server
   c. Communication between OpenFlow controller and switch
   d. Changing forwarding entries according to the policy given by controller
   e. Implementing caching algorithm on the controller
   f. Designing caching pool on the central cache
2) LAN clients: The clients are supposed to keep sending HTTP GET requests mainly for images and videos to the remote server within a certain time period.
3) Caching Design:
   a. Defining the size of cache
   b. Assign counter for every new content downloaded into cache
   c. Increment counter of content whenever request for the specific content arrives at cache
   d. Update cache table with content name and counter value
   d. Keep a tab on cache size, if cache size is full then delete the Least Used content (content with least counter value)

B. Per-member Responsibility

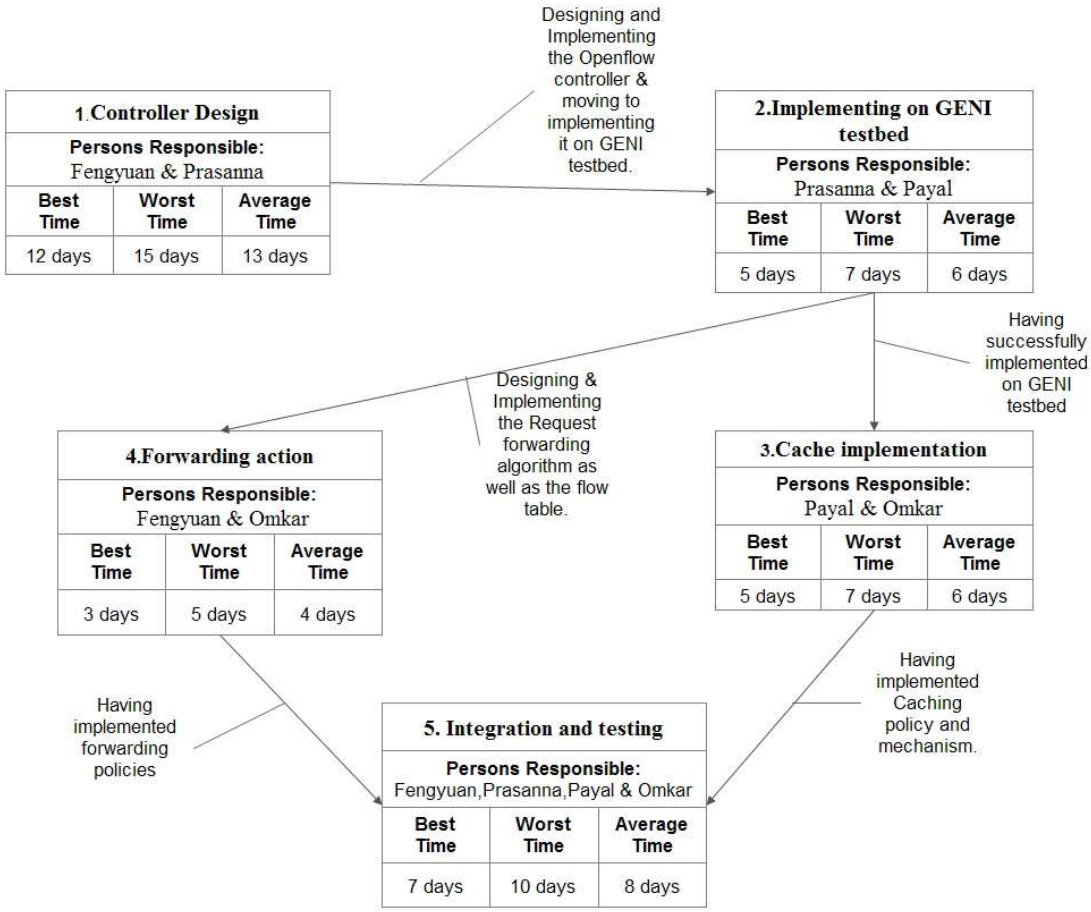
Controller design: Fengyuan, Prasanna
Cache implementation: Payal, Omkar
Forwarding action: Fengyuan, Omkar
Implementing on GENI testbed: Prasanna, Payal
Integration and testing: ALL

