Efficient Streaming for Delay-tolerant Multimedia Applications

A thesis submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

by

Saraswathi Krithivasan
(Roll No. 03429601)

Under the guidance of
Prof. Sridhar Iyer

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY–BOMBAY
2008

To my family

INDIAN INSTITUTE OF TECHNOLOGY BOMBAY, INDIA

CERTIFICATE OF COURSE WORK

This is to certify that **Saraswathi Krithivasan** (Roll No. 03429601) was admitted to the candidacy of Ph.D. degree on July 2003, after successfully completing all the courses required for the Ph.D. programme. The details of the course work done are given below.

S.No	**Course Code**	**Course Name**	**Credits**
1	IT 605	Distributed Sytems	6
2	MA 603	Statistical Analysis and Design	6
3	IT 694	Seminar	4
4	IT 610	Quality of service in networks	6
5	HS 699	Communication and Presentation Skills	PP
		Total Credits	**22**

IIT Bombay

Date: Dy. Registrar (Academic)

Abstract

Delay-tolerant multimedia applications, where clients specify the time when playout must start, fit the profile of many emerging applications, including distance education and corporate training. Such applications typically depend on a Content Delivery Network (CDN), where the Content Service Provider (CSP) disseminates multimedia content to geographically dispersed clients through a distribution network from servers located on the CDN. In these applications, a client C_i connects to the source $\mathcal{S}$ at time t_0 and requests for contents with base encoding rate Γ and playout duration $\mathcal{T}$; C_i specifies the time it is willing to wait for the playout to start, its *delay tolerance* δ_i; Playout at C_i starts at (t_0+δ_i) and ends at $((t_0 + \delta_i) + \mathcal{T})$. Note that t_0 may also be the start time of a scheduled streaming session.

This thesis deals with the issue of maximizing the quality of the multimedia contents delivered at the clients even when link bandwidths are constrained in the path from the source to the clients. To achieve this, we develop a suite of algorithms that can exploit clients' delay tolerance considering the following parameters: (i) *Service type*: whether contents are streamed according to a schedule or occur on-demand and (ii) *Bandwidth*: whether link capacity is static or variable, using appropriate *resources* at the nodes – Transcoders, Layer encoders, or Streaming servers with transcoding or layering functionality. In particular, we consider the following three cases: (i) Scheduled streaming when bandwidths are static, (ii) Scheduled streaming when bandwidths are varying and predicted, and (iii) On-demand streaming when bandwidths are static. The algorithms developed are a result of formulating the objectives in an optimization framework or by an analysis of properties of the network topology and client requirements. Furthermore, where an optimal algorithm is computationally expensive, we develop algorithms based on heuristics as practical alternatives. The approaches are validated through extensive simulations, using topologies derived from real-world scenarios.

The algorithms developed in this thesis would help a CSP serving clients in a subscription based network in: (i) improving the quality of reception at the clients by leveraging their delay tolerance values, (ii) estimating the resources required to provide the best delivered rates to clients, and (iii) determining placement of resources to maximize the delivered rates at clients. Thus, using the analysis presented and algorithms developed in this thesis, a CSP can deploy resources in order to ensure effective quality of service to clients and efficient resource utilization by leveraging clients' delay tolerance.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the proliferation of world-wide computer networks, several popular streaming media applications have emerged: Universities offering their courses to a set of global subscribers, service providers streaming movies requested by their clients, and multinational corporations providing training to employees across cities. Heterogeneous communication architectures comprising of satellite, terrestrial links as well as the Internet are increasingly deployed for such applications. In these applications, a source disseminates multimedia contents (that may be encoded at different rates) to a set of geographically distributed clients, through links of varying capacities and characteristics [18].

We consider applications where the clients specify their requirements: a minimum rate at which the contents are to be played out (a minimum quality requirement) and a startup delay after which they want the playout to start (a specific start time requirement). We term such applications as **delay-tolerant applications**. Most of the contents in the education and entertainment domain are in this category. For such applications it is possible to allow clients the convenience of starting the playout at a specific time while improving the quality of the playout, even in the presence of bandwidth constrained links along the path from the source to the client. We have explored different dimensions of this problem in this thesis.

1.1 Focus area

Figure 1.1 depicts the area of focus for this thesis. We have considered two user-level parameters, *loss* and *start-up delay*. Loss occurs due to dropped data packets when there is insufficient bandwidth on a link to support the rate at which data is flowing through it. Start-up delay is

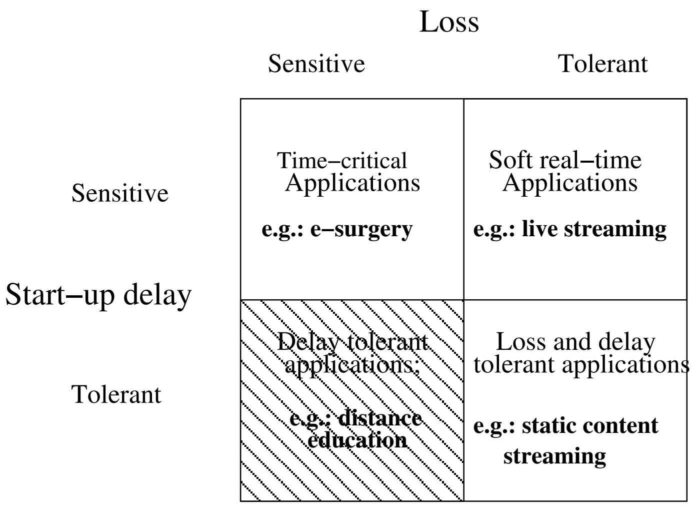

Loss
Sensitive
Tolerant
Start–up delay
Sensitive
Tolerant
Time–critical Applications
e.g.: e–surgery
Soft real–time Applications
e.g.: live streaming
Delay tolerant applications;
e.g.: distance education
Loss and delay tolerant applications
e.g.: static content streaming

Figure 1.1: The problem space

a small time lag due to buffering at the client to control jitter during the playout. We classify applications along the two dimensions *sensitivity* and *tolerance*.

