HSM: A Hybrid Streaming Mechanism for Delay-Tolerant Multimedia Applications

Dissertation

Submitted in partial fulfillment of the requirement for the degree of

Master of Technology

By

Annanda Th. RATH
(Roll No: 04329202)

Under the guidance of
Prof. Sridhar Iyer

Kanwal Rekhi School of Information Technology
Indian Institute of Technology, Bombay

2006

Dedicated to my family

Dissertation Approval Sheet

This is to certify that the dissertation entitled

HSM: A Hybrid Streaming Mechanism for Delay-Tolerant Multimedia Applications

By

Annanda Th. RATH
(Roll no. 04329202)

Is approved for the degree of Master of Technology

____
Prof. Sridhar Iyer
(Supervisor)

____
Prof. Anirudha Sahoo
(Internal Examiner)

____
Prof. Om Damani
(Additional Internal Examiner)

____
Prof. Abhay Karandikar
(Chairman)

Date: __________

Place: __________

INDIAN INSTITUTE OF TECHNOLOGY BOMBAY
CERTIFICATE OF COURSE WORK

This is to certify that **Mr. Annanda Thavymony RATH** was admitted to the candidacy of the M.Tech. Degree on 1th July 2004 and has successfully completed all the courses required for the M.Tech Program. The details of the courses work done are given below

Sr. No	Course No	Course Name	Credits
		Semester 1 (Jul-Nov 2004)	
1	HS 699	Communication and Presentation Skills	4
2	IT 601	Mobile Computing	6
3	IT 605	Computer Networks	6
4	IT 619	IT Foundation Lab	10
5	IT 623	Foundation Course of IT Part II	6
6	IT 694	Seminar	4
		Semester 2 (Jan-April 2005)	
7	EE 701	Introduction to MEMS	6
8	IT 604	Human Computer Interaction Design	6
9	IT 620	New Trends in Information Technology	6
10	IT 628	Information Technology Project Management	6
11	IT 680	Systems Lab	6
		Semester 3 (Jul-Nov 2005)	
12	CS 681	Performance Analysis of Computer Systems and Network	6
13	IT 653	Network Security	6
		Semester 4 (Jan-April 2006)	
14	IT 610	Quality of Service in Networks	6
		M.Tech Project	
15	IT 696	I Stage Project	18
16	IT 697	II Stage Project	30
17	IT698	III Stage Project	42

I.I.T Bombay   Registrar (Academic)

Date:……..

Abstract

Streaming is a technique for transferring data such that it can be processed as a steady and continuous stream. In a content dissemination network, typically a central server at the source streams the content in response to client requests. We term it as *Pure Streaming Mechanism* (PSM). Considering that in a dissemination network controlled by a Content Service Provider (CSP), the backbone links are highly provisioned, **using a streaming server** at the source leads to underutilization of these links. Also the links are occupied for the duration of play out of the multimedia content. This is because according to the streaming property, the streaming server sends only the amount of data equivalent to the streaming encoded rate to the client irrespective of the available bandwidth in the path. Delay-tolerant applications are a special class of on demand multimedia applications where clients request the start of play back at a time specified by $(t + dt_i)$ where t is the current time and dt_i is the delay tolerance acceptable to client i.

In this thesis report, we present a novel Hybrid Streaming Mechanism (HSM) for Delay-Tolerant Multimedia Applications to enhance the following performance parameters: (i) number of serviced clients, and (ii) delivered stream rate at clients. HSM allows streaming from strategically chosen intermediate nodes to which the content is dynamically transferred from the source, using FTP (File Transfer Protocol). As FTP uses the entire bandwidth to transfer the data, it frees up the high bandwidth links faster for serving requests from other clients sharing these links, increasing the efficiency of the service. In HSM, transferred contents are temporally cached at the intermediate node (Streaming Point). Such temporary caching further enhances the performances of HSM as requests for the same content are serviced from the cache.

Contents:

List of Figures

List of Tables

Chapter 1

Introduction

Streaming media is expected to become one of the most popular types of web content in the future. Typically, a central server handles streaming services, is responsible for servicing all client requests. We term this *Pure Streaming Mechanism* (PSM), where streaming capability is available only at the source. In a large-scale network with a large number of concurrent client requests, streaming from only one source has been proven to be inefficient because of the limitation of streaming server capacity and link bandwidth constraints in the network [4][10][3]. To improve the performance of the streaming services as well as to serve more clients, many techniques have been developed, including content replication, and resource sharing [6][9][7][5][18][1][3][14]. Content replication is an efficient way to increase the number of serviced clients, reduce network traffic and workload at central server. However such techniques require large storage, since typically, the content is downloaded in advance in anticipation of client requests. Resource sharing strategies increases the number of serviced clients by exploiting the high skewness in video access patterns. These strategies can be classified into five main categories: Batching [21][23][27], Patching [10], Piggy-backing [25], broadcasting [26], and Interval caching [22][24][20]. The resource sharing techniques come at the expense of requiring every high additional bandwidth and buffer space at the client (See chapter 2 for more details).

In this thesis report, we present a *Hybrid Streaming Mechanism* (HSM) to increase the efficiency of a Content Service Provider (CSP) by using a combination of FTP and streaming mechanisms in the context of a special class of on demand applications termed *delay-tolerant* multimedia applications [15]. In delay-tolerant applications, clients request for the multimedia content specifying their *requirements*, stream *quality* -- a minimum rate at which they want to receive the stream, and *delay tolerance* -- the time they will wait for the play out of the stream. Applications in distance education and corporate training where multiple clients at different time slots access the same contents are typical examples that fall in this category of applications. Note that mechanisms proposed in the literature to efficiently serve requests for multimedia content assume that play out at clients start immediately [5][2][18][1][8].

1.1 What is Streaming?

Streaming is a technique for transferring data such that it can be processed as a steady and continuous stream. Streaming technologies are becoming increasingly important with the growth of the Internet because most users do not have fast enough access to download large multimedia files quickly. With streaming, the client browser or plug-in can start displaying the data before the entire file has been transmitted. For streaming to work, the client side receiving the data must be able to collect the data and send it as a steady stream to the application that is processing the data and converting it to sound or pictures. This means that if the streaming client receives the data more quickly than required, it needs to save the excess data in a buffer. If the data doesn't come quickly enough, however, the presentation of the data will not be smooth. One more feature of streaming is that streaming server sends only the amount of data equivalent to the streaming encoded rate to the client irrespective of the available bandwidth in the path.

1.2 Delay Tolerant Multimedia Applications

Delay-tolerant applications are a special class of on-demand multimedia applications where clients request the start of play back at a time specified by $(t + dt_i)$ where t is the current time and dt_i is the delay tolerance acceptable to client i. Examples of delay-tolerant applications include universities offering their courses to a set of global subscribers and service providers streaming movies requested by their clients.

1.3 Client's Requirements (Client's Constraint)

Client requirements are minimum acceptable requirements for a given client. It consists of two parameters: (i) stream quality, a minimum rate at which a client wants to receive the stream, and (ii) Delay-tolerance, a time at which the client wants the play back to start.

1.4 Pure Streaming Mechanism

Pure Streaming Mechanism (PSM) is a mechanism where the source node contains the only streaming server in the network that processes all the requests from clients. Whenever the client's request arrives, server establishes the connection and starts streaming to client. The amount of data for a period equivalent to client's delay tolerant must be buffered at client side. In this mechanism there may be underutilization of backbone bandwidth in case the delivered stream rate at the client is less than bandwidth of links in the backbone. This is because according to the streaming property, the streaming server sends only the amount of data equivalent to the streaming encoded rate to the client irrespective of the available bandwidth in the path.Note that while the source streams the content to the client, the links from the source to the client are occupied for a period equivalent to the streaming duration; other clients who share those links can not use the links.

1.5 Motivation (Survey of Cambodia Network)

This section we take a look on a network topology of a small country in southeast Asia, Cambodia is a country in southeast Asia consists of 24 provinces, there are three main city in the country, One is Phnom Penh city, the central city for economic and cultural activity, Siem Reap Province, the city of tourist and ancient monuments and the last one is Sihanouk Ville, the city of the tourist and economic. Cambodia's network is a centralized kind of network architecture, there is only one source at central city, and distributes to all the provinces in the country. In Cambodia, there are three main Internet Service Providers and source of information is situated at central city (Phnom Penh). However Cambodia has 24 provinces, only some of them have the Internet connectivity. Figure (2.2) shows a simplify network topology of Cambodia. Figure (2.2) shows that there is only one source of information situated in Phnom Penh city, and connected to all the provincial nodes. In Cambodia, three types of Internet connection are available, modem, DSL, and Satellite system (rarely used). There are a lot of connectivity problems, the links are generally slow as there are many users, but fewer infrastructures and the cost is very high. Streaming video or audio is also available in Cambodia, but less popular because of the connection is slow and the cost of Internet service is generally unaffordable. Distance Education Program and Video Conferencing become popular in the last two years, as many universities and the organizations update the infrastructure to adjust the new technology. The growth of the Internet Users and Technology in Cambodia; give us a hope that in the near future, streaming market in Cambodia would be a right target.

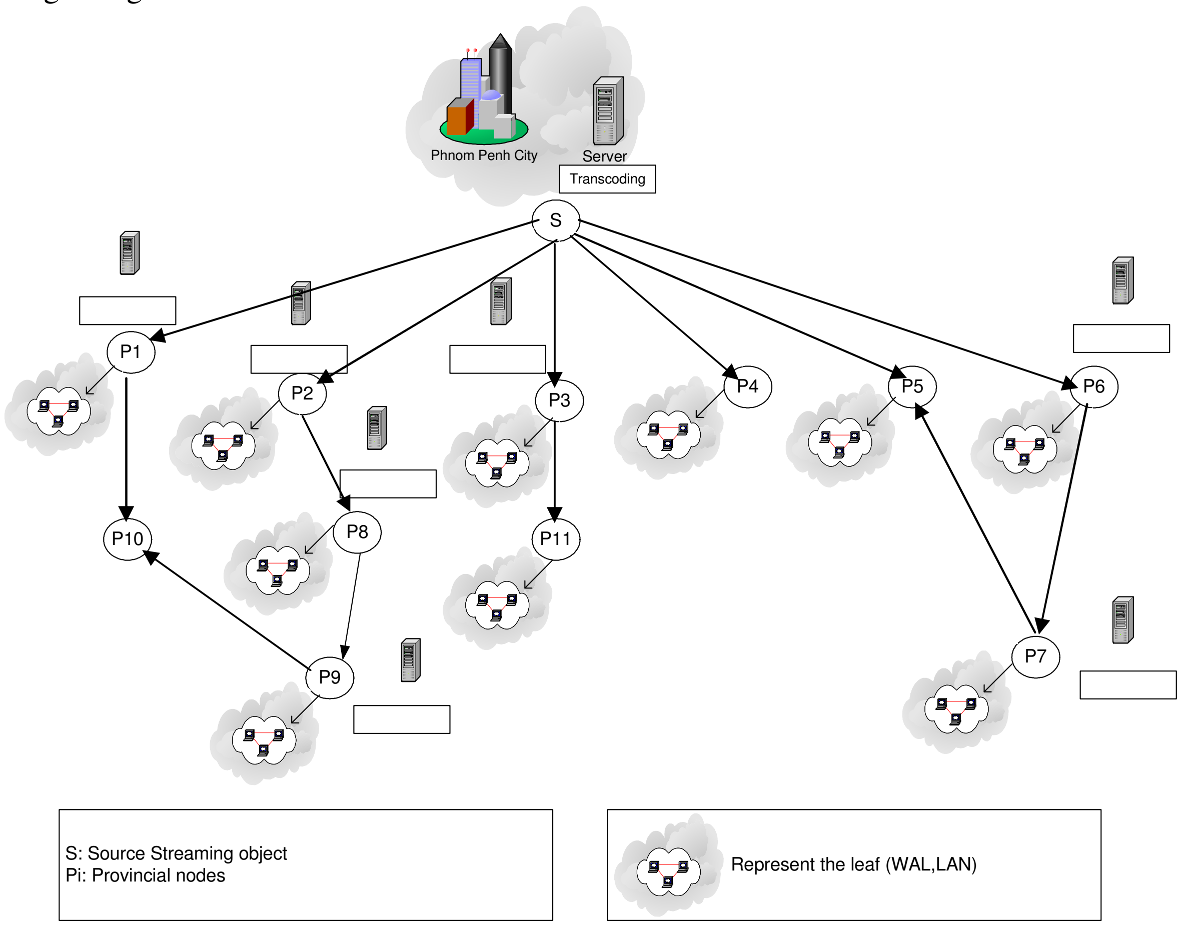

Figure 1.1: A Simplify Cambodia Network

1.6 Problem Definition

In a content dissemination network, typically a central server at the source streams content in response to client requests. Considering that in a dissemination network controlled by a Content Service Provider (CSP), the backbone links are highly provisioned, using a streaming server at the source leads to underutilization of these links. Note that while the source streams the content to the client, the links from the source to the client are occupied for a period equivalent to the streaming duration; other clients who share those links can not use the links. This leads to the rejection of requests for a period equivalent to the streaming duration of the content being streamed. **Using only one streaming server at the source** to serve all the client requests in the network has some drawbacks. (i) Link bandwidth can be underutilization because of the streaming property, (ii) Client occupies the link for entire streaming duration and other client who comes later has to wait till the first client releases the link, (iii) the number of concurrent serviced clients is limited because of the limit capacity of the streaming server, (iv) the work load is at the server central server, as all the client requests go to it.

In order to solve the above problems and to improve the performance of the streaming system (especially, the number of serviced clients), many techniques have been developed, including the content replication and resources sharing. However these techniques require a large storage and high additional bandwidth given the size of multimedia content.

1.7 Solution Outline

In order to solve the above problem, we propose a new streaming mechanism, termed Hybrid Streaming Mechanism (HSM) to improve the efficiency of the streaming performance and to maximize the link bandwidth utilization in the backbone network.

HSM allows streaming from strategically chosen intermediate note, termed *Streaming Point*, to which the content is dynamically transferred from source, using FTP (File Transfer Protocol). As FTP uses the entire link bandwidth, it frees up the high bandwidth link faster for serving requests from other clients sharing these links, increasing efficiency of service. In HSM, transferred content are temporally cached at the streaming point. Such temporally cached further enhances the performance of HSM as requests from the same content are serviced from the cache. The advantages of HSM are: (i) improving the performance of streaming service, (ii) maximizing the number of serviced clients, (iii) maximizing the bandwidth utilization in the backbone network, (iv) improving the delivered stream rate at the client, (v) Reducing the traffic in the network as well as the work load at central server.

Simulation results demonstrate that by combining the FTP and streaming mechanism intelligently, on average performance improvement of 40% is achieved compared with PSM.

Our scheme works under the following assumptions: (i) the links have dedicated bandwidth provisioned for the given application; (ii) the selected intermediate nodes have the streaming capability and (iii) multicasting is also supported in the given network topology.

1.7.1 Hybrid Streaming Mechanism

Hybrid Streaming Mechanism is the combination of FTP and streaming mechanisms, HSM allows selected relay node with streaming capability to stream the content instead of central server. In HSM, data flow is divided into two parts: (i) a FTP flow from source to strategically chosen intermediate node(s) and (ii) a streaming flow from that node to client(s) in the sub trees rooted at that node. As FTP uses the entire link bandwidth, it frees up the high bandwidth links faster for servicing requests from other clients sharing these links, increasing the efficiency of service. Central to HSM are the strategies used for selecting appropriate intermediate nodes as *streaming points* to enhance the system performance.

1.7.2 An Illustrative Example

In this chapter, we present an example to illustrate the differences in PSM and HSM and motivate the need for the proposed hybrid streaming mechanism.

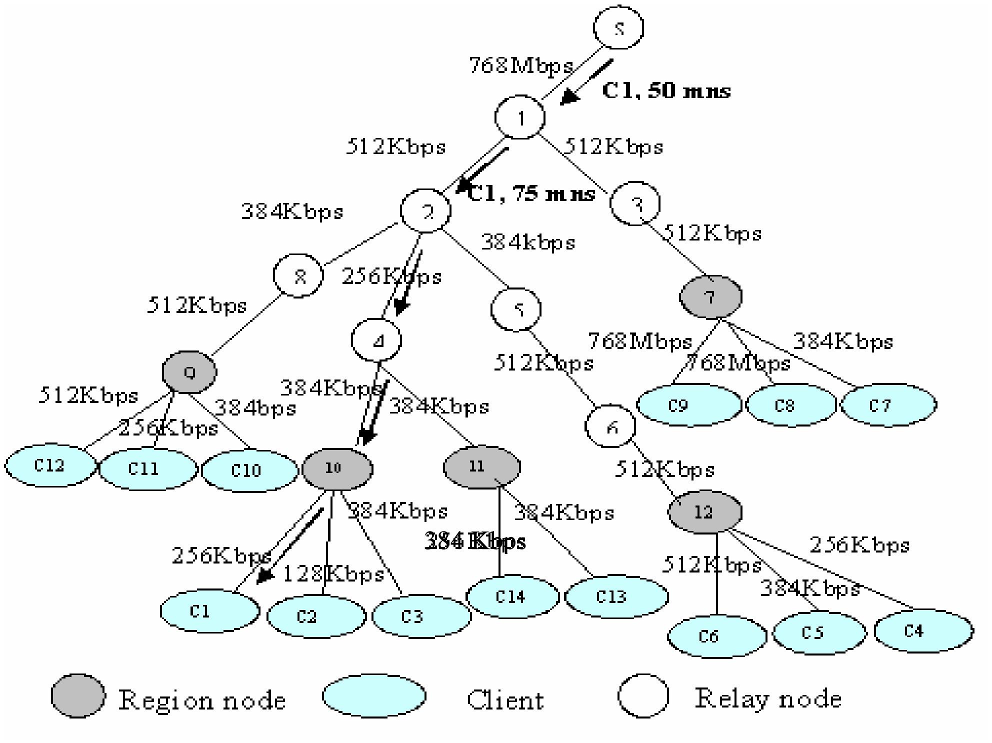

Figure 1.2: A simple tree network topology

A simple network model in Figure (1.2) represents a heterogeneous dissemination network as a tree, with the source S at the root and the clients C_1, C_2,…, C_{14} at the leaves. All other intermediate nodes serve as relay nodes. A node that directly serves a group of clients is termed a region node. We use the term region to refer to the sub tree that contains the region node and the clients it serves. For example in Figure (1.2), the network has 5 regions with nodes 7, 9, 10, 11, and 12 serving as region nodes. We refer to the network from S to the region nodes as the backbone of the content dissemination network. Let us assume that at time zero, client C1 joins the network. Client C2 joins 10 minutes later and clients C6, C9, and C12 join 75 minutes later. Table 2.1 gives the details of clients requesting the stream.