C. Timeline

ID		Task Mode	Task Name	Duration	Start	Finish	Predecessors
1			1) Controller design (Fengyuan, Prasanna)	11 days	Mon 10/15/12	Mon 10/29/12	
2			1.1) Learning about Openflow & Openflow controllers	2 days	Mon 10/15/12	Tue 10/16/12	
3			1.2) Trying out the openflow tutorials	2 days	Wed 10/17/12	Thu 10/18/12	2
4			1.3) Learning & getting acquainted with NOX	2 days	Fri 10/19/12	Sun 10/21/12	3
5			1.4) Writing codes to modify the functionality of the controller in context of the project	3 days	Mon 10/22/12	Wed 10/24/12	4
6			1.5) Testing the controller & it's working & debugging the codes.	3 days	Thu 10/25/12	Mon 10/29/12	5
7			Controller Design accomplished	1 day	Mon 10/29/12	Mon 10/29/12	6
8			2) Implementation on GENI testbed (Prasanna &	6 days	Tue 10/30/12	Tue 11/6/12	1
9			2.1) Creaing slice and creating Virtual Machines	2 days	Tue 10/30/12	Wed 10/31/12	
10			2.2) Implementing the Openflow controller into GENI	2 days	Thu 11/1/12	Fri 11/2/12	9
11			2.3) Creating switches & building a network hierarchy using VMs	2 days	Sat 11/3/12	Mon 11/5/12	10
12			Implementation on GENI testbed completed(Made Interim Report)	1 day	Mon 11/5/12	Mon 11/5/12	
13			3) Forwarding Action (Fengyuan, Omkar)	4 days	Wed 11/7/12	Sat 11/10/12	8
14			3.1) Studying the flowtable of a switch	1 day	Wed 11/7/12	Wed 11/7/12	
15			3.2) Understanding how getRequest() works	1 day	Thu 11/8/12	Thu 11/8/12	14
16			3.3) Programming the switches & controller to follow the forwarding scheme	2 days	Fri 11/9/12	Sat 11/10/12	15
17			Forwarding Scheme Implemented	1 day	Sat 11/10/12	Sat 11/10/12	
18			4) Cache implementation (Payal & Omkar)	6 days	Wed 11/7/12	Wed 11/14/12	8
19			4.1) Studying various cache policies used	1 day	Wed 11/7/12	Wed 11/7/12	
20			4.2) Implementing the cache policy on the controller and the	3 days	Thu 11/8/12	Sat 11/10/12	19
21			4.3) Testing the caching scheme & making it more efficient	2 days	Sun 11/11/12	Mon 11/12/12	20
22			Cache Implementation completed	1 day	Mon 11/12/12	Mon 11/12/12	
23			5) Integration & Testing	5 days	Thu 11/15/12	Wed 11/21/12	13,18
24			5.1) Integrating the Forwarding & Caching scheme	2 days	Thu 11/15/12	Fri 11/16/12	
25			5.2) Performing tests on the implemented scheme by varying test cases	2 days	Sat 11/17/12	Mon 11/19/12	24
26			5.3) Troubleshooting & making the implementation more refine as well as making	1 day	Tue 11/20/12	Tue 11/20/12	25
27			Implementation & Testing of the project completed	1 day	Wed 11/21/12	Wed 11/21/12	



D. PERT



IV. Demo and Test Plans

Scripts will be used to control client requests for web pages, images and video files from the remote server.

Scripts will make sure the cache is filled and overflowed with content to demonstrate our Cache implementation.

Tests to verify different logic installed at controller and switch will be aptly performed.

We will also record the throughput and delay per link by setting up timers. In the end, link throughput and delay will be compared between centralized cache and the regular distributed cache system.

Acknowledgment

We are thankful to Dr. Dutta for the help given and directing us on the right path for the project.

References

[1] http://en.wikipedia.org/wiki/Web_cache.

[2] http://www.openflow.org/

[3] http://www.openflow.org/wp/openflow-components/

[4] http://www.noxrepo.org/

[5] http://groups.geni.net/geni/wiki/OpenFlow/Controllers#OpenFlowControllersinGENI