1. Multimedia applications such as e-surgery which are demonstrated in real time fall under the *loss-sensitive* and *start-up delay sensitive* category, i.e., such applications need loss-free transmission in real time. These applications require dedicated bandwidth from the source to the client so that the quality of reception is guaranteed.

2. Applications such as live sports streaming fall under the *loss-tolerant* and *start-up delay sensitive* category. Such applications have been researched extensively. A review of the existing mechanisms for effective and efficient delivery of multimedia in [14] [51] indicates that existing work treats applications involving multimedia dissemination as soft real-time applications that can tolerate some transmission errors and *explores ways to minimize the startup delay*.

3. Applications such as streaming of personal video clips fall under the *loss-tolerant* and *start-up delay tolerant* category. We believe that such applications can be handled by the current best effort networks without any additional features. By allowing for *longer* buffering at the clients, quality can be improved but, without guarantees for loss-free playout. Typically, such applications do not have stringent quality requirements.

4. We focus on the fourth category of multimedia applications – those that are *loss-sensitive* and *start-up delay-tolerant*. We have developed techniques for exploiting client specified

delay tolerance in order to maximize delivered multimedia quality in heterogeneous network environments. Most multimedia applications in education and entertainment where contents are valid for a period of time, fall in this category. However, there is hardly any research that deals with improving delivered quality for applications in this category. We show that by providing flexibility of viewing time to the clients and allowing them to specify the start of playout, delivered quality can be improved even in the presence of links with bandwidth constraints. The basic idea is as follows: given that weak links with limited bandwidths can occur in the path from source to a client; when a client requests for a particular content, the source starts transmission immediately; however, since the client wants the playout to start only after a specified time, the extra time is leveraged to deliver contents at a higher encoded rate, enhancing the quality of reception at the client.

1.2 The problem: An overview

Scalability is an inherent motivation for our research. In our past experience in administering a distance education program, the main stumbling block to scalability was the low bandwidth links in the path from the source to the clients. We realized that in order to scale the program to reach remote universities within India, we needed an innovative approach that would overcome bandwidth bottlenecks. Using the notion of delay tolerance, we not only utilize the resources efficiently but also achieve scalability, to reach the contents at acceptable quality to clients. In this section, we define the context and provide a high level definition of the problem. We also provide a discussion of the parameters that affect the problem to outline the scope of the solutions presented in this thesis.

1.2.1 The context

Traditionally, Content Delivery Networks (CDNs) [4] serve multimedia contents to clients through proxy servers deployed at strategic locations at the edge of its core network, as shown in Figure 1.2. Typically, a central server which is a content repository and a set of replica servers which contain replicated contents, form the *core network* of the Content Service Provider (CSP). With the tremendous growth in multimedia applications and the number of clients accessing such applications, a replica server itself may serve clients connected to it through a multi-hop distribution network of heterogeneous links. We focus on such a distribution network compris-

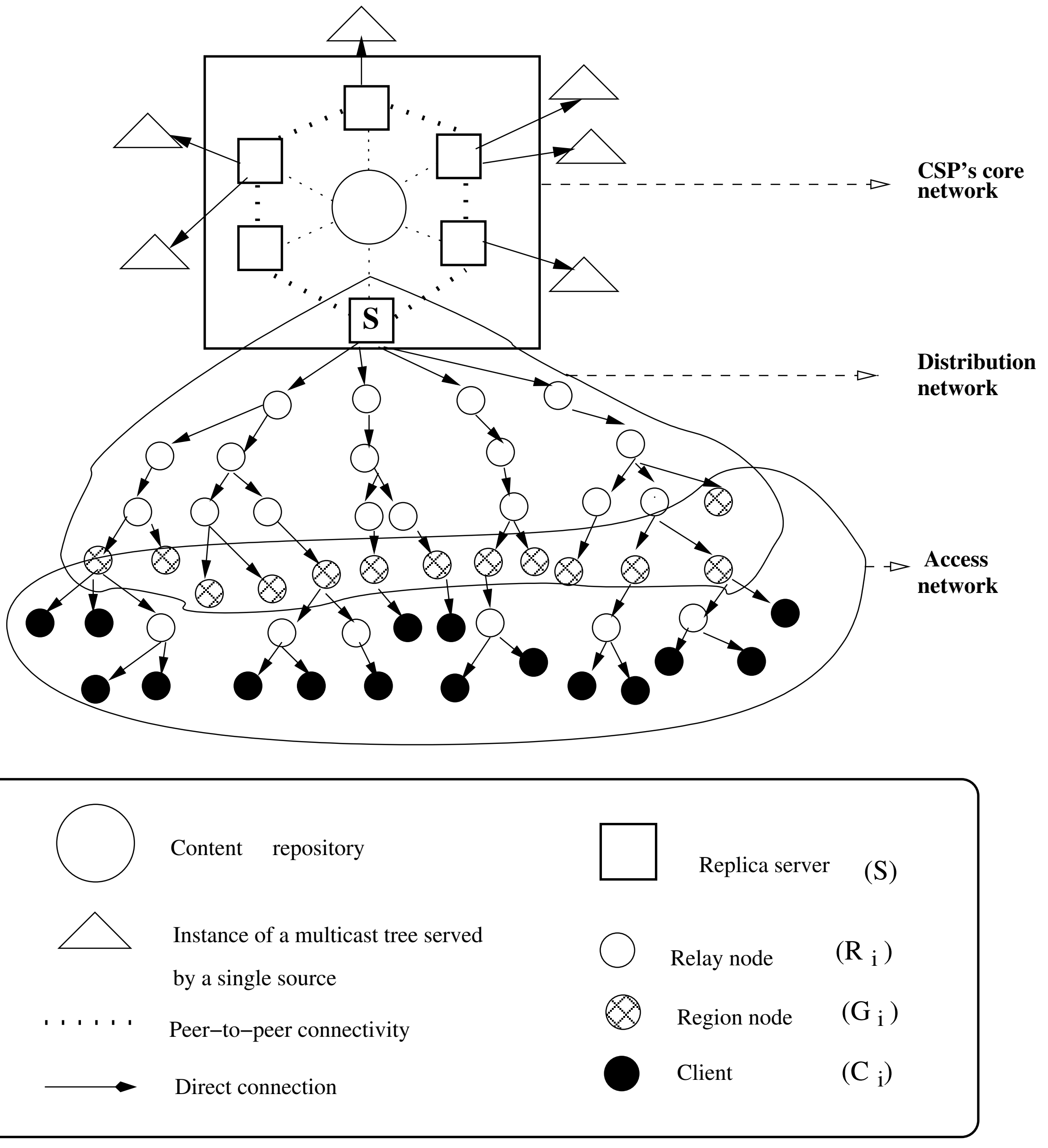


Figure 1.2: The context

ing of a replica server serving as a source for clients over a multicast tree.