We illustrate the difference between PSM and HSM by considering the above arrival pattern, as discussed below:

Case 1: PSM

In a streaming application where clients do not tolerate any startup delay, the weakest link in a client's path dictates the encoding rate of the stream to provide loss-free transmission to that client.

In PSM, the source node contains the only streaming server in the network that processes all the requests from clients. In this mechanism, there may be underutilization of backbone bandwidth in case the delivered stream rate at the client is less than bandwidths of links in the backbone. This is because according to streaming property, the streaming server sends only the amount of data equivalent to streaming encoded rate to the client irrespective of the available link bandwidth in the path. For example if the streaming object is encoded at 256Kbps, only 256 kb is sent by server to client every second even when the link bandwidth is greater than 256 Kbps. In delay-tolerant applications, the delivered stream rate at a client can be enhanced using buffers in the nodes in the path of the client [].

Considering client C1 which requests the stream at t=0. Let the play out duration of the stream be 2 hrs. We first calculate the delivered stream rate at C1, considering its delay tolerance. C1 gets 320 Kbps. (The formula used is derived in Section 3.3.2). When the streaming server is placed at the source, stream flows from S to C1 along the path (S-1-2-4-10-C1). The server sends the stream encoded at 320 Kbps, which occupies the path for 2 hrs, the play out duration. Table 1.2 shows the available link bandwidths in the path of C1 when it is being serviced using PSM.

C14 joins the network at time t=10. Since C14 shares links (S-1-2-4) with C1, its request cannot be serviced. Client C6 joins network at t=75. It shares links (S-1-2) with C1. Given its requirements, C6 can get only a stream rate of 240 Kbps. Since this rate is below C6's minimum required rate, request from C6 is also rejected. Similarly, clients C9 and C12 also get rejected. Thus, using PSM, only one out of five clients is serviced by the CSP. HSM allows selected relay nodes with streaming capability to stream the content instead of central server (source). When a request arrives at the central server, it determines the stream rate that can be provided to the client given the client's delay tolerance requirement and the location of the streaming server, termed streaming point. The central server then starts sending data by using FTP to the chosen streaming point and allows it to serve the clients.

Table 1.1: Details of clients requesting the stream

Clients	Request Time (Minutes)	Requirements (Delay-Tolerant, rate)	PSM	HSM
C1	0	(30,256)	Served	Served
C14	10	(60,256)	Not served	Served
C6	75	(30,256)	Not served	Served
C9	75	(05,480)	Not served	Served
C12	75	(30,256)	Not served	Served

Case 2: HSM

As FTP uses the entire bandwidth for transferring data, the links between the central server and the streaming point are fully utilized. As a result, these links are freed earlier compared with PSM. In HSM, the data sent by source to the streaming point is cached at that node for a period equivalent to the streaming duration, in the interest of future requests for the same content. We term this period *Time To Live of the Content* (TTLC). TTLC at a relay node is extended when a new request is made for the same content before the original TTLC expires. For a detailed discussion refer to section 3.4.2.

Consider the same scenario presented in Table 1.1 with HSM. As before, the delivered stream rate at C1 is 320 Kbps. But now we choose node 4 as the streaming point. (Details of streaming point selection are presented in Section 3.2.2) FTP mechanism is used to transfer data from the source to the streaming point along the path (S-1-2-4). Table 1.2 shows the available link bandwidths in the path of C1 when it is being serviced using HSM.

C14 joins the network 10 minutes after C1. Since C14 shares links (S-1-2-4) with C1, it is not possible for C14 to initiate a new stream from S. However, since C14 is requesting for the same streaming object, as the object is being cached at node 4, its request can be serviced from node 4. C14 gets the stream at 320 Kbps which is greater than it minimum rate requirement. Note that C14 doesn't join C1's ongoing streaming; a new stream is sent to

C14 from node 4. Clients C6, C9, and C12 join the network at time t=75. At t=75, C1's transmission across links S-1 and 1-2 are finished and the links become free. All three clients C6, C9, and C12 get serviced with a stream rate of 480Kbps, their streaming points being at nodes 5, 1, and 8 respectively. As a result, under HSM all 5 clients are serviced. In this the above example, we assume that the file size of the streaming content requested by the client C1 with 320 Kbps is 2250 MB.

The above example we demonstrate that HSM performs better than PSM in terms of number of serviced clients. This is because in HSM, links from the source to the streaming point are freed sooner than PSM. Another important feature of HSM is that the copy of the content is temporarily cached at the streaming point. Requests for the same content from other clients in the sub tree can be serviced from the cache. This property allows HSM to improve not only the number of serviced clients but also the traffic in the network.

Table 1.2: Details of client C1

Links in the path of client C1	Link Bandwidth (Kbps)	Delivered stream rate at C1 (Kbps)	Unused bandwidth (Kbps)		Link busy period (Minutes)	
			PSM	HSM	PSM	HSM
S -1	768	320	448	0	120	50
1-2	512	320	192	0	120	75
2-4	256	320	0	0	150	150
4-10	384	320	64	64	120	120
4-C1	256	320	0	0	150	150

1.8 Organization of the Thesis

The origination of the thesis is as follows: chapter 2 presents about the literature survey in the area of multimedia network and the related work, chapter 3 presents about the functional overview of the HSM and the HSM's components. Chapter 4 talks about HSM based tool architecture and the implementation. We present the performance evaluation of HSM in chapter 5 and the conclusion and the future works in chapter 6.

Chapter 2

Literature Survey and Related Work

In this chapter we discuss about the literature survey and the related work that we have done so far. Our goal for this chapter is to present some areas of the multimedia network and research that has been done so far in this area. We present some literature that is related to our works in section 2.2.

2.1 Literature Survey

Streaming media is a broad area where many researches have been conducted; the aim of the research is to improve the existing system performance as well as to provide the good streaming quality to the clients. In this section we focus on some literature which is relevant to our work and the area of the multimedia network. We have done the literature survey based on some specific areas in the multimedia network, such as caching location in multimedia network, caching strategies in multimedia streaming, transcoding mechanism, the placement of the multimedia object solution in the hybrid data replication and some techniques used for reducing the playout buffering delay. All these techniques aim at improving the number of serviced clients as well as the streaming quality.

2.1.1 Cache Location Problem

In the literature, a cache location problem in [13], studies the problem of where to place the network caches, emphasis is given to caches that are transparent to the clients since they are easier to manage and they require no cooperation from the clients. The goal is to optimize the gain for the system by minimizing the overall traffic in the network and reducing the average delay to the clients.

With the given number of caches in the network, it tries to find the best location of the given caches by placing the caches along the routes from clients to servers and the caches are placed transparently to the servers and clients. An en-route cache intercepts any request that passes through it, and either satisfies the request or forwards the request toward the server along the routing path.

2.1.2 Caching Strategies in Transcoding-Enabled Proxy

The literature in [1], caching strategies in transcoding-enabled proxy system for streaming media distribution network, introduces the caching strategies for streaming media distribution network over Internet, this strategies designed for efficient delivery of rich media web content to heterogeneous network environment and client capabilities. The proxy in this system performs transcoding as well as caching. The proxy transcodes the requested video into an appropriate format and deliver it to the user. One potential advantage of this system is that the content origin servers need not to generate different bit-rate versions, and the heterogeneous clients with various network condition will receive video that are suited for their capabilities, as content adaptation is more appropriately done at network edges.

2.1.3 Proxy Prefix Caching

The literature presented in [17], proxy prefix caching for multimedia streams, presents about the proxy prefix caching to reduce the initial latency at the clients as well as to improve the streaming quality. A prefix caching technique whereby a proxy stores the initial frames of popular clips. Upon receiving a request for the stream, the proxy initiates transmission to the client and simultaneously requests the remaining frames form the server. In addition to hiding the delay, throughput, and loss effects of a weaker service model between the server and the proxy. This prefix caching technique aids the proxy in performing work ahead smoothing into the client play back buffer. By transmitting large frames in advance of each burst, work ahead smoothing substantially reduces the peak and variability of the network resource requirement along the path from the proxy to the client.

Storing the initial frames of each continuous media stream is motivated by the observation that audio and video applications typically experience poor performance due to the unpredictable delay, throughput, and loss properties of the Internet.

2.1.4 Batched Patch Caching

The literature in [5], a batched patch caching for streaming media, is the combination of batch patching at an origin server and prefix/interval caching at an edge server receiving the clients' requests. A regular channel delivers the full video from start to finish while a patching channel delivers only the missing part of the video from the start till the point at which the clients join the regular channel. The client receives both the patch and the ongoing stream and buffers the latter while playing back the former. One the patch is exhausted, the client switches to the buffered regular multicast. The usage of edge servers can reduce the transmission costs by offering a caching opportunity close to the clients. Most of the caching schemes are either based on full caching (the complete video stream is stored at the proxy) or prefix caching. Additionally, caching the prefix hides the startup latency and jitter in the network, and thus allows for user-transparent request batching.

2.1.5 Multicast Technique for True Video-on-Demand Services

The literature presented in [10], multicast technique for true video-on-demand services, is an efficient way to improve the number of serviced clients, as the same streaming object can be sent to differences destinations at the same time, hence, more concurrent clients can serviced. Streaming server can service the clients that access the same streaming object at the same time by using only one streaming flow which goes to different destination clients in the network. This technique is more applicable for distance education program and the cooperative training where the clients from differences places start the course at the same.

2.1.6 Multimedia Object Placement Solution

In this literature [9] presents about the multimedia object placement solution for hybrid transparent data replication, the aim of the placement is to bring the content close to the clients, reducing the initial startup delay and work load as well as the traffic in the network. In this scheme, the differences encoded rate of the multimedia object are placed in the intermediate node according to its popularity, transmission cost and access cost. This paper works on an optimization based approach by developing the cost function [13] to find out the appreciate place for the multimedia object. The cost function consists of three parameters: (i) popularity of the object or the access frequency, (ii) cost for transferring the content, (iii) cost of accessing the object. The idea is simple; first, for each object at each node, find the value of the cost functions, second minimizing the cost function. The placement of the streaming object is decided according to the cost function value, the cost function value must be minimized.

2.1.7 Delay Reduction Technique for Playout Buffering

The literature presented in [2], delay reduction technique for playout buffering provides a delay sensitive solution for playout buffering. This technique works on the delay prediction by using the recorded historical information, and use the historical information to make short-term prediction about network delay with the aim of not reacting too quickly to short-lived delay variations. This allows an application-controlled tradeoff of packet lateness against buffering delay, suitable for applications which demand low delay but can tolerate or conceal a small amount of late packet. In this technique, *aging techniques* is used to improve the effectiveness of the historical information and hence, the delay prediction. The result of the technique gives the smooth streaming flow as the short-lived delay variation is ignored.

2.1.8 Multipath Routing for Video-delivery over Bandwidth-limited

In the literature, multipath routing for video-delivery over bandwidth-limited network [19], provides the solution to improve the performance of the streaming service such as the number of serviced clients and the streaming quality providing to the clients. The idea of multipath routing is that the streaming data is sent through many paths from source to client instead of one path. The streaming object is divided into the segments and sent it to the clients, as it uses more paths to send the data, it can send the data faster, hence, reduce the delay and provide the better stream rate to the client. As different part have difference links bandwidth, so the time for the packet to reach the client may be different, hence packet may be disordered. To solve this problem, they provide one scheduling mechanism at the server to make sure that the packet that has been sent by server to client arrives on time. The scheduler calculates the exact time that each segment of the streaming object should be sent to client and its arrival time. This technique is an efficient way to increase the number of serviced clients and improve the streaming quality.

2.2 Related Work

Most of the research in the area of multimedia dissemination treats delivery of multimedia as real time application [2][18], which can tolerate a small delay for the purpose of solving the delay jitter problem. Mechanisms proposed in the literature focus on minimizing this start up delay [2]. When we consider multimedia delivery over the Internet, there were reasons for using streaming with minimal start up delay: (i) caching or buffering the content was high due to the size of multimedia files: many mechanisms, [17][13][9][7] have been proposed for efficient content management (ii) price of Internet connection was high: many mechanisms have been proposed for effective use of bandwidth, including [4][11][6][3][16][14].

Resource sharing strategies is one of the efficient techniques to increase the number of the serviced clients by exploiting the high skewness in video access patterns. These strategies can be classified into five main categories: Batching [21][23][27], Patching [10], Piggy-backing[25], broadcasting [26], and Interval Caching[22][24][20]. Batching off-loads the storage subsystem and uses efficiently server bandwidth and network resources by accumulating the request for the same videos and serving them together by utilizing the multicast facility. Patching is similar to batching, but it expands the multicast tree dynamically to included new requests, thereby improving resource sharing, but it requires additional bandwidth and buffer space at the client. Piggy-backing offers similar advantages to patching, but it adjusts the playback rate so that the request catches up with a preceding stream, resulting in a lower-quality presentation of the initial part of the requested video. Broadcasting techniques divide each popular video into multiple segments and broadcast each segment periodically. The improved resource sharing here comes at the expense of requiring every high additional bandwidth and buffer space at the client. Interval caching caches interval between successive streams for the same video in the main memory of the server. It does not sacrifice the quality of playback, does not lengthen the waiting time, and does not expect much resource from the client. It has also become more cost-effective with the falling prices of semiconductor memories.

In the recent literature, an optimal chaining scheme proposed in [19] for a Video-on-Demand application uses the concept of collaborative networks. In this mechanism, clients store fragments of streaming content shared between them. This mechanism is not realistic when Internet is used in the dissemination network. Multi-path routing for video delivery over bandwidth-limited networks [20] is another mechanism in which quality of streaming service and the numbers of serviced clients are improved as data is sent faster than one way routing. In this mechanism, links are freed sooner allowing other clients to get the services. But this scheme has a drawback of high computational overhead when the number of client requests increases, as the streaming server performs the scheduling function also.

Mechanisms proposed in [13], explore the caching location problem and propose strategies to reduce the traffic in the network and to improve the efficiency of streaming service, the following techniques including: Prefix Caching, Full object and permanently caching, and object caching based on it popularity [9]. However these techniques lead to high storage requirement given the size of the streaming content. In our proposed scheme, we introduce a caching mechanism with a small overhead; the content is cached in the cache memory at the relay node for a period of time equivalent to the streaming during for the interest of the new request of the same object. We also introduce the possibility to extend the time to live of the content in the cache when new requests are made for the same content.

In the recent times with the drop in the prices of memory and connectivity, and abundance of network bandwidth, clients demand convenience while accessing content. Today's content dissemination networks exhibit heterogeneous characteristics, as networks have combinations of satellite, terrestrial, and Internet links from the source to the clients. Our work focuses on such heterogeneous networks and explores ways of combining different mechanisms for effective and efficient content dissemination, when clients specify their delay tolerance. We present HSM, a simple hybrid streaming mechanism in the context of delay tolerant applications, considering the current trend in multimedia dissemination focusing on users' convenience, to maximize bandwidth utilization and number of serviced clients.

Chapter 3

Functional Overview of HSM

In this chapter, we discuss the functional overview of HSM, the top level algorithm of HSM and the expression used in HSM. We assume that the links have dedicated bandwidth provisioned for the given application and the selected intermediate nodes have the streaming capability. Multicasting is also supported in the given network topology.

3.1 Top-Level Algorithm of HSM

In this section we present the top-level algorithm of HSM to understand how the HSM works and what are the components in the HSM.

Table 3.1: HSM, Top-Level Algorithm

```
Algorithm:
When a client's request arrives /* client specifies minimum rate required and its delay Tolerance */
Determine the stream rate delivered to client considering its delay tolerance
/* Equation (2) from Section 3.3.2 is used */
    If stream rate < client's minimum rate requirement
        Reject request
    Else
        If the link is free
        /*streaming point selection*/
            If the stream rate <= the weakest link in the path from source to region node
                . Use Strategy 1 for streaming point selection /* refer to Section 3.2.2*/
            Else
                . Use Strategy 2 for streaming point selection /* refer to Section 3.2.2*/
            End
            . Transfer the content by using FTP from source to selected streaming point (SP)
              stream from SP to client
            . Find the time to transfer the contents to SP /* Equation (1) from Section 3.3.1 */
        Else /* Link is not free*/
            If the same content is already cached at streaming point
                . Accept request and stream from cache
                . Update TTLC /* refer to Section 3.4.2 */
            Else
                . Reject request
            End
        End
    End
End
```

3.2. Three Phased Operations

In general, HSM contains three main phased operations: (i) the rate calculation: this happens at the first step when client sends the request to server, (ii) the streaming point selection; in this phase, server determines the streaming point for the client based on some strategies, (see section 3.2.2) and (iii) using the new streaming point location, server starts transferring the content by using FTP from source to the selected steaming point and starts streaming the content to the client.

3.2.1. Client's Deliverable Stream Rate Calculation

When client requests for the service, client sends its requirements to the central server. The server then determines the possible deliverable stream rate for the client. If the deliverable stream rate is greater than the client's rate requirement, server sends an accept message to the client, otherwise, a reject message is sent to the client. This first step is very important as the client can specify its requirement consisting of its delay-tolerance – a time at which it wants to play back and its rate' requirement -- the minimum delivery streaming rate that it wants to get. If the server finds that the deliverable rate of the client is less than its minimum requirement, it sends the rejection message to the client without proceeding to the next step.

Example: referring to figure (1.2), assuming the streaming duration is 2 hours and client C1 accesses the streaming object with 256 Kbps as its minimum delivery stream rate requirement and 30 minutes delay tolerance. (S-1-2-4-10-C1) is the path from server to C1. Referring to table (1.2), we get weakest link in the path of C1 is 256 Kbps. By using equation (2), we can calculate the delivery stream rate of C1 as follow:

30*60*256/2*3600 + 256 = 320 Kbps.