Consider a multimedia stream originating at source $\mathcal{S}$, the root of the multicast tree and delivered to clients $C_1, C_2, \dots C_n$ connected through relay nodes $R_1, R_2, \dots R_i$. Without loss of generality, we define the multicast tree to be made up of the following two components: (i) Distribution network and (ii) Access network. We define the distribution network and access network with reference to Figure 1.2.

- **Distribution network:** This is a multicast tree with the source $\mathcal{S}$ as root and the *region nodes* G_is as leaves. We assume that this network is controlled by the CSP through

Service Level Agreement(SLA) with the Internet Service Provider (ISP). Typically, a specified amount of bandwidth is allocated for a given application on every link in the distribution network; we refer to this situation as *provisioned links* or simply refer to the link being *provisioned*.

- **Access network:** This network is a forest with each region node G_i as root and the clients C_is as leaves. Typically bottlenecks occur here. CSP may not have control over the access network nodes.

  In our model, users on an intranet, within the same administrative domain or different administrative domains within an organization, may be connected through a local server to the CSPs network. Such local servers, and not the users themselves, are the clients seen by the source. The local server places requests for contents on behalf of the users in the organization. The contents from this server can be accessed by several users simultaneously in the multicast mode as high bandwidth connectivity exists between different administrative domains and users within the organizations network.

- **Region nodes**: Region nodes G_is connect the distribution network to the access network. These nodes are edge nodes in the distribution network; they are typically designated by the CSP.

1.2.2 Problem definition

Having defined the context, we provide a high-level problem definition in this section. A detailed problem definition is provided in Section 1.3.
Given:

- A multicast tree enabled with a specified set of resources such as transcoders, layer encoders, and streaming servers, with source $\mathcal{S}$ serving clients C_is;

- Each C_i specifies two requirements: (i) a minimum acceptable rate γ_i^{min} (in kbps) and (ii) delay tolerance δ_i (in seconds), the time it is willing to wait, once connection is established, for start of playout of content.

- Content has *playout duration* $\mathcal{T}$ and is encoded at *base encoding rate* Γ, the best rate at which the contents can be played out at any client.

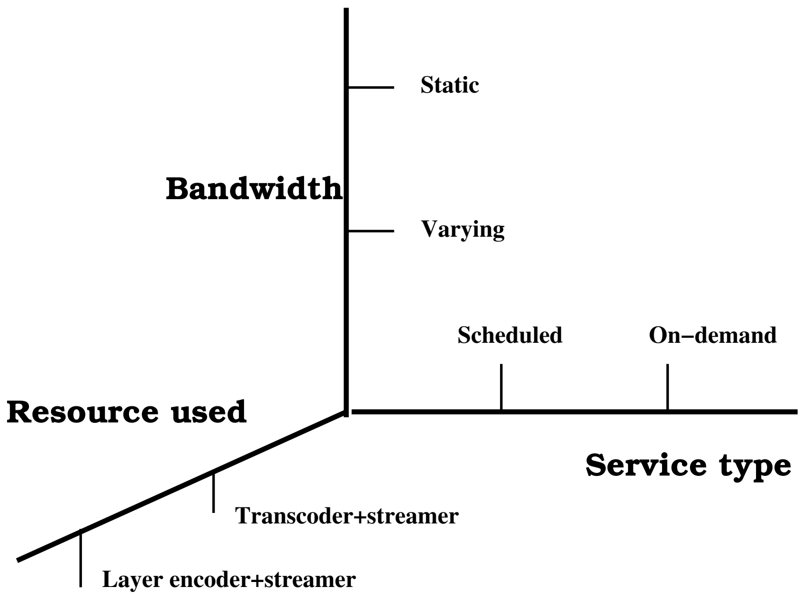


Figure 1.3: Parameters defining the dimensions of the problem

Assumptions: The topology of the distribution network and access network is known with link bandwidths remaining constant over a given prediction interval $\mathcal{P}$.

Objective: Leverage delay tolerance δ_i specified by each C_i to:

- provide continuous playout for each C_i at a rate r_i such that each r_i is greater than or equal to the minimum acceptable rate γ_i^{min} and
- maximize r_is across all C_is; i.e., minimize $\forall i, i = 1, 2, \ldots m$, $\sum_i (\Gamma - r_i)^2$, where m is the number of clients in the multicast tree.

1.2.3 The parameter space

We consider the following parameters which define the dimensions of the problem. The resulting parameter space is shown in Figure 1.3:

1. **Service types**
   - *Scheduled streaming*: In scheduled streaming, content from a source $\mathcal{S}$, is streamed synchronously to all the clients, starting at time t_0. When $\mathcal{S}$ is capable of transcoding or layering, each subtree in the distribution network may be served with a stream encoded at a different rate. However, the streaming session starts at t_0 for all clients.

- *On-demand streaming*: In on-demand streaming, each client may request for the contents at a different time; in this case the streaming is asynchronous. Suppose a client C_i requests for contents at time t_i specifying its requirements: a minimum acceptable rate γ_i^{min} and delay tolerance δ_i. As different clients (which may share links in their paths), request for the contents at different times, the shared links may be busy servicing a client request when another client from the same subtree requests for the contents. Thus, streaming from the source $\mathcal{S}$ starts when the links from $\mathcal{S}$ to C_i are free. Note that playout of the contents encoded at least at γ_i^{min} must start at C_i at time (t_i + δ_i).

2. **Link Bandwidths**

- *Known and static*: In this case, link bandwidths are assumed to be known and static over the period starting from when a client requests for transmission to end of playout at the client, defined as the *connection duration*, given by $(\delta_i + \mathcal{T})$.
- *Varying and predicted*: In this case, link bandwidths are assumed to be varying over the period starting from when a client requests transmission to the end of playout at the client. However, we assume a bandwidth profiling module which estimates bandwidths available over several prediction intervals that cover this period. We assume these predicted bandwidths to be static over each prediction interval.

3. **Resources used**

- *Transcoder*: Transcoder is a resource that encodes multimedia contents to a lower bit rate from its original encoded rate without changing the data format. This process of lowering the bit rate is also known as **transrating**.
- *Layer encoder*: Layer encoders encode the multimedia contents into a number of discrete layers. There is a base layer which is the lowest encoded rate and a number of enhancement layers which can be decoded and used in conjunction with the base layer for enhanced quality.
- *Streaming server with transcoder/layer encoder*: A streaming server is a server that is capable of sending a multimedia stream at a specific encoded rate. In our analysis, we assume the streaming server to have transcoding or layering capability. Streaming servers are used only in conjunction with on-demand streaming.

Thus, given the three parameters each taking two values, we have eight possible combinations to consider. However, as we explain below, the resources used dimension is handled based on the service type.

Combining the parameters

Given that our main objective is to maximize the delivered rates at clients by leveraging their delay tolerance, either of the resources – transcoders or layer encoders – can be used to adapt the encoding rate of the stream to deliver appropriate rates to clients. When link bandwidths are varying, use of transcoders may involve high overheads and loss of video quality due to the following reasons: (i) since any link in the client path can be the weakest link over the session duration, transcoders would be required at every relay node in the multicast tree (ii) in bit-rate reduction transcoding, even though resolution degradation is negligible, successive transcodings introduce degradation, characterized as a generation loss [40]. When link bandwidths are varying, since each node may be transcoding the stream to several different lower rates over the session duration to serve the clients, this could reduce the quality of the resulting video due to generation loss. Thus, transcoding is not an efficient solution when link bandwidths are varying. Layered encoding where layers are coded so as to serve most clients with thei optimal rates is an efficient alternative. Transcoding standards such as MPEG-4 [26] have integrated multi-layered coding.

Based on the above discussion, we choose a resource: transcoders when the link bandwidths are static and layer encoders when link bandwidths are varying, to formulate our solutions. When client requests are on-demand, streaming servers are placed at appropriate relay nodes to utilize the available bandwidth on the links efficiently. Here again, streaming servers having transcoding capability are used when the bandwidths are static; When bandwidths vary over the session duration streaming servers with layer encoding capability are used. Thus, we come up with the following four combinations of parameters:

- *Scheduled streaming, Known and static link bandwidths*
- *Scheduled streaming, Varying and predicted link bandwidths*
- *On-demand streaming, Known and static link bandwidths*
- *On-demand streaming, Varying and predicted link bandwidths*

Note that when the CSP has complete control over the distribution network and the access network, it can provision all the links in the network; in such a case, the link bandwidths are *known and static*. Even when the CSP does not control the access network, in a subscription-based network, it is reasonable to assume that the CSP would have enough knowledge to predict the bandwidths on the links over specified intervals of time.

1.3 Overview of contributions and solution approaches

In this thesis, we have identified multimedia applications which we have termed *delay-tolerant applications*; in these applications, delivering the content with acceptable quality is given more importance than delivering the contents in real-time. Given the nature of the contents in these applications (which are relevant for a period of time), it makes lots of sense to use the delay tolerance of clients to improve the quality of reception at the clients. This idea is useful to the current CDN scenario, especially in the developing countries, where bandwidth bottlenecks occur along the path to a client; even when the CSP provisions links in its distribution network, the *last-mile* problem persists.

Thus, our first contribution in this thesis is the idea of delay-tolerant applications. Applications such as distance education and corporate training readily fit the specifications for delay-tolerant applications. We explore various properties of such applications, given the context and parameters that affect such applications. We also specify the architecture of nodes required to support such applications.

We start with the simple case when all link bandwidths are static to understand the impact of various parameters on the delivered rates at the clients and resource requirement and placement. Then, we relax the assumption of static bandwidths to include variability in bandwidth over prediction intervals spanning the session duration. We develop algorithms that would find the loss-free delivered rates at the clients when bandwidths are varying. Given that the contents are valid over a period of time, subscribed clients may be able to access the contents over this validity period. Considering this on-demand streaming case with static link bandwidths, we attempt to maximize the number of clients serviced and the delivered rates at the clients, given an arrival schedule.

In summary, we have developed and implemented the algorithms that can be used by the CSP in a tool to make decisions on service quality and resource requirement and placement. We have developed solutions for the various sub-problems identified and demonstrated their usefulness in aiding a CSP to administer delay-tolerant multimedia applications.

We now consider the four combinations discussed in the previous section and briefly outline our solution approach to problems identified under each case.

1. **Problem 1A: Scheduled streaming, static link bandwidths**

   - *Determining optimal rates delivered at clients for a given transcoder placement option:* Our objective is to develop solutions to find the optimal rates delivered at clients given that a specific set of nodes have transcoders. We first formulate this as an optimization problem, and show that the optimization-based approach is computationally expensive. We develop an optimal linear-time algorithm *find_opt_rates_I*, that exploits the properties of the multicast tree to find the optimal delivered rates at clients, for a given transcoder placement option.

   - *Determining optimal number of transcoders and their placement for providing best delivered rates at clients*: When all relay nodes are transcoder capable, algorithm *find_opt_rates_I* determines the stream rates flowing through each link in the network and the optimal rates delivered at the clients. We call these as the *best delivered rates*, when there is no constraint on the number of transcoders. Even though all relay nodes have transcoders, all of them are not enabled to serve the clients with the best delivered rates. Our objective is to find the minimum set of transcoders referred to as the *optimal set*, denoted by $\mathcal{O}$, required to serve the clients with the best delivered rates.