3.2.2 Streaming Point Selection

In HSM, a selected relay node serves as the steaming point for all the clients in its sub tree instead of the central server. Thus the streaming point selection strategy is an important part of HSM. In this section, we present two selection strategies based on the following criteria: (i) the position of the streaming point should help to improve the number of serviced clients and /or (ii) the position of the streaming point should help to improve the stream rate for other requests which come from the regions serviced by that streaming point.

3.2.2.1 Selection Strategies

Strategy 1: At a relay node having maximum number of out going links (Improving the number of serviced clients)

Considering a network topology where *all* the links in the path from source to the region node are provisioned with high bandwidth. Suppose the clients in this network have very low bandwidth connections to the region node (the "last mile" problem). In such as case, if the delivered stream rate at a client is less than or equal the weakest link in the backbone, we select the relay node with the most number of out going links as the streaming point. A step-by-step approach to find out the streaming point in strategy 1 is presented below:

```
When client arrives
        Find the weakest link in the path from the source to region node of the client
        Find the client's delivered stream rate
        If the client's delivered stream rate is less than or equal to weakest link
                Choose the node with the maximum outgoing link as the streaming point
        End
  End
```

The reasoning for this strategy is as follows:

- When the stream rate is <= bandwidth of the weakest link in the path from the source to region node, the stream will flow without introducing any delay up to the region node. Hence, any node in the client's path can be chosen as the streaming point.
- However, when the relay node with most out going links is chosen, more clients can be serviced concurrently.

Strategy 2: At any node below the weakest link in the path (Enhancing the stream rates)

In this strategy, any node below the weakest link in the path from source to the region node serving the client is chosen as the streaming point. A step-by-step approach to find out the streaming point in strategy 2 is presented below:

```
When client arrives
    Find the weakest link in the path from the source to region node of the client
    Find the client's delivered stream rate
    If the client's delivered stream rate is greater than the weakest link
        Choose the node below the weakest link as the streaming point
        Apply the strategy 1 to choose the node below the weakest link as a
        streaming point
    End
```

The reasoning for this strategy is as follows:

- The weakest link in a client's path uses up most of the client's delay tolerance.
- When the client's delivered stream rate is greater than the weakest link rate up to the region node, the streaming point is chosen below this weak link. This enables service of other requests from clients in the sub tree made within TTLC to get the stream at that rate, as the stream's flow is not subjected to the weakest link. This strategy may improve the stream rates for the clients.

3.2.2.2 Example to Illustrate these Strategies

Consider a simple network model in Figure (1.2). We present two scenarios to illustrate the streaming strategies. Table 3.2 gives the details of clients requesting the stream.

Illustration of strategy 1*:* Considering the clients in Table 3.2 (strategy1). Client C2 specifies a delay tolerance value of 90 minutes. Stream rate that can be delivered to this client is 224 Kbps. This rate is less than the weakest link (256 Kbps) in the path from source to the region node serving this client. As per strategy 1, we choose the streaming point at node 2. To validate this idea, we study two different cases: first we consider node 4 as the selected streaming point for client C2. Requests of clients C4, C14, C11 arrive at 15 minutes after C2 when the link (S-1-2) is being occupied by C2. Hence requests from C4 and C11 get rejected. Only C14's request can be serviced from the cached content in node 4. Now we consider the same scenario with node 2 as the streaming point. In this case client C4 and C11 can be serviced concurrently. Thus according to strategy 1, when the stream rate of the first client who accesses the stream is less than or equal to the weakest link in the path from the source to region node, the streaming point should be at a node which have the most out going links. This allows serving more client requests concurrently.

Illustration of strategy 2: Let us consider the clients in Table 3.2 (strategy2). Client C1 allows delay tolerance of 90 minutes. The delivered stream rate at C1 is 448 Kbps. This rate is greater than the weakest link in the path from source to the region node serving this client. According to Strategy 2, we choose the streaming point at node 4. All the clients C1, C3, and C13 get 448 Kbps. This is because the content requested by C1 is stored at node 4 and since C3 and C13 are requesting the same content before the TTLC expires, both these clients can be served by node 4. The stream flowing from node 4 is not subject to the weakest link in the path. Note that if we stream from the source or any node above node 4, the delivered rates at C1, C3, and C13 are 448Kbps, 320 Kbps and 320 Kbps respectively, as the weakest link rate is 256 Kbps.

Table 3.2: Details of clients requesting the stream

Clients	Request (Minutes)	Client's requirement (Delay-tolerant, rate)	Service Strategy
C2	0	(90,128)	Strategy 1
C4	15	(30,128)	
C11	15	(30,128)	
C14	15	(60,128)	
C1	0	(90,256)	Strategy 2
C3	100	(30,256)	
C13	110	(30,256)	

3.2.2.3 Remarks

According to the two strategies presented above, there are two possible places for streaming point: (i) at the node, which has the maximum outgoing links and (ii) at the node below the weakest link from the source to region node. Given the random nature of the clients' request and its requirements, it is hard to predict the delivered stream rate at the client, it may happen that some time, delivered stream rate is less than the weakest link and some time it is greater than the weakest link. Hence, in order to cover the two cases, in HSM mechanism we deploy two streaming points for one region, one at the node which has the maximum outgoing links and other at the node below the weakest link. Note that one streaming point may serve more than one region.

In conclusion, we can say that in HSM, central server knows the position of the two streaming points in the region from where the request originated, but what server must determine is which of the two streaming points should be turned on for servicing the client.

3.2.3. Content Transferring and Streaming

Content transferring and streaming is the last step in the HSM after the selection of the streaming point and the delivered stream rate calculation. The streaming content is transferred by using FTP from source to streaming point, where it is temporally cached for further request of the same content.

3.3 Expression used in HSM

In this section we first derive the expression for time to transfer the file from source to streaming point and then the expression for delivered stream rates at clients given their delay tolerance values. The notation using to derive the expression is given in table 3.3.

Table 3.3: Notations

SP	Streaming Point
d_i	Queuing delay at node i
D_q	Total queuing delay in the path
SD	Streaming Duration
T_{ftp}	Time to transfer a file using FTP
T_w	Time to transfer a file across the weakest link
B_i	Link bandwidth at link i
SR_i	Stream rate of client i
L_{min}	Minimum link bandwidth in the path
C_i	Client number i
CS_g	Cache size at streaming point g

3.3.1. Time to transfer file-using FTP

With reference to Figure (3.1), let there be n relay nodes 1, 2, …, n from source S to the streaming point SP. Let B_1, B_2, …, B_{n+1} be the link bandwidths in the path from S to SP. Time to transfer the file across the weakest link (T_w) from S to SP is given by:

$$T_w = \frac{filesize}{Min(B_1, B_2, \dots, B_{n+1})}$$

Let d_1, d_2, …, d_n be the queuing delays at nodes 1, 2,…,n.

Assuming that propagation delay is negligible and there is no other competing traffic. Hence, the total queuing delay is given by:

$$D_q = \sum_{i=1}^{n} d_i$$

Total time to transfer file from S to SP is:

$$T_{ftp} = T_w + D_q \qquad (1)$$

3.3.2 Equation for Deliverable stream rate at a client

In delay-tolerant applications, clients specify two parameters: minimum rate at which they want to receive the stream and their delay tolerance, time they can wait to receive the stream. The deliverable stream rate at client i is given by the expression:

$$SR_i = \frac{CD_i * Lmin}{SD} + Lmin \qquad (2)$$

We derive the expression with reference to Figure (3.1) below:
Let L_{min} be the minimum of link bandwidths L_1, L_2, …, L_{m+1} , in the path between SP and client C_i. When the stream is encoded at L_{min}, client receives it without any loss.
Let CDi be the client delay tolerance of C_i. C_i waits for time CD_i before the play out starts. However, during this waiting time, an amount of data can be streamed to C_i given by: L_{min} *CD_i. The amount of extra data that C_i gets per second is $\frac{Lmin*CD_i}{SD}$ where SD is the total duration of the stream. Thus, the deliverable stream rate at C_i, $\frac{CD_i*Lmin}{SD} + Lmin.$

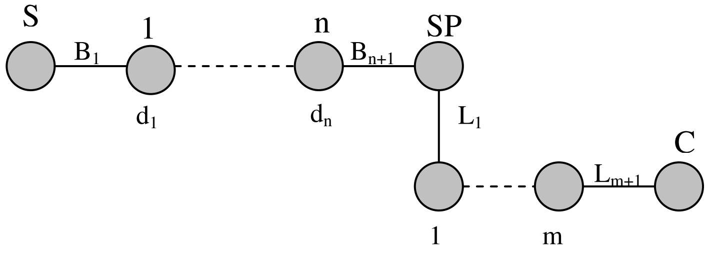

Figure 3.1: Example used for deriving the expression in HSM

3.4 Caching Management at Relay Nodes

In HSM, when the content is transferred from the source to the streaming point, it is temporarily cached. We need to determine the memory requirement for the cache and a mechanism to manage the cache. In Section 3.4.1, we present the cache requirement at a selected streaming point in the network based on the link bandwidth and the number of links connected to that point. We present a simple cache management mechanism with very little overhead in Section 3.4.2.

3.4.1 Memory Requirement at Relay Nodes

We derive the formula for cache size at a streaming point by finding the maximum amount of data that can flow through each sub tree originating at the streaming point. A step-by-step approach is presented below:

- Find the number of sub trees rooted at the streaming point. Let this be N.
- Find the number of regions in each sub tree. Let this number be R.
- For each sub tree
    - Find the weakest link for region j, $Bmin_j$
    - Find the max of weakest link across all regions R, W_i= Max ($Bmin_j$), j=1, ..,R and i= 1, …, N
    - The maximum amount of data that can flow in the sub tree i = W_i*SD, where SD is the stream duration.
- Let FS_{max} be the maximum file size across all content files stored at S. Since clients in the regions can specify delay tolerance, we must find the largest file size that need to be cached. Thus, the cache size for a sub tree is given by:

  $$Max(W_i * SD, FS_{max})$$
- The cache size at streaming point in node g, considering all the sub trees is given by:

$$CS_g = \sum_{i=1}^{N} Max(W_i * SD, FSMax) \qquad (3)$$

Example: (With the reference to figure 2.1)

Let SD= 2hours (7200 seconds)

FS_{max}= 2 GB.

Let node 2 be the chosen streaming point.

We calculate the cache size at node 2 as follows:

Number of sub trees rooted at node 2, N=3

For sub tree 1, Number of regions, R=1; Bmin1= Min (384, 512)=384;

Max($Bmin_1$)= 384.

For sub tree 2, Number of regions, R=2; $Bmin_1$= Min (256, 384) =256;

$Bmin_2$= Min (256, 256) =256; Max ($Bmin_1$, $Bmin_2$)= (256, 256)= 256.

For sub tree 3, Number of regions, R=1; $Bmin_1$= Min (384, 512, 512)=384;

Max ($Bmin_1$) = 384

Cache size at node 2 (CS_2) =

Max (384*7200, 2 GB) + Max (256*7200, 2 GB) + Max (384*7200, 2 GB) = 7.36 Gb

3.4.2 Time to Live of Content (TTLC) in the Cache

In HSM, content is temporarily cached at the chosen streaming point. We use a simple method to determine the value of Time To Live of Content (TTLC) such that the cache management has no additional overheads. Let us consider a client i with delay tolerance CD_i requesting for a stream with duration SD. The client's transmission starts at time = t0+CD_i. The client finishes its transmission at (t0+CD_i+SD). Hence the stream needs to be active for the duration CD_i+SD.

We choose this value as the TTLC for the stream in the cache at the streaming point. When multiple clients access the same stream at the same time, we choose the maximum of the delay tolerance values of the clients in the above expression.

$$TTLC = CD_i + SD \qquad (4)$$

Note that the TTLC can not less than CD_i+SD because if it is, the content will be removed before client finishes the stream. When there is a new request for the same stream before the TTLC expires, it is extended to T_c+ CD_k-(T_c-t_k)+SD, where T_c is the TTLC of the current content, t_k is the time when client k's request arrives and CD_k is the delay tolerance of client k. **Example**: Let client C1 with 30-minute delay tolerance is requesting a stream with 2 hours duration. TTLC for this stream is T = 30 + 120 =150 minutes. Let a new request for the same stream comes from client C2 at t=90. C2's delay tolerance is 90 minutes. The extended value of TTLC for the stream is: 150 + 90 - (150 - 90) + 120. Thus, the content is alive till t=300 minutes.

Chapter 4

HSM based Tool Architecture and Implementation

This chapter, we discuss about the HSM's architecture and the main components of HSM. We also present the HSM Simulator which is implemented in the Matlab for evaluating the general performance of HSM and PSM.

4.1 HSM's Architecture

In HSM, there are three main components as presented in the Chapter 3: (i) the deliverable stream rate calculation, (ii) streaming point selection and (iii) the content transferring and streaming. These three phased operations form the core of HSM. We divide HSM's architecture into three modules: (i) inputs module, (ii) operation module that contains all the three phased operations mentioned above and (iii) outputs module.

4.1.1 HSM's Components

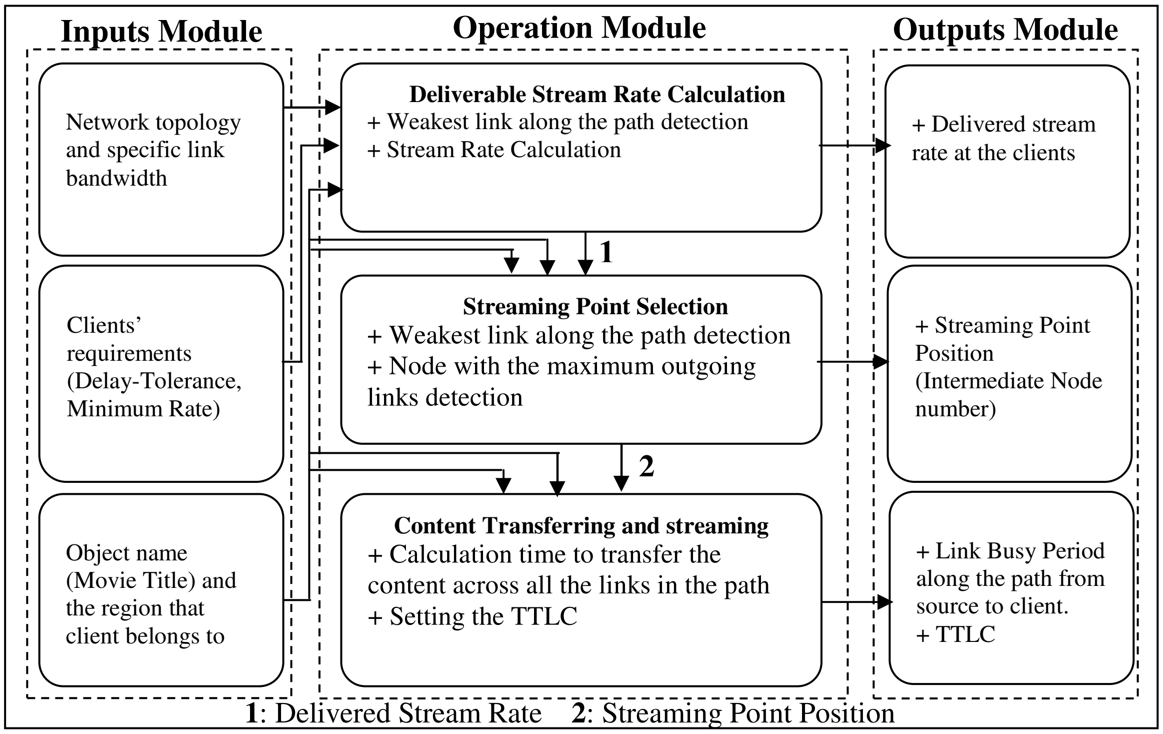
Inputs Module
Operation Module
Outputs Module
Network topology and specific link bandwidth
Clients' requirements (Delay-Tolerance, Minimum Rate)
Object name (Movie Title) and the region that client belongs to
Deliverable Stream Rate Calculation
+ Weakest link along the path detection
+ Stream Rate Calculation
Streaming Point Selection
+ Weakest link along the path detection
+ Node with the maximum outgoing links detection
Content Transferring and streaming
+ Calculation time to transfer the content across all the links in the path
+ Setting the TTLC
+ Delivered stream rate at the clients
+ Streaming Point Position (Intermediate Node number)
+ Link Busy Period along the path from source to client.
+ TTLC
1
2
1: Delivered Stream Rate 2: Streaming Point Position

Figure 4.1: Top Level HSM's Architecture

4.1.1.1 Deliverable Stream Rate Calculation Module

This module is responsible for calculating the deliverable stream rate at the client given the client's requirements and network topology. When the request arrives, server determines the weakest link along the path from source to client. Given the weakest link along the path and the client's delay tolerance, server is able to determine the delivered stream rate at the client by applying the formula in section 3.3.2. Next step is to find out the position of the streaming point for the given request.

4.1.1.2 Streaming Point Selection Module

This module is responsible for determining the position of the streaming point. This module requires the delivered stream rate and the path from source to client as inputs. To determine which node along the path from source to client, is the streaming point, first server needs to figure out what is the weakest link along the path from source to region node, and then apply the streaming point selection strategies presented in section 3.2.2.

4.1.1.3 Content Transferring and Streaming Module

The last module is responsible for content downloading and streaming. Given the delivered stream rate and the position of the streaming point; server then starts transferring the content to the streaming point by using FTP and starts streaming from that point to the client. The transferred content is temporally cached at that streaming point and its TTLC is also set here.

4.1.2 Inputs Module

There are three major inputs for HSM:

- ***Network Topology*** - In HSM, server needs to know the characteristic of all links in the entire network. We assume that the given network topology is a tree based network, meaning, there is no cycle in the network.
- ***Clients' Requirements*** – clients' requirements consists of two parameters: (i) delay-tolerance-- the maximum waiting time acceptable to the client and (ii) rate requirement -- the minimum stream rate acceptable to the client.
- ***The object name and the region node*** - the object name (example the movie title requesting by the client) is given by the client while requesting the stream, the region node can be extracted from the path of the client in the given network topology.