   - *Determining optimal placement for a given number of transcoders:* In practical scenarios, resources available are limited. Given a number of transcoders $q < |\mathcal{O}|$, we know that the delivered rates at some clients would be less than their best delivered rates. Our objective is to place the transcoders such that the decrease in delivered rates across clients is minimal; i.e., given a limited number of transcoders, we find the placement option that delivers the best possible rates to clients. We formulate this as an optimization problem and also develop optimal algorithms that consider combinations of nodes to find the best placement. We experimentally show that the computation time required by the optimal algorithms render them infeasible in practice. This motivates us to develop two greedy alternatives: a *max-gain* algorithm and a *min-loss* algorithm and evaluate these algorithms to demonstrate their effectiveness.

2. **Problem 1B: Scheduled streaming, varying link bandwidths**

   - *Determining optimal rates delivered at clients when the link bandwidths are varying*: When link bandwidths vary over the session duration, we assume a bandwidth prediction module that provides an *advance estimate* of the available link bandwidths over prediction intervals spanning ($\mathcal{T}$ + max(δ_i)), where δ_is are the delay tolerance values of the clients. We formulate this as an optimization problem. Given the exponential complexity of the optimization approach, we explore different solutions based on the nature of bandwidth variation. We start with simple ones that convert the problem to the static bandwidth case. Subsequently, using the insights gained from the shortcomings of these solutions, we develop a link-by-link algorithm to find the delivered rates at the clients, considering the flow of the stream through each link.

   - *Adjusting stream rates if revised bandwidth estimates are available*: Finally, we develop an approach to exploit more precise estimates that may become available at the beginning of each prediction interval. We show how to use the *revised estimate* of link bandwidths, if available at the beginning of every prediction interval, to adjust the delivered rates at the clients. We show that this algorithm is effective in exploiting the available bandwidth on the links over each prediction interval compared to just using the advance estimate.

3. **Problem 2A: On-demand streaming, static link bandwidths**

   - *Determining optimal rates delivered at clients when the clients request for contents over a period of time:* We assume an observation period over which client request arrivals are monitored. Network characteristics and client requirements are given. Given highly provisioned links in the distribution network, we propose an algorithm that combines data transfer and streaming mechanisms to efficiently handle requests for the same contents over a period of time. Our goal is to maximize the number of serviced clients and the delivered rates at the clients.

   - *Determining appropriate placement for streaming servers:* Given the arrival pattern of client requests over the observation period, streaming servers have to be placed at appropriate network nodes in order to maximize number of clients serviced. Using observations relating to the network characteristics that affect data transfer and streaming, we develop rules for placement of the streaming servers. These rules are applied in the algorithm for finding the number of serviced clients and the rates delivered at the clients.

4. **Problem 2B: On-demand streaming, varying link bandwidths**
   Determining optimal rates delivered at clients over a observation period, given the network characteristics and an arrival pattern, when the distribution network links are highly provisioned and access network links have varying bandwidths: Using the insights gained from the analysis for solving Case 1B and Case 2A, algorithms can be designed for this case; however, detailed analysis is not in the scope of this thesis.

Lastly, given that all our algorithms require buffers at the network nodes, we discuss the implication of buffers, considering the special case when a client device is memory constrained. This discussion is relevant to clients joining the network using mobile phones or other hand-held devices which may have limited memory.

1.4 Organization of the thesis

The rest of this thesis is organized as follows: In Chapter 2, we identify the factors that affect multimedia dissemination and present several real-world examples that fall under delay-tolerant applications. We then present a literature survey of concepts and mechanisms that are relevant to servicing delay-tolerant applications. In Chapter 3, we present the system model and a detailed definition of the problem. We begin the chapter with definitions and key assumptions used in this thesis. We then trace the flow of data for the scheduled and on-demand streaming applications; we conclude the chapter with an overview of architectures for the different types of nodes in the multicast network. We present solutions to the three sub-problems identified under problem 1A, for scheduled streaming, static link bandwidths in Chapters 4, 5, and 6. In Chapters 7 and 8, we deal with the sub-problems identified for scheduled streaming, varying link bandwidths and On-demand streaming, static link bandwidths, respectively. In Chapter 9, we discuss another important resource required for servicing delay-tolerant applications: buffers. In this chapter, we analyze the size of buffers required at various nodes for servicing delay-tolerant applications and also discuss the case of client devices with limited memory. We identify future work required in this area and present a schematic for a CSP tool which can be built using the algorithms presented in this thesis, in the concluding Chapter 10. Such a tool would help a CSP in making decisions on resource requirement, deployment, scheduling, and admission control to maximize the efficiency of servicing delay-tolerant applications.

Chapter 2

Examples of delay-tolerant applications and literature survey

2.1 Introduction

Increasing popularity of applications such as live broadcasts of events, streaming stored movies, video games, video conferencing, and distance education implies that a large amount of multimedia content is being disseminated to users scattered at different locations. Several factors influence the design and implementation of techniques for the dissemination of multimedia contents. These factors are:

- *Network topology*: The underlying network can be Intranet, Internet or heterogeneous; An Intranet can comprise of a LAN, satellite network, or a network which combines different network technologies, but typically controlled by a single administrative domain. Internet is a public network, which provides best effort service to all the applications sharing the resources. A heterogeneous network is a combination of private and public networks and thus can include interconnected Local Area Networks (LANs) and Wide Area Networks (WANs). Mechanisms needed for effective and efficient dissemination of multimedia vary based on the characteristics of the network, such as, resource availability, nature of links, link access mechanisms, etc. [15] [21].

- *Encoding mechanism*: Multimedia files are huge in size compared to data files. Several standards exist for efficient coding of the multimedia data to reduce their size. Moving Picture Experts Group (MPEG) [26] has defined a family of standards for coding audiovisual information, e.g., MPEG-1, MPEG-2, and MPEG-4 [26][46]. Encoding techniques generally trade off between the file size, resolution quality, and the complexity of the decoding algorithm. Based on the nature of the multimedia content, appropriate algorithms can be chosen that strikes a balance between these three factors.

- *Delivery mode*: Typically, the following delivery modes are used while streaming multimedia:

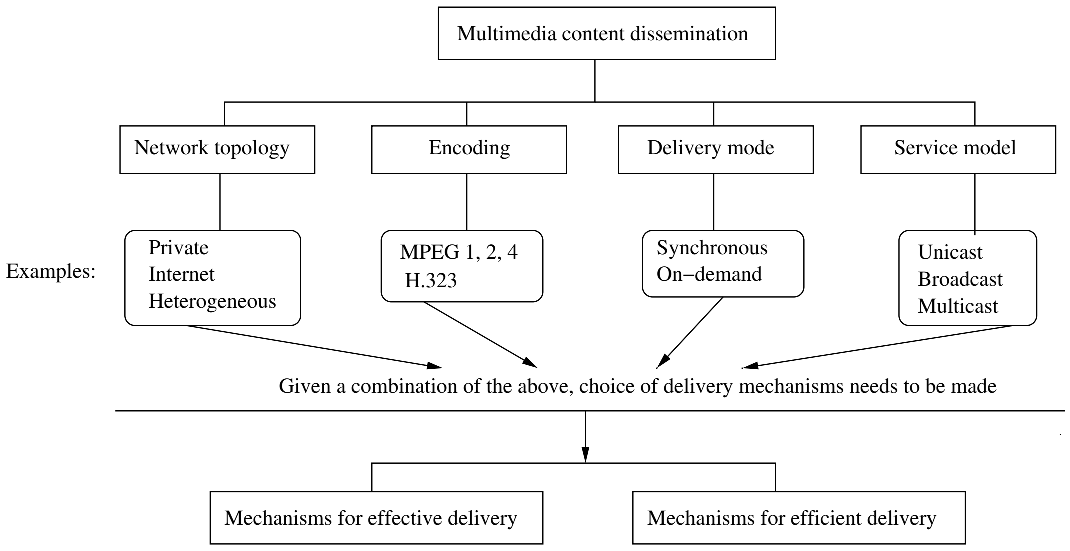
  - Synchronous mode: where a streaming session is multicast to multiple receivers receiving the stream simultaneously. There are two ways in which synchronous transmission can happen: (i) from a live source: Here a live media stream is encoded on the fly and transmitted to receivers in real time, or (ii) from a stored medium: Here the media stream is played from a recorded source (e.g. tape, CD).

    In both cases, broadcast is synchronous and the clients receive the transmission simultaneously. In the rest of this thesis, we refer to transmission in the synchronous mode as *scheduled streaming*.

  - On-demand mode: where the media files are typically stored and served to clients as and when they request for the contents. We refer to transmission in this mode as *on-demand streaming*.

  The mode of transmission places different requirements on the system and network resources. For example, for the live transmission of sporting events it is desirable to provide end-to-end guaranteed minimum bandwidth, for providing continuous and jitter-free playout; Synchronous transmission from a stored medium and on-demand delivery require storing of the multimedia contents, introducing server and storage management issues.

- *Service model*: Service model can be unicast, broadcast, or multicast. Unicast of content involves individual connections between the source and each of the receivers while a broadcast can be received by any receiver on the network. Typically a multicast model is assumed while delivering multimedia contents to multiple clients, as serving the contents to individual receivers with a unicast stream will quickly deplete the network resources. Details of a multicast model and its associated protocols can be found in [28][31].



Multimedia content dissemination
Network topology
Encoding
Delivery mode
Service model
Examples:
Private
Internet
Heterogeneous
MPEG 1, 2, 4
H.323
Synchronous
On−demand
Unicast
Broadcast
Multicast
Given a combination of the above, choice of delivery mechanisms needs to be made
Mechanisms for effective delivery
Mechanisms for efficient delivery

Figure 2.1: Factors that affect multimedia content delivery

Given any combination of these factors, the mechanisms used for the delivery of multimedia contents must be:

- Effective, i.e., guarantee a minimum quality of reception to ensure a smooth playout of the multimedia files.
- Efficient, i.e., optimally use network and system resources, allowing for scalability.

Thus, the choice of mechanisms for effective and efficient delivery of multimedia becomes important. These factors that affect the dissemination of multimedia are summarized in Figure 2.1.

A survey of adaptive Internet multimedia applications can be found in [51]. We have provided a classification of the mechanisms for effective and efficient delivery and discussion of relevance of these mechanisms in heterogeneous network environments in [14]. Given the vast literature in the area of multimedia dissemination, in this chapter, we map our research to a real-world problem that fall under the delay-tolerant application category and discuss existing mechanisms that can play a role in developing the solutions for this application. We provide a comprehensive survey of the concepts and mechanisms, discussing their relevance to this thesis.

2.2 Corporate training

Before we present the model for this application, we recap the definition for delay-tolerant applications: Delay tolerant applications are those applications where, a client C_i connects to the source $\mathcal{S}$ at time t_0 and requests for contents with base encoding rate Γ ; C_i specifies the time it is willing to wait for the playout to start, its delay tolerance δ_i; Playout at C_i starts at (t_0+δ_i) and ends at $((t_0 + \delta_i) + \mathcal{T})$, where $\mathcal{T}$ is the playout duration of the contents.

Consider a corporate network of a Multi National Company (MNC) as shown in Figure 2.2. Note that this figure represents the same network as given in Figure 2.2 which defined the context for our problem. We explain the mapping between the two figures below:

- A central office responsible for content generation and dissemination is the content repository.
- The central office is connected to other country offices through a P2P overlay network. Each of the country offices has local content repositories serving different training centers located in the country.
- Our focus is on the network served by each country node. Training centers are connected to the country node, denoted by $\mathcal{S}$, over a multicast overlay. Several streams can be served from $\mathcal{S}$, each stream served to a set of training centers; each stream is disseminated through a multicast tree. A training center may be receiving more than one stream, hence part of different multicast trees. Note that the links are typically provisioned for each stream originating from $\mathcal{S}$.
- Clients access training sessions through:
  1. a Local Area Network (LAN) within the training center: this would be a high bandwidth link.
  2. an intermediary node via a leased line or dial-up modem: in this case the last link could have low bandwidth.
  3. a wireless device: in this case, while the link bandwidth may be high, the client device may be memory constrained, affecting buffer availability; this motivates us to look at memory-constrained client devices as a special case.