4.1.3 Outputs Module

There are three output parameters: (i) the delivered stream rate at the client, (ii) the streaming point position and (iii) the link busy period along the path from source to the requesting client. All these three output parameters are monitored by the central server. Central sever has a database that stores: (i) the current state of the links, (ii) nodes which are acting as the streaming point and (iii) the objects that are being cached at each streaming point. This information is important for serving the further requests.

4.1.4 HSM's Pseudo Code

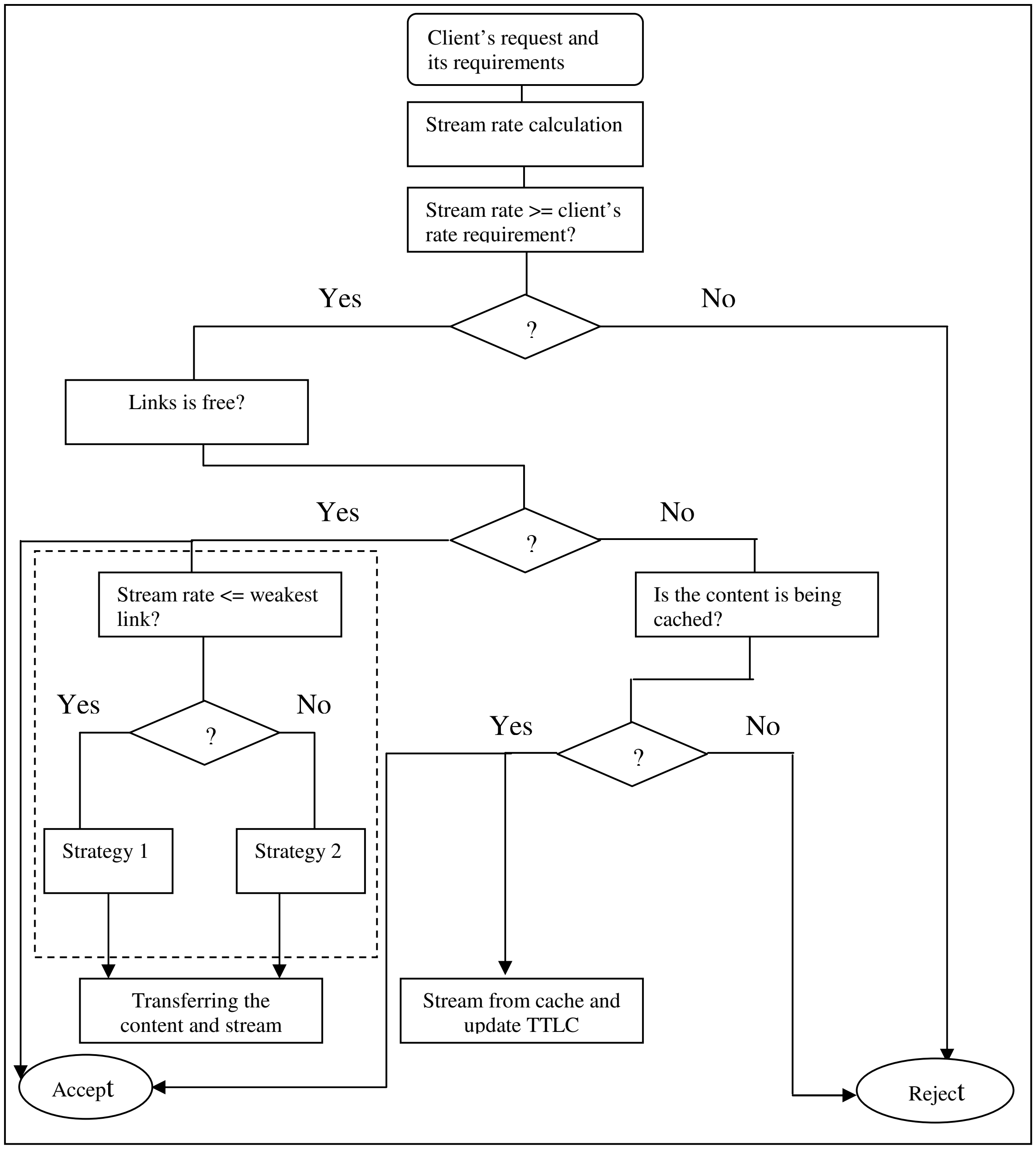
Client's request and its requirements
Stream rate calculation
Stream rate >= client's rate requirement?
Yes
No
?
Links is free?
Yes
No
?
Stream rate <= weakest link?
Is the content is being cached?
Yes
No
?
Yes
No
?
Strategy 1
Strategy 2
Transferring the content and stream
Stream from cache and update TTLC
Accept
Reject

Figure 4.2: HSM's Pseudo code

4.2 Simulator Implemented in Matlab

In this section, we present the design of the simulator to evaluate the performance of PSM and HSM. Our goal is to compare the performance of HSM to PSM in terms of (i) number of serviced client and (ii) the delivered stream rate improvement. The implemented simulator consists of two functionalities: (i) performs as PSM like simulator and (ii) HSM like simulator. The implementation of PSM is for the purpose of the comparison between PSM and HSM. Figure 4.2 shows the functional overview and the components of the simulator that we have implemented in Matlab and the details of each component are presented below.

- ***Static Network Topology*** – refers to the existing network topology given by the user, for example Gnutella network topology or any real network topology in the Internet nowadays.
- ***Network Topology Generator*** – This module allows user to generate the network topology according to given inputs, such the number of nodes in the network, the depth of the network and the bandwidth of the links in the network. It gives the completely random network topology for the experimentation.
- ***PSM Module*** – This module performs as the PSM simulator and it is responsible for finding out the number of serviced clients and the percentage of stream rate improvement for a given client arrival pattern.
- ***HSM Module*** – This module performs as the HSM simulator and it is responsible for finding out the number of serviced clients and the percentage of stream rate improvement for a given client arrival pattern.
- ***Outputs*** – the outputs of the simulator are the number of serviced clients and the percentage of stream rate improvement of clients.

4.2.1 PSM Simulator's Architecture

- ***Request Arrival Rate*** – Number of the client requests per second.
- ***Observation period* – P**eriod over which the clients are monitored.
- ***Delay-Tolerant (Interval)*** – This is the interval of the client's delay tolerance value. For example if the interval value is [15 minutes – 120 minutes], clients' delay tolerance is chosen between 15 to 120 minutes randomly.

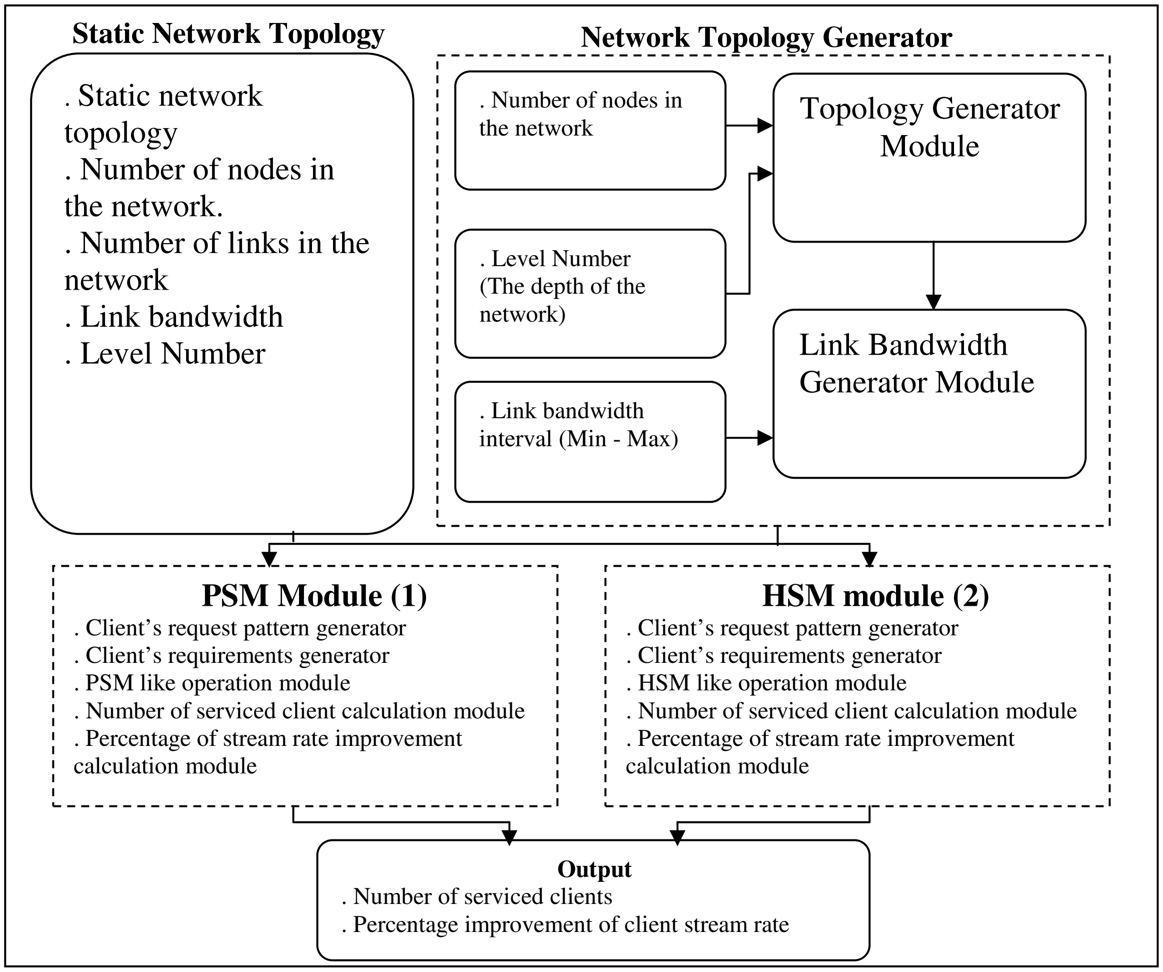
Static Network Topology
. Static network topology
. Number of nodes in the network.
. Number of links in the network
. Link bandwidth
. Level Number
Network Topology Generator
. Number of nodes in the network
. Level Number (The depth of the network)
. Link bandwidth interval (Min - Max)
Topology Generator Module
Link Bandwidth Generator Module
PSM Module (1)
. Client's request pattern generator
. Client's requirements generator
. PSM like operation module
. Number of serviced client calculation module
. Percentage of stream rate improvement calculation module
HSM module (2)
. Client's request pattern generator
. Client's requirements generator
. HSM like operation module
. Number of serviced client calculation module
. Percentage of stream rate improvement calculation module
Output
. Number of serviced clients
. Percentage improvement of client stream rate

Figure 4.3: Simulator's Architecture

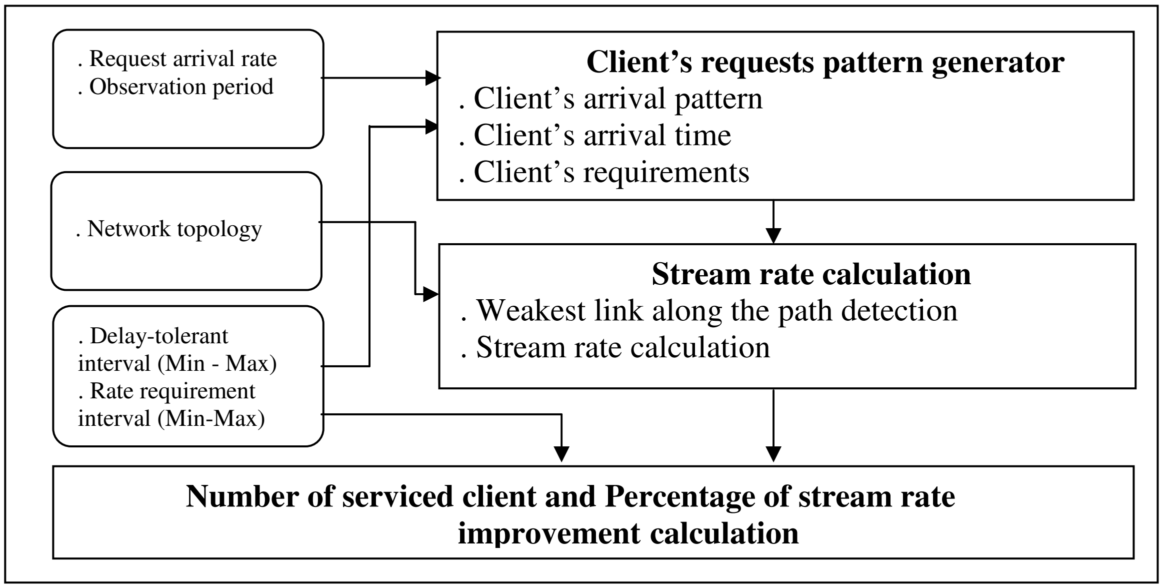
. Request arrival rate
. Observation period
. Network topology
. Delay-tolerant interval (Min - Max)
. Rate requirement interval (Min-Max)
Client's requests pattern generator
. Client's arrival pattern
. Client's arrival time
. Client's requirements
Stream rate calculation
. Weakest link along the path detection
. Stream rate calculation
Number of serviced client and Percentage of stream rate improvement calculation

Figure 4.4: PSM Module (1)

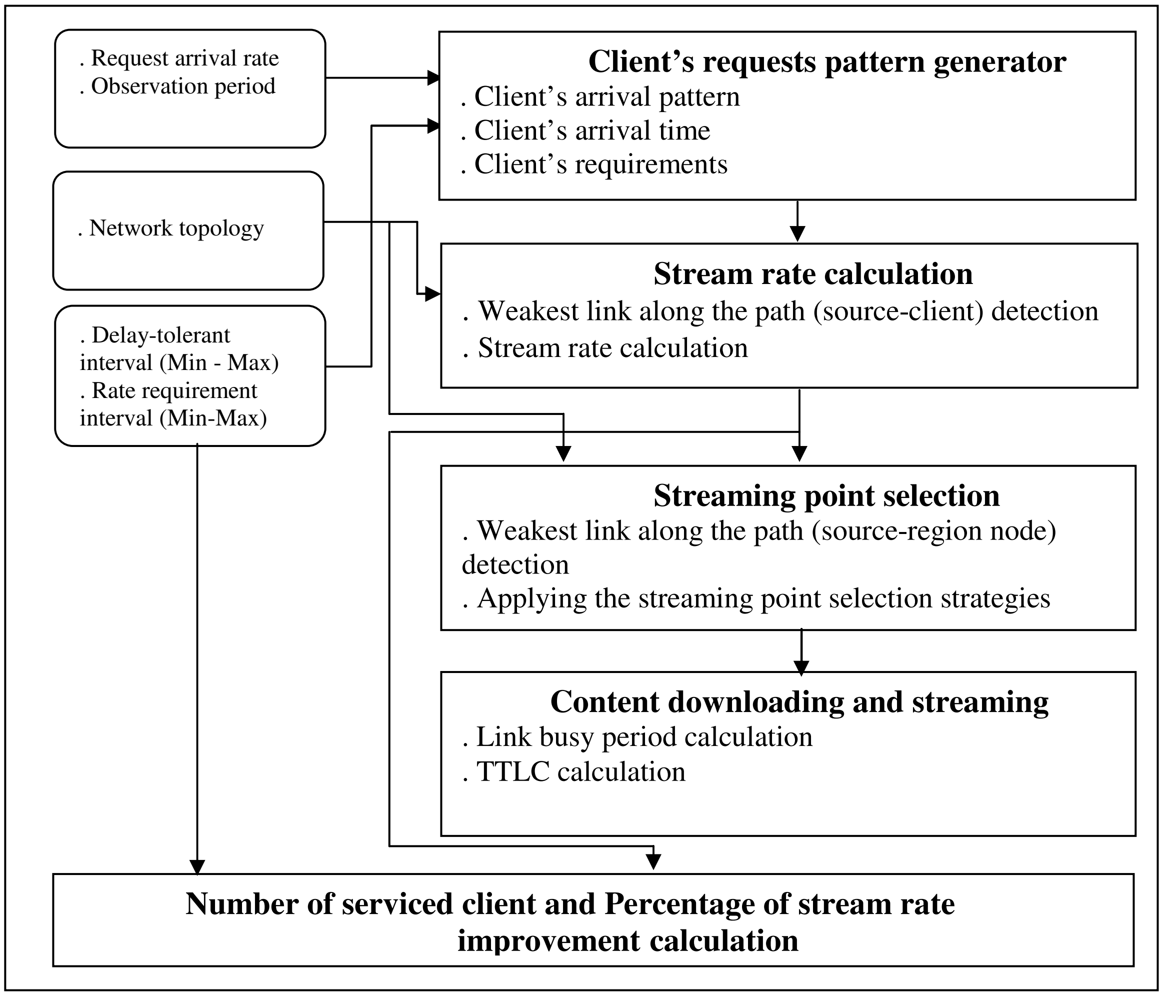
. Request arrival rate
. Observation period
. Network topology
. Delay-tolerant interval (Min - Max)
. Rate requirement interval (Min-Max)
Client's requests pattern generator
. Client's arrival pattern
. Client's arrival time
. Client's requirements
Stream rate calculation
. Weakest link along the path (source-client) detection
. Stream rate calculation
Streaming point selection
. Weakest link along the path (source-region node) detection
. Applying the streaming point selection strategies
Content downloading and streaming
. Link busy period calculation
. TTLC calculation
Number of serviced client and Percentage of stream rate improvement calculation

Figure 4.5: HSM Module (2)

- ***Rate requirement (Interval)*** – This is the interval of the client's rate requirement value. For example if the interval value is [128 Kbps – 256 Kbps], clients' rate requirement is chosen between 128 to 256 Kbps randomly.
- ***Client's requests pattern generator*** – This module generates the clients' request pattern with the given inputs such as the client's delay interval, rate requirement interval. The output parameters of this module are: (i) the client's arrival time, (ii) the client' requirements and (iii) the region at which the client belongs to. All the values are randomly distributed
- ***Stream Rate Calculation*** – This module calculates the delivered stream rate at the client given the network topology, client's requirements and the requested object.
- ***Number of serviced clients and percentage improvement of the stream rate*** – This module is responsible for calculating the number of serviced clients and their stream rate improvement within the observation period.