We summarize the characteristics of such a network below:

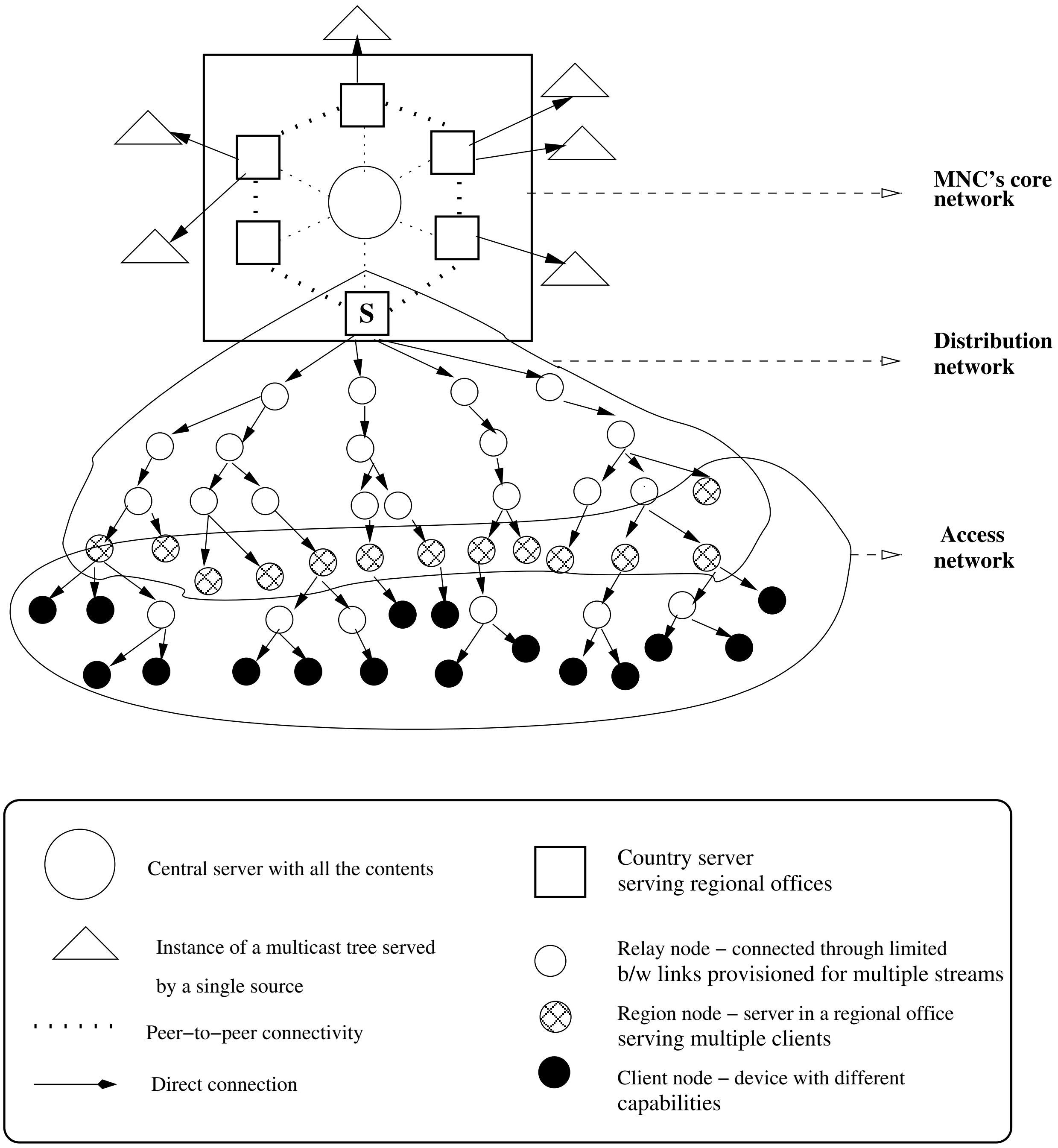

S
MNC's core network
Distribution network
Access network
Central server with all the contents
Country server serving regional offices
Instance of a multicast tree served by a single source
Relay node − connected through limited b/w links provisioned for multiple streams
Peer−to−peer connectivity
Region node − server in a regional office serving multiple clients
Direct connection
Client node − device with different capabilities

Figure 2.2: Example: Corporate training

- *System model*: Country office is the single source; training centers within the country are region nodes connected through heterogeneous links. Clients can be directly connected to the training center or through an access network. Unlimited buffers are assumed at all relay nodes.

- *Operational specifications*: Each training center subscribes to courses based on client registrations. Suppose there are 8 courses disseminated from the country center. Note that a subset of the training centers would subscribe to one or more of these courses. Also, at each training center different clients may subscribe to different courses. Thus, based on the subscription from each training center for each course, different multicast trees emerge.

- *Transmission characteristics*: Scheduled transmission to subscribed clients having requirements: minimum rate, delay tolerance. Note that delay tolerance of training centers could be specified based on factors such as work schedule, being in different time zones, or improving quality due to low bandwidth connections.

In the above scenario, the addition of the following requirement necessitates a solution involving on-demand streaming:
In addition to the scheduled streaming sessions, each lecture has an accessibility period of, say 4 hours. Note that accessibility period for each lecture is predetermined. During this period, clients can access the lecture at any time, to review the lecture. Now, we have a case for on-demand streaming for the same contents. Note that each client C_i is connected to the source $\mathcal{S}$ for a period: $((\mathcal{T} + \delta_i)$ - $t_b)$, where t_b is the time during which links in C_i's path may be busy serving some other client.

2.3 Accessing content through a wireless connection

In this section, we outline an existing multimedia application and explain how by using the concept of delay-tolerance, this application can be tied to a CSP as in the example discussed in the previous section. Such a step would allow for the following benefits: (i) a regional repository of a CSP can be used for serving clients instead of maintaining media servers at each location, (ii) variety of contents could be stored at the regional repository, allowing for numerous choices of programs to customers, and (iii) utilize the time (of travel in the example that we present) to compensate for lower bandwidths available on the CSP's network.