4.2.2 HSM Simulator's Architecture

- ***Streaming Point Selection*** – This module is responsible for figuring out which node should be acting as the streaming point for the client in the particular region in the network. The output of this module is the information of the node at which the stream must start from.
- ***Content Transferring and Streaming*** – This module is responsible for calculating the link busy period in the path from the source to client, this information is necessary for the central server to monitor the links status in the network. It also calculates the TTLC of all the content that are being cached at the intermediate nodes.

4.3 Matlab Codes

In this section, we present the snapshot of the Matlab Code that we have developed so far. The Simulator implemented in Matlab consists of five differences components, (i) Network Topology Generator, (ii) Client Request Pattern Generator, (iii) PSM module and (iv) HSM module. For more details see the appendix.

Chapter 5

Performance Evaluation of HSM

In this section we present the results of simulations evaluating the performance of PSM and HSM, using Matlab. Our objective is to compare the performance of the two mechanisms under various network topologies and client requirements. The following performance metrics are used: (i) the number of serviced clients, and (ii) percentage improvement of client's stream rate as compared with its minimum rate requirement.

We define the parameters used in the simulations in Section 5.1 and present details of our experiments in Section 5.2. In Section 5.4, we discuss the results of our simulations.

5.1 Simulation Parameters and Scenario

The following parameters remain same across all our experiments: (i) Multimedia play out duration is set to 2 hours, (ii) Without loss of generality, queuing delay and propagation delay are set to zero, (iii) Period over which client arrivals are monitored, termed *observation period*, is set to 4 hours and (iv) Arrival rate of the clients' requests is varied from 1 to 30 per minute.

Performance of HM depends on the network topology, links characteristics, and client requirements. In order to study the impact of these parameters on the performance of HSM, we have taken 100 differences topologies that fall into two classes:

- The first 50 topologies are in Class 1; these topologies have high bandwidth at the links from source to region node. The bandwidths are chosen randomly from the range (256 Kbps – 768 Kbps).
- Next 50 topologies are categorized into Class 2 topologies that have low bandwidth (weak links) in the upper part of the network from source to region node. The bandwidths are chosen randomly from the range (128 Kbps – 256 Kbps).

All topologies have total number of nodes in the range 100 to 500, where the number of nodes is selected randomly. For each topology, we have done 100 times simulation with different client request time, and the result of each topology is an average across the results given by the 100 times simulation. The final result is the average of the 50 results represented the 50 topologies mentioned above. This scenario is repeated for both the classes, class 1 and class 2 networks.

5.2 Details of Experiments

In the first set of experiments, we use Class 1 topologies to evaluate the performance of PSM and HSM.

5.2.1 Number of Serviced Clients VS. Client's Delivered Stream Rate

***Experiments 1 (Class 1)*:** Set the clients' minimum rate to 128 Kbps and delay tolerance values to 30 minutes for all clients. We find the number of serviced clients and the percentage improvement of clients' stream rates under PSM and HSM. The result given in figure (5.1) and (5.2) is an average across all the results given by the 50 network topologies and 100 times simulation for each topology.

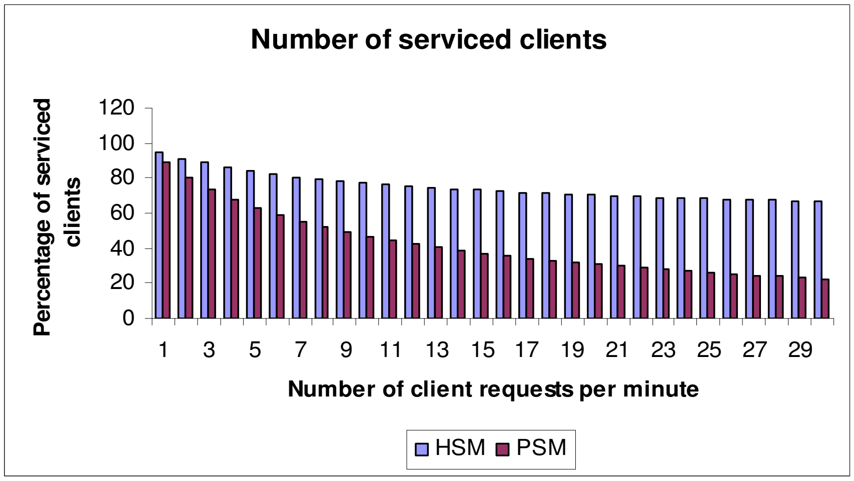

Figure 5.1: Number of serviced clients with identical client requirements (class 1)

Figure (5.1) shows that when the request rate increases, the number of serviced clients decreases for both the mechanisms. This is as expected. However, the decrease is more pronounced in PSM compared to HSM. While the number of clients gradually decreases in

HSM, it drops more rapidly in PSM. Also, the difference between the number of serviced clients in PSM and HSM keeps widening as the number of requests increases.

The results given in figure (5.2) show that whenever the number of client requests increases, the percentage improvement decreases for both the mechanisms. From this graph, we see that PSM services clients with better stream rates compared with HSM. However, PSM rejects 78% of client requests compared with 30 % of requests rejected by HSM. In order to observe both the parameters – number of serviced clients and stream rate improvement at the clients - we present Figure (5.3). In this figure, X-axis represents the number of client requests per minute and Y-axis (on the left) represents the percentage of clients serviced and Y-axis (on the right) represents percentage of stream rate improvement.

5.2.2 Impact of Client's Delay Requirement on System Performance

In the next set of experiments, we use Class 1 topologies to evaluate the impact of clients' delay tolerance on the number of serviced clients using PSM and HSM.

Experiments 2 (Class 1)**:** In this set of experiments, we use Class 1 topologies to evaluate the impact of clients' delay tolerance on the number of serviced clients using PSM and HSM. We set the clients' minimum rate to 128 Kbps. For each network topology, we run four experiments 100 times each: for each experiment the delay tolerance values of all clients are equal. The values of delay tolerance chosen for experiments 1-4 are 30, 60, 90, and 120 minutes respectively. This result is an average across all the results given by the 50 network topologies and 100 times simulation for each topology.

Results in Figure (5.4) demonstrate that as clients' delay tolerance increases, the performance of HSM gets better. When the client delay tolerance is equal to the streaming duration, HSM services nearly 100% of the clients' requests. In the case of PSM, as shown in figure (5.5), client delay tolerance has very little effect on the number of client requests serviced. This show that HSM performs better and better than PSM when the client' delay tolerance increases.

Experiments 3 (Class 2): In this set of experiments, we use Class 2 topologies to evaluate the performance of PSM and HSM. We set the clients' minimum rate to 128 Kbps and delay tolerance values to 15 minutes for all the clients. We find the number of serviced clients and the percentage improvement of clients' stream rates under PSM and HSM. . The result given in figure (5.6), is an average across all the results given by the 50 network topologies and 100 times simulation for each topology.

Figure (5.6) displays the number of service client and the percentage of stream rate improvement under PSM and PSM. Results in Figure (5.6) demonstrate that, in class 2 network topology, HSM still performs better than PSM in term of number of serviced clients, but only marginally. It is also happened to client stream rate, where PSM performs slightly better than HSM. In general HSM performs well for class 1 topology.

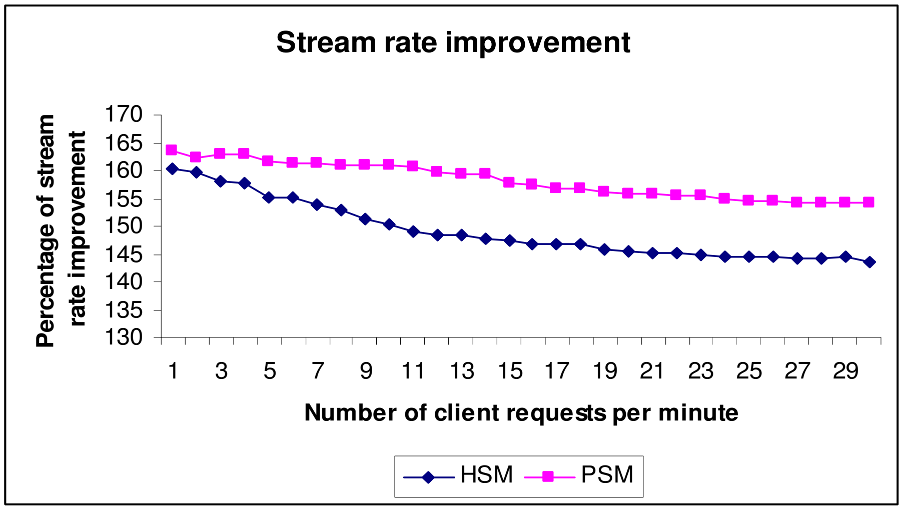

Figure5.2: Percentage of stream rate improvement with identical client requirements (Class 1)

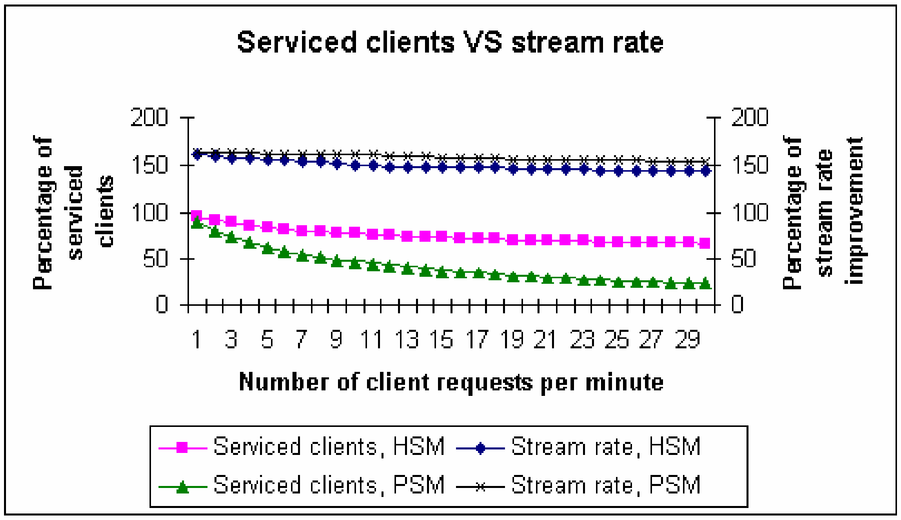

Figure 5.3: Percentage of serviced clients vs. percentage of stream rate improvement

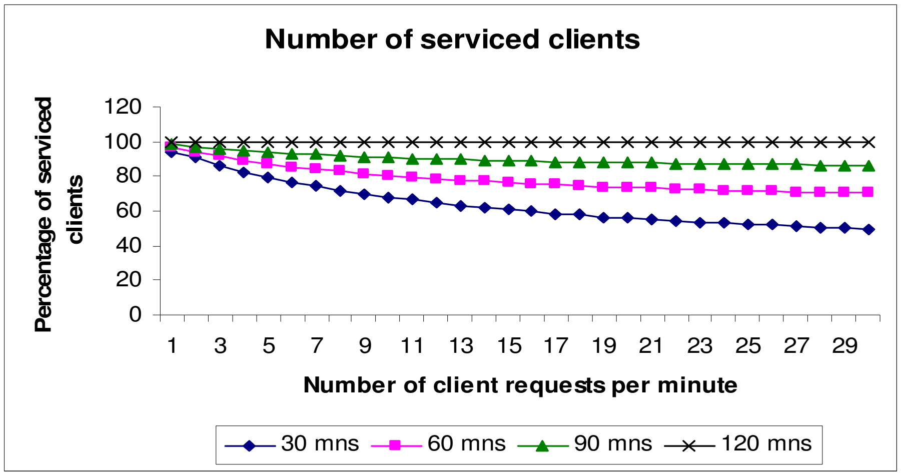

Figure 5.4: Impact of client delay tolerance values on HSM

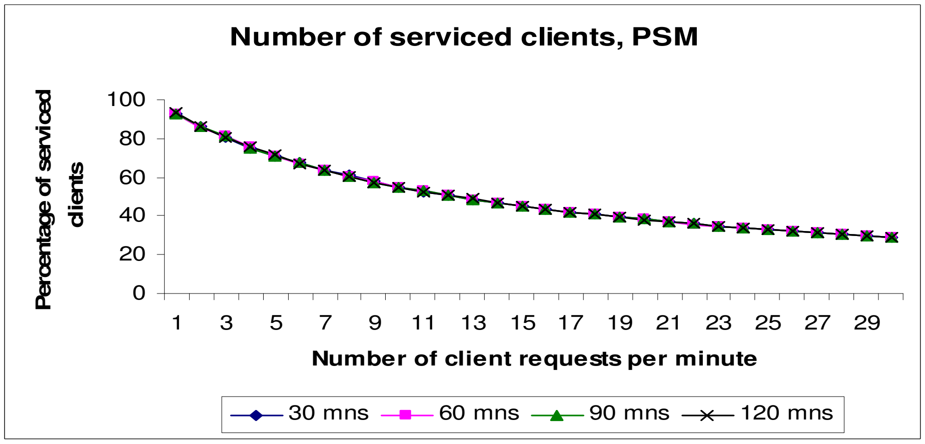

Figure 5.5: Impact of client delay tolerance values on PSM

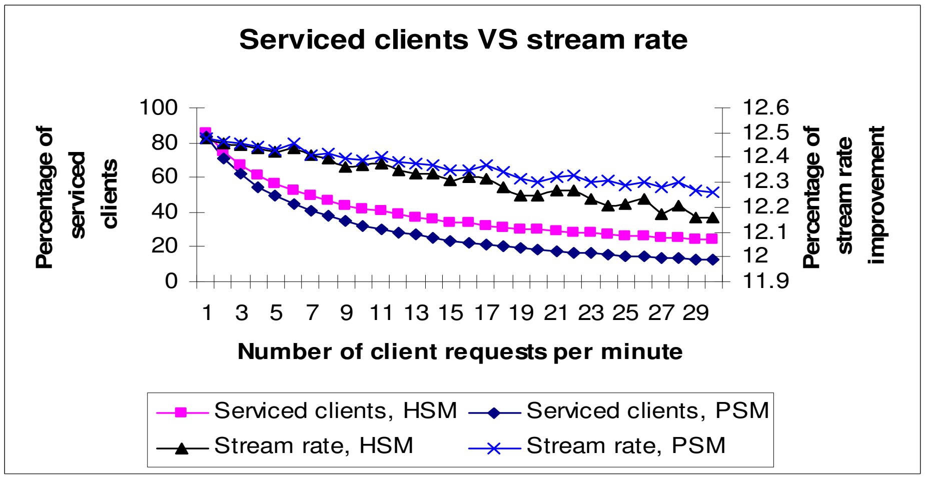

Figure 5.6: Percentage of serviced clients vs. percentage of stream rate improvement (Class 2)

5.3 A Case Study: Gnutella Peer Network

In this section, we present a case study on the Gnutella Peer Network []. We simplify the original network topology (figure 5.7) to a tree-based network by removing cycles in the topology. Our approximated Gnutella Peer Network backbone contains 510 nodes. In this simulation, we set the clients' minimum rate to 128 Kbps and delay tolerance values to 30 minutes for all clients. We observe the number of serviced clients and the percentage of clients' stream rates improvement under PSM and HSM. The result given in figure (5.8) is an average across the 100 times simulation with different client request time for the Gnutella Peer network.

Figure (5.8) displays the number of serviced clients and the percentage of stream rate improvement. With the given results, we observe that HSM performs better than PSM in term of number of serviced clients, but less in the percentage of stream rate improvement. Note that when the number of client requests reaches 30 per minute, HSM performs 20 percent better than PSM. While PSM rejects more client requests, it provides stream rate that are on average 5 percent better than HSM.

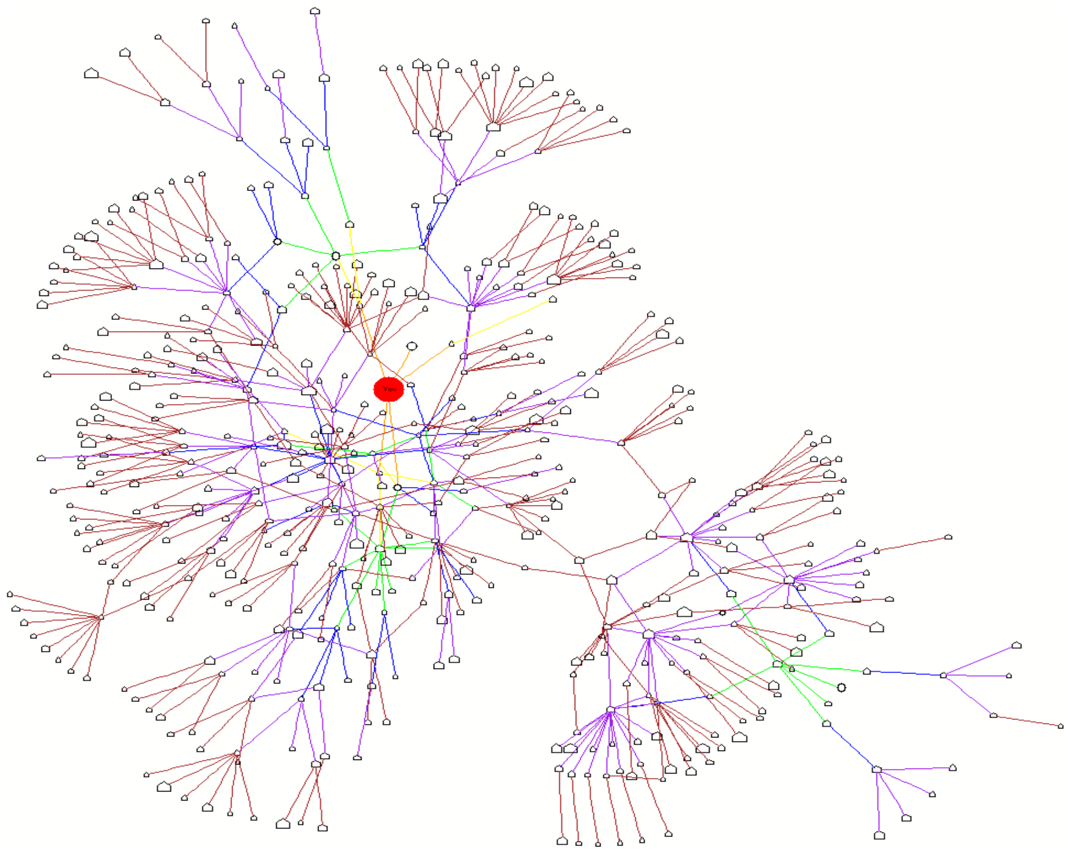

Figure 5.7: Gnutella Peer Network

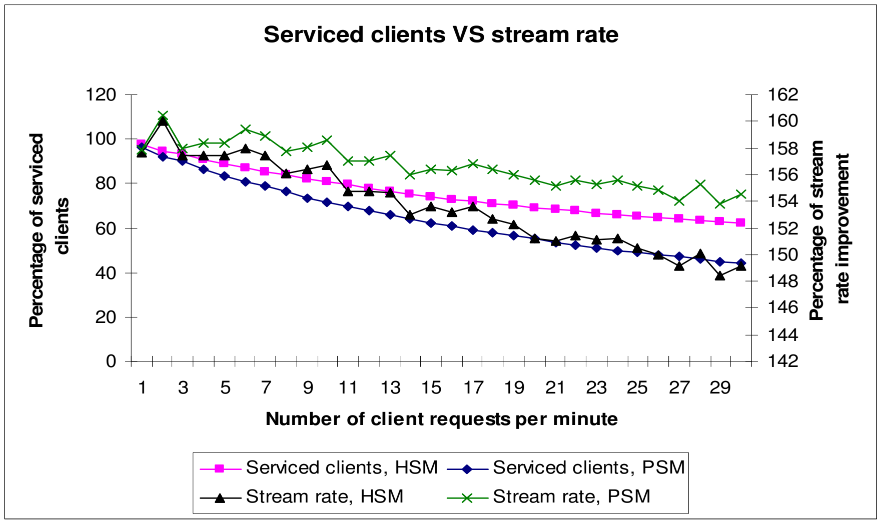

Figure 5.8: Percentage of serviced clients vs. percentage of stream rate improvement (Gnutella)

5.4 Analysis of Results

Performance of HSM depends on the following factors: (i) network topology with specific link bandwidths, and (ii) clients' requirements. We observe that for Class 1 topologies where the link bandwidths are provisioned such that the upper links from the source to the regional nodes have high bandwidth, HSM is a better scheme as the available bandwidth can be better utilized with this mechanism. In class 2 topologies links from the source to the region nodes have low bandwidths. In this case, using FTP to transfer the file to a relay node does not provide any advantage, as the time for transferring the file is same even when streaming server is placed at the source. The only advantage of HSM is that by choosing a streaming point appropriately, requests from clients for the same content can be serviced from the cached contents. Thus, we observe only marginal improvement in the number of clients serviced with such topologies.

To summarize, HSM works well with Class 1 topologies because of the impact of FTP property used in HSM. If the dissemination network falls in Class 2 category, PSM is preferred as the costs involved in enabling relay nodes with streaming capability may outweigh the benefits.

5.5 Costs-Benefit Analysis

In this section we present a simple analysis the trade-off between cost of putting streaming capability at relay nodes and the benefit from improved client services. Our goal is to find the break-even point, time taken for the CSP to cover the cost of putting streaming capability at the relay nodes by the revenue from improved client services.

Cost of providing streaming capability at the relay nodes is given by: N*C, where N is the number of relay nodes with the streaming capability and C is the cost per streaming server. Let N_H be the number of serviced clients per unit time when HSM is used. Let N_P be the number of serviced clients per unit time when PSM is used. Note that when PSM is used only one streaming server is placed at the source. By using HSM, the increase in number of clients serviced is given by (N_H-N_P). The additional revenue the CSP makes by servicing these clients is given by P*(N_H-N_P), where P is the price a client pays for the service.

The break-even point in *unit time* is given by: $Y= N*C/P*(N_H-N_P)$

Example: Consider a network with 20 selected relay nodes with streaming capability. If we use PSM, we serve 200 out of 400 clients per day using only one central server. If we use HSM, 300 out of 400 clients are served per day. Suppose one streaming server costs $2000 and a client pays $4 for the service,

Y=20*2000/4*(400-300) = 100 days.

Chapter 6

Conclusion

Leveraging clients' delay tolerance, better stream rates can be delivered to clients, even when links are constrained in their path from the source. Typically in a content dissemination network controlled by a CSP, weak links are at the edge of the network closer to the clients. By using a combination of FTP and streaming mechanisms, provisioned links in the CSP's backbone can be fully utilized, serving more client requests when compared to a centralized server handling all the streaming requests. HSM, the proposed hybrid streaming mechanism uses this idea to improve the performance and hence the revenue for a CSP.

We have shown that by intelligently choosing an appropriate relay node as streaming point, on the average 40% more requests can be serviced using HSM as compared with PSM. In HSM, the transferred content is cached temporarily at the streaming point, used to service future requests for the same content. This feature further enhances the performance of HSM when class 1 topologies with highly provisioned backbone are used in the simulations. With class 2 topologies having weak links in the backbone, we observe only marginal improvement in the performance of HSM resulting from additional requests serviced from the caches at streaming points. The future work is to extend HSM from static to dynamic link bandwidth assumption. Next section, we present HSM extensions.

6.1 HSM Extensions

In this section, we discuss HSM extensions to improve the delivered stream rates at clients as well as to minimize the number of the nodes with the streaming capacity. For instance, HSM works with the dedicated link bandwidth assumption; meaning, bandwidths are stable whenever the accepted request has been sent to client. This assumption is not very suitable with the Internet scenario. For the extensions of HSM, we propose following three additional features for HSM to improve the performance of HSM as well as to minimize the resource utilization such as streaming server and link bandwidth utilization: (i) Admission Control, (ii) extension of static bandwidth to dynamic bandwidth and (iii) optimal placement of streaming point.

6.1.1 Admission Control

In this section, we present about the functionality and the purpose of the admission control in HSM. Our proposed HSM is working with First Come First Serve (FCFS) scenario, where the first client requests the stream, gets its best streaming rate and client coming later must adjust the stream quality with the first request. FCFS is not an efficient way to provide the good quality of the stream to client because clients in the network have different link bandwidth and requirements; hence. We give a simple example below to illustrate the disadvantages of FCFS in HSM and to illustrate the need of the admission control.

Considering the scenario having 10 clients' request per 10 minutes and let say those 10 clients are C1, C2, …, 10 get the delivered stream rate at 128, 384, 384, 448, 512, 384, 449, 512, 384, 512 Kbps respectively. Assuming C1 to C10 request the stream at t=0, t=1, t=2, …, t=10 minutes respectively. Let consider the above arrival pattern, for instant, HSM performs as follow:

First C1 is accepted as it arrives first, and then the immediate accepted message is sent to C1. In this case C2 to C10 have to follow C1's stream rate, C2 to C10 gets 128 Kbps. However C2 to C10 can get higher rate if C1's request gets rejected. Note that Central server can not initiate the same stream object with different flows and encoded rate to the clients, because while serving one client, links are busy, it is not possible to serve others that are sharing the links with the client being serviced.

To rectify the above problem, we propose an additional feature for HSM. The idea is as follows: First we determine the admission control window; this window is a time for which the central server waits for collecting the client requests. Central server does not send the accepted or rejected message to clients till the end of the admission control time. In this case, client has to wait for sometime in order to get the response from the server. In the worst case, client will wait for the period equivalent to the admission control window size and in the best case; it will get the reply immediately, if it requests the stream just a moment before the admission control time expires. When the admission control time expires, server starts counting the number of requests and their delivered stream rate and then it applies some statistical analysis to find out the common delivered stream rate for all the requested clients. The median and mode are used. A step-by-step approach to figure the common delivered stream rate is as follows: find out what is the median and mode value of the given sample (delivered stream rate). If the mode value is greater than 50% (this is a threshold value, it can be any value determine by CSP) than the common delivered stream should be applied with the majority. If not, the median is applied to find out what is the common rate for all those clients. We illustrate the approach by using the example below.

Example, let take the same arrival pattern given in the above example, the same client C1 to C10, with the delivered stream rate 128, 384, 384, 448, 512, 384, 449, 512, 384, 512 Kpbs respectively. Assuming that, the admission control window size is 10 minutes. We find the median and mode for the given 10 elements (Delivered stream rate), mode is 4 (384Kbps) and median is 416 Kbps. As the mode value is 40%, which is less than 50%, so the common delivered stream rate at the client should be applied to median, which is 416Kbps. Hence, the common delivered rate for all the 10 clients is 416 Kbps.

6.1.2 Extending from Static to Dynamic Links Bandwidth

For instance, HSM is working only with the assumption of the static link bandwidth or dedicated links bandwidth. So in this section we discuss about the idea of converting HSM with static link bandwidth to HSM with dynamic link band width. First we assume that the link bandwidth vary in the interval L = [Bmin, Bmax], it has never exceeded the maximum and never dropped below the minimum. Let Bt is the current available link bandwidth when client requests the stream. The idea is that instead of serving the client with the current available link bandwidth, we serve them with the link bandwidth equivalent to (Bmin + Bt)/2. This prevents the interruption of the stream when the link goes down. This allows us to have the additional data to adjust to the situation when the links go down. This scheme is working under the assumption, the link bandwidth decrease linearly, not exponentially; otherwise the amount of extra data may not sufficient to adjust the lost due to the drastically drop of the link rate.

6.1.3 Optimal Placement of the Streaming Point

For instance, we assume that there are two possible streaming points in each region, one at the node with the maximum outgoing links and other at the node below the weakest link along the path from source to region node. The central server chooses one of them by performing the selection strategy presented in chapter 3. The idea of the optimal placement is to minimize the number of the streaming point in the entire network by using the statistical analysis based on the history of the clients' request from each region, for example, in a given network topology with 100 regions. For some regions, we know that the client's link is very low as compared with the backbone network and base on the history, client's delivered rate has never exceeded the weakest link in the backbone network. In this case, the streaming point below the weakest link may not be required. Hence, we can take it out. By using this mechanism, we can minimize the number of streaming point in the entire network.

Bibliography

[1] Bo Shen, Sung-Ju Lee and Sujoy Basu. Caching strategies in transcoding-enabled proxy systems for streaming media distribution networks. IEEE Transaction on Multimedia, 6(2):375{386, June 2000.

[2] Cormac J. Sreenan, Jyh-Cheng Chen, Member, IEEE, Prathima Agrawal and B. Narendran. Delay reduction techniques for playout buffering. IEEE Transaction on Multimedia, 2(2):88{97, June 2000.

[3] Danjue Li, Chen-Nee Chuah, Gene Cheung and S. J. Ben Yoo. Yoo. Muvis: Multi-source video streaming service over wlans. KIC, 2003.

[4] W. Z. Y.-Q. Z. a. J. M. P. Depeng Wu, Yiwei Tomas Hou. Dapeng Wu, Yiwei Thomas Hou, Wenwu Zhu, Ya-Qin Zhang and Jon M. Peha. IEEE Transaction on circuit and system for video technology, 11(3):282{300, March 2001.

[5] P. Frossard and O. Verscheure. Batched patch caching for streaming media. IEEE COMMUNICATION LETTER, 6(4):159{161, April 2002.

[6] Guan-Ming Su and Min Wu. Efficient bandwidth resource allocation for low-delay multi-user video streaming. IEEE Transaction for Circuits and Systems for Video Technology, 15(9):1124{1137, September 2005.

[7] Jianliang Xu, Bo Li and Dik Lun Lee. Placement problem for transparent data replication proxy service. IEEE Journal on Selected Areas in Communications, 51(6):1383{1398, 2002.

[8] X. C. Jiangchuan Liu and J. Xu. Proxy cache management for grained scalable video streaming. IEEE IFOCOM, 2004.

[9] Keqiu Li , Hong Shen, Francis Y. L. Chin and Liusheng Huang. A multimedia object placement solution for hybrid transparent data replication. Japan Advanced Institute of Science and Technology and University of Hong Kong, 2005.

[10] S. S. Kien. Hua, Ying Cai. Multicast technique for true video-on-demand services. ACM Multimedia, pages 191-200, 1998.

[11] D. K. Mohamed, M. Hefeeda, Bharat Bhargava. A hybrid achitecture for cost-effective on-demand media streaming. Department of Computer Science, Purdue University, West Lafayette, October 2003.

[12] G. of Cyberspace Directory. http://www.cybergeography.org/.

[13] Danny Raz, P.Krishnan and Yuval Shavitt. Caching location problem. IEEE/ACM Transactions Networking, 8(5):795{825, October 2000.

[14] W. Z. Qian Zhang and Y.-Q. Zhang. Resource allocation for multimedia streaming over the Internet. IEEE Transaction on Multimedia, 3(3):339{355, September 2001.

[15] Saraswathi Krihivasan and Sridar Iyer. Enhancing QoS for Delay-tolerant Mutimedia Applications: Resource utilization and Scheduling from a Service Provider's Perspective.

[16] Y.-S. M. Sang-Ho Lee, Kyu-Young Whang and I.-Y. Song. Dynamic buffer allocation in video-on-demand systems. IEEE Transactions on knowledge and data engineering, 15(6):1535{1551, 2003.

[17] J. R. Subhabrata Sen and D. Towsley. Proxy prefix caching for multimedia streams. IEEE Transaction on Multimedia, pages 1310{1318, 1999.

[18] C.-L. C. Te-Shou Su, Shih-Yu Huang and J.-S. Wang. Optimal chaining scheme for video-on-demand applications on collaborative networks. IEEE Transactions on multimedia, 7(5):972{980, October 2005.

[19] S.-H. G. C. Victor O.K, Li Jiancong Chen. Multipath routing for video delivery over bandwidth-limited network. IEEE Trans, 22(10):1920{1932, 2004.

[20] J. Almeida, D.Eager, and M. Vernon. A Hybrid Caching Strategy for Streaming Media Files. In Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking, January 2001.

[21] A. Dan, D. Sitaram, and P. Shahabuddin. Scheduling Policies for an On-Demand Video Server with Batching. In Proceedings of the ACM Conference on Multimedia, pages 391-398, October 1994.

[22] A. Dan, D. M. Dias, R. Mukherjee, D. Sitaram, and R.Tewari. Buffering and Caching in Large-Scale Video Servers. In Digest of Papers. IEEE International Computer Conference, pages 217-225, March 1995.

[23] A. Dan, P. Shahabuddin, D. Sitaram, and D. Towsley. Channel Allocation under batching and VCR Control in Movie-on-Demand Servers. Journal of Parallel and Distributed Computing, 30(2): 168-179, November 1995.

[24] A. Dan and D. Sitaram. A Generalized Interval Caching Policy for Mixed Interactive and Long Video Workloads. In Proceedings of Multimedia Computing and Networking Conference (MMCN), pages 344-351, January 1996.

[25] L. Golubchik, J. C. S. Lui, and R. Muntz. Reducing I/O Demand in Video-On-Demand Storage Servers. In Proceedings of the ACM SIGMETRICS Conference on Measurements and Modeling of Computer Systems, pages 25-36, May 1995.
[26] Kien A and Hua Simon Sheu. Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand System. In Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'97), pages 89-100, September 1997.
[27] N. J. Sarhan and C. R. Das. A Simulation-Based Analysis of Scheduling Policies for Multimedia Servers. In Proceedings of the Annual Simulation Symposium, pages 183 - 190, March 30 - April 2, 2003.

Acknowledgments

I express my sincere gratitude towards my guide **Prof. Sridhar Iyer** for his constant support and encouragement. His invaluable guidance has been instrumental in the successful completion of the project work

I would like to thank Mms. Saraswathi Krithivasan for her help in the completion of my project work.

Last but not the least; I would like to thank the entire KReSIT family for making my stay at ITT Bombay a memorable one.

Appendix

Topology Generator

```
numnodes= input('Enter number of nodes in the network:');
%Enter the level of the network
levelnum= input('Enter the number of level:');
%clientnum= input('Enter the client number:');
%Verifying the number of the level
while (levelnum > numnodes) && (numnodes~= 1)
   levelnum=input('Number level should be less than number of nodes:')
end
%global/general vaviable storing,
%nodenum row 1, second row, third row  region number,forth clientnum
 generalvar= ones(5,1);
 generalvar(1,1)= numnodes;
 generalvar(2,1)= levelnum;
 %Table numbering the link
 Number_link= ones(numnodes);
 %End creating the table numbering the link.
 relaynodes= ones(numnodes); % declare the relay nodes matrix n*n
 M=triu(relaynodes,1);   % take the upper triangular
 levelnodes= ones(numnodes,1);
 region= ones(numnodes,2);
 Links=triu(Number_link,1);
 %Intial zero to the matrix
 for i=1:numnodes
    for j=i+1:numnodes
           M(i,j)= 0;
              Links(i,j)=0;
     end
end
% initial the levelnodes matrix to zero
for p=1:numnodes
   for h=1:2
       region(p,h)= 0;
   end
end
%Start generating the random topology N*N matrix
linknumber=1;
levelnodes(1,1)= 1;
i= 1;
while i~= numnodes
    position= round(i*rand(1,1));
    if position== 0
       position= 1;
    end
         if levelnodes(position,1)== levelnum
              %will do pick the random number again
          else
                 %linknode= round(8*rand(1,1));
                 linknode= round(3*rand(1,1));
                 if linknode== 0
                    linknode= 1;
                 end
                 halfnodes= round(numnodes/2);
                 halflevel= round(levelnum/2);
```

```
                    if (max(levelnodes)< levelnum) && (levelnum> halfnodes)
                        j= 1;
                        while j~= numnodes
                            if levelnodes(j,1)== max(levelnodes)
                                position= j;
                                break
                            end
                            j= j+1;
                        end
                    else
                        if (max(levelnodes)< levelnum) && (i>= halfnodes)
                          j= 1;
                            while j~= numnodes
                                if levelnodes(j,1)== max(levelnodes)
                                position= j;
                                break
                                end
                            j= j+1;
                            end
                        end
                    end
                    levelnodes(i+1,1)= levelnodes(position,1)+1;
                    if max(levelnodes)< round(halfnodes)
                    M(position,i+1)= 256*linknode;
                    else
                    M(position,i+1)= 128*linknode;
                    end
                    Links(position,i+1)= linknumber;
                    linknumber=linknumber+1;
                    region(i+1,1)= position;
                    region(i+1,2)= 0;
                    region(position,2)= i+1;
                    i= i+1;
          end
end
% End generating the random nodes N*N martix
generalvar(5,1)= numnodes-1; %record the link number in the gengeral variables
 Total_level= max(levelnodes)
 csvwrite('NtoNmatrix.dat',M);
 csvwrite('nodeslevel.dat',levelnodes);
 csvwrite('suce_pred_node.dat',region);
 csvwrite('Link_number.dat',Links);
 %Generate the client to each region
   count= 0; % count is the number of the region in the entire network topology
   for r=1: numnodes
       if region(r,2)== 0
           count= count+1;
       end
   end
%Enter the number of client
 Total_number_in_region= count
%Client number input
clientnum= input('Enter the client number:');
generalvar(3,1)= count;
generalvar(4,1)= clientnum;
while clientnum< count
   clientnum= input('Enter new number, number of client should be greater or equal to number
of region:');
end
clientregion= ones(clientnum+1,count); % declare the client-region based matrix
```

```
regionbased= ones(count,levelnum+2);   % declare the region-based matrix
                                       % levelnum+1 column is the number of client in the region
                                       % levelnum+2 column is the number of
                                       % intermediate node in the path
%initial the client region matrix with value zero
   for ir=2:clientnum+1
       for ic=1:count
           clientregion(ir,ic)= 0;
       end
   end
   rp=1;% region position index
   for r=1: numnodes
       if region(r,2)== 0
           clientregion(1,rp)= r;
           rp= rp+1;
       end
   end
% Generate number of client in each region randomly
% initial region based matrix to zero
  for rr=1:count
      for rc=1:levelnum
          regionbased(rr,rc)= 0;
      end
  end
% ri is the region index for loop 'for'
  temp= clientnum-count;
  temp1= clientnum-count;
  clientc= 0; % countc the number of client that have been distrituted
  for ri=1:count
        halfregionnum= round(count/2);
      if temp>= 1
          if ri== count
              regionbased(ri,levelnum+1)= temp1-clientc+1;
          else
               cn= round((temp/halfregionnum)*rand(1,1));
               regionbased(ri,levelnum+1)= cn+1;
               clientc= clientc+cn;
          end
      else
      regionbased(ri,levelnum+1)= 1;
      end
  end

%Ditributing the link bandwith randomly to the clients in each region
  startpoint=2;
  for id=1:count
      for k=startpoint:regionbased(id,levelnum+1)+startpoint-1
          linkvalue= round(8*rand(1,1)); % the link bandwith vary from 64 to 512 kbps
           if linkvalue== 0
               linkvalue= 1;
           end
          clientregion(k,id)= 64*linkvalue;
      end
      startpoint= startpoint+regionbased(id,levelnum+1);
      k= startpoint;
  end
%Start from this point, finding the  path.
%from node in the region to S
%count is the region number in the entire network
```

```
  for i=1:count
      R= clientregion(1,i);% assign the number of the region to varable R
      numpath= levelnodes(R,1);
      regionbased(i,numpath)= R;
      regionbased(i,levelnum+2)= numpath-1;
      regionbased(i,1)= 1;
        while (R~= 1)
            k= 1;
          while (k~= numnodes)
             if M(k,R)~= 0
                numpath= numpath-1;
                regionbased(i,numpath)= k;
                R= k;
                break
             end
             k= k+1;
          end
        end
  end
%End finding the path from the node covered the region to source(S)
%Writing to data file, exporting the matrix client in each region matrix
%to client-in-each-region.dat file
%and exporting data from region-based matrix to region-based.date file
csvwrite('client_in_each_region.dat',clientregion);
csvwrite('region_based.dat',regionbased);
csvwrite('generalvariable.dat',generalvar);
%Build the clients'contraints table
%The table consites of the rate constraint, delay constraint and optimal rate
%the rate constraint is between 64 to 256 kbps
%the delay is between 15 mns to 60 mns
%generate the clients'constraints randomly
clientconstraint= ones(clientnum,5);
 for i=1:clientnum
     rate= round(4*rand(1,1)); % generate the rate constraint
     if rate== 0
        rate= 1;
     end
     clientconstraint(i,1)= rate*64;
     delay= round(6*rand(1,1));% generate the delay constraint
     if delay== 0
        delay= 1;
     end
     clientconstraint(i,2)= delay*30;
     clientconstraint(i,3)=0; % initial the optimal rate to zero
 end
csvwrite('client_constraints.dat',clientconstraint);
%End generating the clients'constraints
```

Client Request Pattern Generator

```
%Finding the optimal rate for the client in the network given its rate
%constraint and delay constraint
NtoNmatrix=csvread('NtoNmatrix.dat'); % Random intermediate nodes matrix
clientregion=csvread('client_in_each_region.dat');%client in each region matrix
regionbased=csvread('region_based.dat');% Regionbased matrix storing the region and the path
clientconstraint= csvread('client_constraints.dat'); %Client constraint table
weakness_link_region= csvread('weakness_link_region.dat');
temporary_caching_place=csvread('temporary_caching_place.dat');
generalvar= csvread('generalvariable.dat');
Link_number_m= csvread('Link_number.dat');
a_r=csvread('rate.dat');
S_d=csvread('simulation_time.dat');
average_arrival_r=a_r(1,1);
Simulation_duration=S_d(1,1);
nodesnum= generalvar(1,1);
levelnum= generalvar(2,1);
regionnum= generalvar(3,1);
clientnum= generalvar(4,1);
link_number= generalvar(5,1);
Total_clients= round(average_arrival_r*Simulation_duration*60);
%Creating the client's request table
Client_request= ones(Total_clients,6);
Client_Pro= ones(Total_clients,3);
%snap_shot for the duration of 10 minutes
Snap_shot= average_arrival_r* 10;
%St is a start up time
 st=0;
loop= Simulation_duration*60/10;
client_number=1;
for i=1:loop
    destination= client_number+Snap_shot-1;
   for j=client_number:destination
      %Distributing the link bandwidth
       %linkvalue= round(4*rand(1,1)); % the link bandwith vary from 64 to 512 kbps
       linkvalue= round(1*rand(1,1)); % the link bandwith vary from 64 to 512 kbps
        if linkvalue== 0
           linkvalue= 1;
        end
        Client_request(j,1)=128*linkvalue;   %Client_request(j,1)=128*linkvalue;
      %End distributing the link bandwidth
      %Distributing the rate constraint
      %rate= round(4*rand(1,1)); % generate the rate constraint
      rate= round(1*rand(1,1)); % generate the rate constraint
          if rate== 0
           rate= 1;
          end
        Client_request(j,2)= rate*128;
      %End distributing the rate constraint
      %Distributing the delay constraint
      delay= round(4*rand(1,1));% generate the delay constraint
       % delay= round(1*rand(1,1));% generate the delay constraint
        if delay== 0
           delay= 1;
        end
      Client_request(j,3)= delay*90;
      %End distributing the delay constraint
```

```
            %Request time
               r_time= round(10*rand(1,1));% generate the delay constraint
               Client_request(j,4)= r_time+st;
               Client_Pro(j,1)= j;
               Client_Pro(j,3)= r_time+st;
            %End distribution request time
            %Optimal rate
              Client_request(j,5)=0;
            %End initiating the optimal rate
             %Distributing region number
            region= round((regionnum)*rand(1,1));
            if region==0
               region=1;
            end
            Client_request(j,6)= region;
            Client_Pro(j,2)= region;
            %End distributing region number

         end
         client_number= destination+1;
         st=st+10;

      end
      Client_Pro1= sortrows(Client_Pro,3);
      csvwrite('client_request1_s.dat',Client_Pro1);
      csvwrite('client_request1_ftp.dat',Client_Pro1);
      csvwrite('client_request_s.dat',Client_request);
      csvwrite('client_request_ftp.dat',Client_request);
      Tc= ones(1,1);
      Tc(1,1)=Total_clients;
      csvwrite('Total_clients.dat',Tc);
```

PSM module

```
%Pure streaming mechanism (PSM)
NtoNmatrix=csvread('NtoNmatrix.dat'); % Random intermediate nodes matrix
clientregion=csvread('client_in_each_region.dat');%client in each region matrix
regionbased=csvread('region_based.dat');% Regionbased matrix storing the region and the path
clientconstraint= csvread('client_constraints.dat'); %Client constraint table
weakness_link_region= csvread('weakness_link_region.dat');
Total_clients=csvread('Total_clients.dat');
temporary_caching_place=csvread('temporary_caching_place.dat');
generalvar= csvread('generalvariable.dat');
Link_number_m= csvread('Link_number.dat');
nodesnum= generalvar(1,1);
levelnum= generalvar(2,1);
regionnum= generalvar(3,1);
clientnum= generalvar(4,1);
link_number= generalvar(5,1);
%global Total_clients;
Client_Pro1=csvread('client_request1_ftp.dat');
Client_request=csvread('client_request_ftp.dat');
```

```
% Link numer and its busy period
link_busy_period= ones(link_number,4);
%Server Side object table
for i=1:link_number
    link_busy_period(i,1)=0;
    link_busy_period(i,2)=0;
    link_busy_period(i,3)=0;
    link_busy_period(i,4)=0;
end
Server_side= ones(regionnum,6);
for i=1:regionnum
    Server_side(i,1)=0;
    Server_side(i,2)=0;
    Server_side(i,3)=0;
    Server_side(i,4)=0;
    Server_side(i,5)=0;
    Server_side(i,6)=0;
    Server_side(i,7)=0;
end
%Start doing simulation
%Link_number_m : maintain the link number matrix
%Server_side: maintain the information about the object being streaming
%Client_request: maintain the information about the client's request
%link_busy_period : maintain the link status, busy or free
%Client_Pro: maintain the current client property
%weakness_link_region: maintain the weakness link in the region
%temporary_caching_place: streaming position of reach region (1,regionnum)
%regionbased: maintain the path from s to region node
%%%%Streaming
Movieduration= 120*60;
Threshold_period=0; % We can vary this value
start_up_time= Client_Pro1(1,3);
 for i=1:Total_clients
     Client_number= Client_Pro1(i,1);
     Client_region= Client_Pro1(i,2);
     Client_request_time= Client_Pro1(i,3);
     Client_delay= Client_request(Client_number,3);
     Client_rate_c= Client_request(Client_number,2);
     region_start_streaming_from= Server_side(Client_region,5);
     region_end_streaming_at= Server_side(Client_region,6);
     previous_client_request_time=Server_side(Client_region,7);
     waiting_time= Client_request_time+Client_delay;
     Client_start= waiting_time;
     Client_end= waiting_time+120;
     Weaklink=Client_request(Client_number,1);
     if (Client_request(Client_number,1)>weakness_link_region(Client_request(Client_number,6),1))
         Weaklink= weakness_link_region(Client_request(Client_number,6),1);
     end
       % In case of client join other request
if Client_request_time < region_start_streaming_from && region_end_streaming_at> waiting_time
           %Current_client_start= Client_request_time+Client_delay;
           if Client_request_time== previous_client_request_time
               client_rate= Server_side(Client_region,2);
                if client_rate>= Client_rate_c
                       delay= ((client_rate-Weaklink)*7200/Weaklink)/60;
                         if delay <= Client_delay
                             Client_request(Client_number,5)=client_rate;
                         else
                Client_request(Client_number,5)=-11; %first case delay constrait violated
                  end
          else
             Client_request(Client_number,5)=-12; % second case rate constraint violated
          end
       else
          Client_request(Client_number,5)=-111; % Special case
       end
    % In case it needs to start a new stream or join with other region
    else
       %Check the link status, whether we can initiate the stream or
       %not
   if Client_request_time> region_start_streaming_from && waiting_time < region_end_streaming_at
          Client_request(Client_number,5)=-13;% third case
   end
       % In case of client wait till the old stream finish and
       % initiate a new stream or joint with others
 if Client_request_time> region_start_streaming_from && waiting_time > region_end_streaming_at
          remaining_delay= waiting_time - region_end_streaming_at;
             expected_rate= remaining_delay*60*Weaklink/7200+ Weaklink;
             link_is_free=1;
               Remaining_bandwidth= ones(1,levelnum-1);
                for k=1:levelnum
                   Reamining_bandwidth(1,k)= 100000;
                end
              for in=1:levelnum-1
                if (regionbased(Client_region,in+1)~=0)
          LinksN= Link_number_m(regionbased(Client_region,in),regionbased(Client_region,in+1));
        V_bandwidth= NtoNmatrix(regionbased(Client_region,in),regionbased(Client_region,in+1));
                   if link_busy_period(LinksN,2)> waiting_time
                      link_is_free=0;
                   end
                   Remaining_bandwidth(1,in)= link_busy_period(LinksN,3);
                end
              end

              if link_is_free==1
               for in=1:levelnum-1
                if (regionbased(Client_region,in+1)~=0)
          LinksN= Link_number_m(regionbased(Client_region,in),regionbased(Client_region,in+1));
        V_bandwidth= NtoNmatrix(regionbased(Client_region,in),regionbased(Client_region,in+1));
                   link_busy_period(LinksN,1)= Client_request_time;
                   %end_time= expected_rate*7200/V_bandwidth;
                   link_busy_period(LinksN,2)= Client_start+ 120; %(end_time/60);
                   link_busy_period(LinksN,3)= V_bandwidth-expected_rate ;
                   link_busy_period(LinksN,4)= Client_region;
                end
               end
                 if expected_rate>= Client_rate_c
                   Client_request(Client_number,5)=expected_rate;
                   Server_side(Client_region,2)= expected_rate;
                   Server_side(Client_region,3)= temporary_caching_place(1,Client_region);
                   Server_side(Client_region,4)=7200;
                   Server_side(Client_region,5)=Client_start;
                   Server_side(Client_region,6)=Client_end;
                   Server_side(Client_region,7)=Client_request_time;
                 else
                   Client_request(Client_number,5)=-14; %fourth case
                 end
        else %being else here
          % Find out which region that client can join
          if min(Remaining_bandwidth)>0 && min(Remaining_bandwidth)>= expected_rate
           for in=1:levelnum-1
                if (regionbased(Client_region,in+1)~=0)
 LinksN= Link_number_m(regionbased(Client_region,in),regionbased(Client_region,in+1));
V_bandwidth= NtoNmatrix(regionbased(Client_region,in),regionbased(Client_region,in+1));
                  link_busy_period(LinksN,1)= Client_request_time;
                  %end_time= expected_rate*7200/V_bandwidth;
                  link_busy_period(LinksN,2)= Client_start+ 120;%(end_time/60);
                  link_busy_period(LinksN,3)= link_busy_period(LinksN,3)-expected_rate ;
                  link_busy_period(LinksN,4)= Client_region;
                end
              end
                 if expected_rate>= Client_rate_c
                  Client_request(Client_number,5)=expected_rate;
                  Server_side(Client_region,2)= expected_rate;
                  Server_side(Client_region,3)= temporary_caching_place(1,Client_region);
                  Server_side(Client_region,4)=7200;
                  Server_side(Client_region,5)=Client_start;
                  Server_side(Client_region,6)=Client_end;
                  Server_side(Client_region,7)=Client_request_time;
                 else
                  Client_request(Client_number,5)=-15; %fifth case
                 end
                    else % start begin1
          preferable_rate=-1;
          for h=1:regionnum
              Region_encoded_rate= Server_side(h,2);
              Streaming_start= Server_side(h,5);
              if Client_start>= Streaming_start
                 remaining_delay1= waiting_time-Streaming_start;
                 delay_c= ((Region_encoded_rate-Weaklink)*7200/Weaklink)/60;

                 if delay_c<=remaining_delay1
                   if preferable_rate< Region_encoded_rate
                      preferable_rate= Region_encoded_rate;
                   end
                 end
              end
          end
          if preferable_rate~=-1
              expected_rate= preferable_rate;

              for in=1:levelnum-1
                 if (regionbased(Client_region,in+1)~=0)
  LinksN= Link_number_m(regionbased(Client_region,in),regionbased(Client_region,in+1));
V_bandwidth= NtoNmatrix(regionbased(Client_region,in),regionbased(Client_region,in+1));
                  link_busy_period(LinksN,1)= Client_request_time;
                  %end_time= expected_rate*7200/V_bandwidth;
                  link_busy_period(LinksN,2)= Client_start+ 120; %(end_time/60);
                  link_busy_period(LinksN,3)= V_bandwidth-expected_rate ;
                  link_busy_period(LinksN,4)= Client_region;
                 end
              end
                 if expected_rate>= Client_rate_c
                    Client_request(Client_number,5)=expected_rate;
                    Server_side(Client_region,2)= expected_rate;
```

```
                                Server_side(Client_region,3)= temporary_caching_place(1,Client_region);
                                Server_side(Client_region,4)=7200;
                                Server_side(Client_region,5)=Client_start;
                                Server_side(Client_region,6)=Client_end;
                                Server_side(Client_region,7)=Client_request_time;
                              else
                                Client_request(Client_number,5)=-16; %sixth case
                              end
                          else
                            Client_request(Client_number,5)=-17;%seventh case
                          end
                          end %end begin1
                        end %end else here
                    end
                  % In case we can initiate the new stream
                  % New stream
                  if Client_request_time>= region_end_streaming_at
                    expected_rate= (Client_delay*60*Weaklink)/7200 + Weaklink;
                    for in=1:levelnum-1
                      if (regionbased(Client_region,in+1)~=0)
                LinksN= Link_number_m(regionbased(Client_region,in),regionbased(Client_region,in+1));
                V_bandwidth= NtoNmatrix(regionbased(Client_region,in),regionbased(Client_region,in+1));
                        link_busy_period(LinksN,1)= Client_request_time;
                        %end_time= expected_rate*7200/V_bandwidth;
                        link_busy_period(LinksN,2)= Client_start+ 120; %(end_time/60);
                        link_busy_period(LinksN,3)= V_bandwidth-expected_rate ;
                        link_busy_period(LinksN,4)= Client_region;
                      end
                    end
                      if expected_rate>= Client_rate_c
                        Client_request(Client_number,5)=expected_rate;
                        Server_side(Client_region,2)= expected_rate;
                        Server_side(Client_region,3)= temporary_caching_place(1,Client_region);
                        Server_side(Client_region,4)=7200;
                        Server_side(Client_region,5)=Client_start;
                        Server_side(Client_region,6)=Client_end;
                        Server_side(Client_region,7)=Client_request_time;
                      else
                        Client_request(Client_number,5)=-18; % huit case
                      end
                  end
                 end
         end
        csvwrite('link_busy_period_s.dat', link_busy_period);
        csvwrite('Server_side_table_s.dat', Server_side);
        csvwrite('client_request1_s.dat',Client_Pro1);
        csvwrite('client_request_s.dat',Client_request);
        sum=0;
        accepted_client=0;
        rejected_client=0;
        percentage_improvement=0;
        for j=1:Total_clients
          if Client_request(j,5) > 0
            accepted_client=accepted_client+1;
            sum=sum+Client_request(j,5);
            improve=(Client_request(j,5)-Client_request(j,2))*100/Client_request(j,2);
            percentage_improvement=percentage_improvement+improve;
          else
            rejected_client = rejected_client+1;
```

```
    end
  end
  %sum
  %global Average_s;
  %global percentage_r_s;
  %global percentage_a_s;
  s_info= ones(1,4);
  Average_s= sum/accepted_client;
  %accepted_client
  percentage_a_s= (accepted_client/Total_clients)*100;
  %rejected_client
  percentage_r_s= (rejected_client/Total_clients)*100;
  percentage_im=percentage_improvement/accepted_client;
  s_info(1,1)=Average_s;
  s_info(1,2)=percentage_r_s;
  s_info(1,3)=percentage_a_s;
  s_info(1,4)=percentage_im;
  csvwrite('s_info.dat',s_info);
```

PSM module

```
%Finding the optimal rate for the client in the network given its rate
%constraint and delay constraint
NtoNmatrix=csvread('NtoNmatrix.dat'); % Random intermediate nodes matrix
clientregion=csvread('client_in_each_region.dat');%client in each region matrix
regionbased=csvread('region_based.dat');% Regionbased matrix storing the region and the path
clientconstraint= csvread('client_constraints.dat'); %Client constraint table
weakness_link_region= csvread('weakness_link_region.dat');
Total_clients=csvread('Total_clients.dat');
temporary_caching_place=csvread('temporary_caching_place.dat');
generalvar= csvread('generalvariable.dat');
Link_number_m= csvread('Link_number.dat');
nodesnum= generalvar(1,1);
levelnum= generalvar(2,1);
regionnum= generalvar(3,1);
clientnum= generalvar(4,1);
link_number= generalvar(5,1);
%global Total_clients;
Client_Pro1=csvread('client_request1_ftp.dat');
Client_request=csvread('client_request_ftp.dat');
% Link numer and its busy period
link_busy_period= ones(link_number,4);
%Server Side object table
for i=1:link_number
   link_busy_period(i,1)=0;
   link_busy_period(i,2)=0;
   link_busy_period(i,3)=0;
   link_busy_period(i,4)=0;
end
Server_side= ones(regionnum,6);
for i=1:regionnum
   Server_side(i,1)=0;
   Server_side(i,2)=0;
   Server_side(i,3)=0;
   Server_side(i,4)=0;
   Server_side(i,5)=0;
   Server_side(i,6)=0;
```

```
end
%Start doing simulation for ftp
%Link_number_m : maintain the link number matrix
%Server_side: maintain the information about the object being streaming
%Client_request: maintain the information about the client's request
%link_busy_period : maintain the link status, busy or free
%Client_Pro: maintain the current client property
%weakness_link_region: maintain the weakness link in the region
%temporary_caching_place: streaming position of reach region (1,regionnum)
%regionbased: maintain the path from s to region node
%%%%Streaming
Movieduration= 120*60;
Threshold_period=0; % We can vary this value
%start_up_time= Client_Pro1(1,3);
 for i=1:Total_clients
     Client_number= Client_Pro1(i,1);
     Client_region= Client_Pro1(i,2);
     Client_request_time= Client_Pro1(i,3);
     Client_delay= Client_request(Client_number,3);
     Client_rate_c= Client_request(Client_number,2);
     region_start_streaming_from= Server_side(Client_region,5);
     region_end_streaming_at= Server_side(Client_region,6);
     waiting_time= Client_request_time+Client_delay;
     Client_start= waiting_time;
     Client_end= waiting_time+120;
     Weaklink=Client_request(Client_number,1);
     if (Client_request(Client_number,1)>weakness_link_region(Client_request(Client_number,6),1))
        Weaklink= weakness_link_region(Client_request(Client_number,6),1);
     end
       % In case of client join other request in the same region
       if Client_request_time < region_start_streaming_from && region_end_streaming_at > waiting_time
           client_rate= Server_side(Client_region,2);
             if client_rate>= Client_rate_c
                  delay= ((client_rate-Weaklink)*7200/Weaklink)/60;

                   if delay <= Client_delay
                      Client_request(Client_number,5)=client_rate;
                   else
                      Client_request(Client_number,5)=-11; %first case delay constrait violated
                   end
             else
                Client_request(Client_number,5)=-12; % second case rate constraint violated
             end
      % In case it needs to start a new stream or join with other region
       else
          %Check the link status, whether we can initiate the stream or
          %not
        if Client_request_time> region_start_streaming_from && waiting_time < region_end_streaming_at
             Client_request(Client_number,5)=-13;% third case
          end
          % In case of client wait till the old stream finish and
          % initiate a new stream or joint with others
          if Client_request_time> region_start_streaming_from && waiting_time > region_end_streaming_at
             remaining_delay= waiting_time - region_end_streaming_at;
                 expected_rate= remaining_delay*60*Weaklink/7200+ Weaklink;
                 link_is_free=1;
                    Remaining_bandwidth= ones(1,levelnum-1);

                     for k=1:levelnum
                        Reamining_bandwidth(1,k)= 100000;
           end
         for in=1:levelnum-1
           if (regionbased(Client_region,in+1)~=0)
          LinksN= Link_number_m(regionbased(Client_region,in),regionbased(Client_region,in+1));
          V_bandwidth= NtoNmatrix(regionbased(Client_region,in),regionbased(Client_region,in+1));
               if link_busy_period(LinksN,2)> waiting_time
                   link_is_free=0;
               end
               Remaining_bandwidth(1,in)= link_busy_period(LinksN,3);
           end
        end
        if link_is_free==1
         for in=1:levelnum-1
           if (regionbased(Client_region,in+1)~=0)
             LinksN= Link_number_m(regionbased(Client_region,in),regionbased(Client_region,in+1));
            V_bandwidth= NtoNmatrix(regionbased(Client_region,in),regionbased(Client_region,in+1));
               link_busy_period(LinksN,1)= Client_request_time;
               end_time= expected_rate*7200/V_bandwidth;
               link_busy_period(LinksN,2)= Client_request_time+(end_time/60);
               link_busy_period(LinksN,3)= 0;%V_bandwidth-expected_rate ;
               link_busy_period(LinksN,4)= Client_region;
           end
          end
             if expected_rate>= Client_rate_c
               Client_request(Client_number,5)=expected_rate;
               Server_side(Client_region,2)= expected_rate;
               Server_side(Client_region,3)= temporary_caching_place(1,Client_region);
               Server_side(Client_region,4)=7200;
               Server_side(Client_region,5)=Client_start;
               Server_side(Client_region,6)=Client_end;
             else
               Client_request(Client_number,5)=-14; %fourth case
             end
        else %being else here
            % Find out which region that client can join
            if min(Remaining_bandwidth)>0 && min(Remaining_bandwidth)>= expected_rate
               for in=1:levelnum-1
                  if (regionbased(Client_region,in+1)~=0)
             LinksN= Link_number_m(regionbased(Client_region,in),regionbased(Client_region,in+1));
            V_bandwidth= NtoNmatrix(regionbased(Client_region,in),regionbased(Client_region,in+1));
                      link_busy_period(LinksN,1)= Client_request_time;
                      end_time= expected_rate*7200/V_bandwidth;
                      link_busy_period(LinksN,2)= Client_request_time+(end_time/60);
                      link_busy_period(LinksN,3)= 0;%link_busy_period(LinksN,3)-expected_rate ;
                      link_busy_period(LinksN,4)= Client_region;
                  end
               end
                    if expected_rate>= Client_rate_c
                      Client_request(Client_number,5)=expected_rate;
                      Server_side(Client_region,2)= expected_rate;
                      Server_side(Client_region,3)= temporary_caching_place(1,Client_region);
                      Server_side(Client_region,4)=7200;
                      Server_side(Client_region,5)=Client_start;
                      Server_side(Client_region,6)=Client_end;
                    else
                      Client_request(Client_number,5)=-15; %fifth case
                    end
            else % start begin1
               preferable_rate=-1;
            for h=1:regionnum
```

```
                    Region_encoded_rate= Server_side(h,2);
                    Streaming_start= Server_side(h,5);
                    if Client_start>= Streaming_start
                        remaining_delay1= waiting_time-Streaming_start;
                        delay_c= ((Region_encoded_rate-Weaklink)*7200/Weaklink)/60;
                       if delay_c<=remaining_delay1
                           if preferable_rate< Region_encoded_rate
                               preferable_rate= Region_encoded_rate;
                           end
                        end
                    end
                end
                if preferable_rate~=-1
                    expected_rate= preferable_rate;
                  for in=1:levelnum-1
                        if (regionbased(Client_region,in+1)~=0)
               LinksN= Link_number_m(regionbased(Client_region,in),regionbased(Client_region,in+1));
               V_bandwidth= NtoNmatrix(regionbased(Client_region,in),regionbased(Client_region,in+1));
                            link_busy_period(LinksN,1)= Client_request_time;
                            end_time= expected_rate*7200/V_bandwidth;
                            link_busy_period(LinksN,2)= Client_request_time+(end_time/60);
                            link_busy_period(LinksN,3)= 0 ;
                            link_busy_period(LinksN,4)= Client_region;
                        end
                    end
                        if expected_rate>= Client_rate_c
                            Client_request(Client_number,5)=expected_rate;
                            Server_side(Client_region,2)= expected_rate;
                            Server_side(Client_region,3)= temporary_caching_place(1,Client_region);
                            Server_side(Client_region,4)=7200;
                            Server_side(Client_region,5)=Client_start;
                            Server_side(Client_region,6)=Client_end;
                        else
                            Client_request(Client_number,5)=-16; %sixth case
                        end
                else
                    Client_request(Client_number,5)=-17;%seventh case
                end
                end %end begin1
             end %end else here
end
% In case we can initiate the new stream
% New stream
if Client_request_time>= region_end_streaming_at

    expected_rate= (Client_delay*60*Weaklink)/7200 + Weaklink;
    conditions=0;
    for in=1:levelnum-1
      if (regionbased(Client_region,in+1)~=0)
          if regionbased(Client_region,in+1)~=temporary_caching_place(1,Client_region)
              conditions=1;
          end
          if conditions==1
          LinksN= Link_number_m(regionbased(Client_region,in),regionbased(Client_region,in+1));
          V_bandwidth= NtoNmatrix(regionbased(Client_region,in),regionbased(Client_region,in+1));
          link_busy_period(LinksN,1)= Client_request_time;
         % end_time= Client_start+120;
          link_busy_period(LinksN,2)= Client_request_time+120;
          link_busy_period(LinksN,3)= V_bandwidth-expected_rate ;
          link_busy_period(LinksN,4)= Client_region;
```

```
                else
                LinksN= Link_number_m(regionbased(Client_region,in),regionbased(Client_region,in+1));
                V_bandwidth= NtoNmatrix(regionbased(Client_region,in),regionbased(Client_region,in+1));
                link_busy_period(LinksN,1)= Client_request_time;
                end_time= expected_rate*7200/V_bandwidth;
                link_busy_period(LinksN,2)= Client_request_time+(end_time/60);
                link_busy_period(LinksN,3)= 0 ;
                link_busy_period(LinksN,4)= Client_region;
                end
               end
           end
               if expected_rate>= Client_rate_c
                   Client_request(Client_number,5)=expected_rate;
                   Server_side(Client_region,2)= expected_rate;
                   Server_side(Client_region,3)= temporary_caching_place(1,Client_region);
                   Server_side(Client_region,4)=7200;
                   Server_side(Client_region,5)=Client_start;
                   Server_side(Client_region,6)=Client_end;
               else
                   Client_request(Client_number,5)=-18; % huit case
               end
         end
       end
end
      %end temp
     csvwrite('link_busy_period_ftp.dat', link_busy_period);
     csvwrite('Server_side_table_ftp.dat', Server_side);
     csvwrite('client_request1_ftp.dat',Client_Pro1);
     csvwrite('client_request_ftp.dat',Client_request);
     sum=0;
     percentage_improvement=0;
     accepted_client=0;
     rejected_client=0;
     for j=1:Total_clients
        if Client_request(j,5) > 0
           accepted_client=accepted_client+1;
           sum=sum+Client_request(j,5);
           improve=(Client_request(j,5)-Client_request(j,2))*100/Client_request(j,2);
           percentage_improvement=percentage_improvement+improve;
        else
           rejected_client = rejected_client+1;
        end
     end
     %sum
     %global Average_ftp;
     %global percentage_r_ftp;
     %global percentage_a_ftp;
     ftp_info= ones(1,4);
     Average= sum/accepted_client;
        %accepted_client
        percentage_a_ftp= (accepted_client/Total_clients)*100;
        %rejected_client
        percentage_r_ftp= (rejected_client/Total_clients)*100;
        percentage_im=percentage_improvement/accepted_client;
        ftp_info(1,1)=Average;
        ftp_info(1,2)=percentage_r_ftp;
        ftp_info(1,3)=percentage_a_ftp;
        ftp_info(1,4)=percentage_im;
        csvwrite('ftp_info.dat',ftp_info);
```

Main functions (Simulator)

```
        %global average_arrival_rate;
        %global Simulation_duration;

        % average rate is starting from 1 to 30
        average_rate_ftp=0;
        percentage_ftp_r=0;
        percentage_ftp_a=0;
        average_rate_s=0;
        percentage_s_r=0;
        percentage_s_a=0;
        percentage_im_ftp=0;
        percentage_im_s=0;
        table_ftp= ones(30,4);
        table_s= ones(30,4);
        percentage_rate_improvement=ones(30,2);
        ftp_streaming_rate=ones(30,2);
        ftp_streaming_percentage_a=ones(30,2);
        rate=ones(1,1);
        si=ones(1,1);
        simulation_p=csvread('simulation.dat');
        simulation=simulation_p(1,1);
        n_times=100;
        %average_arrival_rate=input('Average arrival rate');
          for l=1:30
            average_rate_ftp=0;
            percentage_ftp_r=0;
            percentage_ftp_a=0;
            average_rate_s=0;
            percentage_s_r=0;
            percentage_s_a=0;
            percentage_im_ftp=0;
            percentage_im_s=0;
              si(1,1)=simulation;
              csvwrite('simulation_time.dat',si);
              rate(1,1)=l/10;
              csvwrite('rate.dat',rate);
             for j=1:n_times
              streaming_clients
              streaming_ftp
              ftp_in=csvread('ftp_info.dat');
              average_rate_ftp= average_rate_ftp+ftp_in(1,1);
              percentage_ftp_r=percentage_ftp_r+ftp_in(1,2);
              percentage_ftp_a=percentage_ftp_a+ftp_in(1,3);
              percentage_im_ftp=percentage_im_ftp+ftp_in(1,4);
              streaming_info
         s_in=csvread('s_info.dat');
         average_rate_s= average_rate_s+s_in(1,1);
         percentage_s_r=percentage_s_r+s_in(1,2);
         percentage_s_a=percentage_s_a+s_in(1,3);
         percentage_im_s=percentage_im_s+s_in(1,4);
        end
      table_ftp(l,1)=l;
      kl=average_rate_ftp/n_times;
      table_ftp(l,2)=kl;
      ftp_streaming_rate(l,1)=kl;
      table_ftp(l,3)=percentage_ftp_r/n_times;
      table_ftp(l,4)=percentage_ftp_a/n_times;
```

```
    ftp_streaming_percentage_a(l,1)=percentage_ftp_a/n_times;
    table_ftp;
    percentage_rate_improvement(l,1)=percentage_im_ftp/n_times;
    table_s(l,1)=l;
    table_s(l,2)=average_rate_s/n_times;
    ftp_streaming_rate(l,2)=average_rate_s/n_times;
    table_s(l,3)=percentage_s_r/n_times;
    table_s(l,4)=percentage_s_a/n_times;
    ftp_streaming_percentage_a(l,2)=percentage_s_a/n_times;
    percentage_rate_improvement(l,2)=percentage_im_s/n_times;
   end
csvwrite('percentage_rate_improvement.dat',percentage_rate_improvement);
csvwrite('ftp_data_table.dat',table_ftp);
csvwrite('streaming_data_table.dat',table_s);
csvwrite('ftp_streaming_rate.dat',ftp_streaming_rate);
csvwrite('ftp_streaming_percentage_a.dat',ftp_streaming_percentage_a);

subplot(1,1,1)  %for ploting the graph, monitring the size of the axis
bar(ftp_streaming_rate,'group')  % bar chart plotting
title 'Comparision average rate of FTP and Streaming'
saveas(gcf,'ftp_s_rate', 'eps') % save the figure with eps extention

subplot(1,1,1)  %for ploting the graph, monitring the size of the axis
bar(ftp_streaming_percentage_a,'group')  % bar chart plotting
title 'Comparision percentage of accepted users request FTP and Streaming'
saveas(gcf,'ftp_s_accepted', 'eps') % save the figure with eps extention

subplot(1,1,1)  %for ploting the graph, monitring the size of the axis
bar(percentage_rate_improvement,'group')  % bar chart plotting
title 'Percentage Rate Improvment'
saveas(gcf,'percentage_rate_im', 'eps') % save the figure with eps extention
```