Consider the multimedia applications, offering service to travelers during the time they wait for their flights. In such applications, popular multimedia programs such as movies and videos can be made available through access points, to customers with laptops or other hand-held devices. Note that managing such a service at each airport would be an expensive activity. With the current trend of acquisitions and diversification, where airports are acquired and operated by MNCs (especially, in a country like India), a Closed User Group network such as the one we described in the previous example can be extended to service customers at airports also.

Suppose a client connects to the airport portal and places a request for a particular media content, specifying the time after which the customer would start viewing the contents. Note that as a customer leaves for the airport, she/he can estimate the time when they would arrive at the airport and specify this estimated travel time as their delay tolerance. Instead of having huge storage systems at each airport where contents are stored, customer's delay tolerance can be transferred to the access point which would acquire the required data from the appropriate content repository. Note that the access point is a streaming point with a limited cache facility similar to the model we presented for the on-demand streaming case. The connection from access point to end device is a high bandwidth link. Thus, as the customer reaches the airport the contents can be accessed in real-time through an access point.

- *System model*: Client subscribes to the content delivery service through an appropriate portal and requests for a particular content at the time of leaving home for the airport, considering the time it takes for travel to the airport, specified as delay tolerance. The receiving point at the airport is termed *access point*. The last link, a wireless connection at the airport from the access point, has high bandwidth. The client device may be memory constrained, so playout happens from contents cached at the access point.

- *Operational specifications*: CSP has a menu of contents to choose from. Based on the number of requests for a particular content over a period of time, CSP can cache the contents at an appropriate relay node or at the access point itself. CSP may use a caching policy which uses client access information over a specific period of time to decide on which contents to keep in the cache and which to evict.

- *Transmission characteristics*: Transmission can be subjected to two parts:

1. The client's delay tolerance is inherited by the access point; Contents are streamed to the access point at the best possible encoded rate (given the delay tolerance).
2. When the client gets to the airport and connects to the access point, contents are streamed in real-time from the access point.

Having discussed some real-world applications which fit the profile of delay-tolerant applications, we now identify concepts and mechanisms that can be used in the implementation of these applications, from the existing literature.

2.4 Concepts and mechanisms relevant to servicing delay-tolerant applications: A literature survey

Before we examine the relevance of existing mechanisms, we discuss *Delay-Tolerant Networks* (DTNs) which may sound similar to the concept of *delay tolerance* which is the key idea behind this thesis. A delay-tolerant network is a network designed to operate effectively over extreme distances such as those encountered in space communications or on an interplanetary scale [37]. It can be thought of as a network of regional networks; an overlay on top of regional networks, including the Internet [52]. One of the problems that needs to be addressed in a DTN is the long and variable delay, and hence its name. We define delay tolerance to be the time that a client is willing to wait for playout of the multimedia to start. The client requires that the playout be continuous and loss-free, once playout starts. Thus, while DTNs deal with network delays (of the order of hours and days) through a store-and-forward mechanism, we deal with clients with a diverse set of delay-tolerance values (of the order of seconds and minutes); delay tolerance values are leveraged to improve the quality of delivered contents at the clients using buffering capabilities of the nodes. Since buffers can be co-located at the routers or L4 switches [13], we believe that our model is feasible in practice. We elaborate on the issue of buffering in Chapter 3, where we present detailed architecture of the nodes.

The work in [1] deals with broadcasting multimedia products in the spare bandwidth available in a television channel. Customers can be provided with a set of delivery options corresponding to the time of delivery, which is similar to the notion of delay tolerance. This work focuses on maximizing profits for the e-commerce merchant. While we are also motivated by the goal of revenue maximization for the CSP, in this thesis we deal with transcoder placement strategies – placement at relay nodes, bit rate conversions they need to perform – to enhance the delivered rates at clients in a heterogeneous multicast environment.

In the rest of this section, we identify existing concepts and mechanisms that can be exploited in the design and implementation of techniques for the dissemination of multimedia contents to delay-tolerant clients. We review the literature discussing the relevance and usage of these in our work.

Multimedia dissemination is a well-researched topic with many mechanisms proposed for effective and efficient distribution of multimedia data [51][14]. As mentioned, most of the existing literature treats multimedia applications as soft real time applications where playout of the content begins as soon as the client requests for the data. Typically, research has focused on reducing the start-up latency – a small time lag due to buffering at the client to control jitter during the playout. In contrast, we are focusing on delay-tolerant multimedia applications, where client specified delays are used to improve the quality of the multimedia contents delivered.

2.4.1 Network structure

Considering the context of our thesis, our focus is on content delivery networks such as Akamai [4]. As illustrated in Figure 1.2, a CSP's core is a mesh network with peer-to-peer (P2P) connectivity between nodes. Note that the P2P connectivity could be *structured* where Distributed Hash Tables (DHT) are used to identify peers or *unstructured* which are ad-hoc in nature. We refer the reader to [22] for an overview of P2P overlay network schemes.

The distribution network of the CSP is a multicast tree with one of the replica servers in the CSP's core network serving as source to clients distributed in that region. Again, multicasting is a highly researched topic with many protocols and techniques proposed [28][31] for its efficient implementation.

IP multicast remains popular with researchers but has failed to take off in general, except for the limited domain of IPTV. Difficulties with IP multicast - capacity planning and traffic engineering have discouraged ISPs from adopting IP multicast in a big way. IP multicast can be used to deliver the same stream to multiple users (as in the case of IPTV, specifically live TV broadcast), where all clients in the multicast tree have the same requirement. In our case where each client specifies a delay-tolerance requirement, the stream has to be modified to serve each client with its best possible encoded rate. This can be achieved by an Application Level Multicast(ALM) implementation.

The lack of large scale deployment of IP multicast has led to ALM systems that are built using the structured P2P overlays. Any of the different P2P overlays and different implementations of ALM [6] can be used in the delivery networks used in our analysis. Note that in our ALM implementation, unlike the traditional implementation, each client gets the contents encoded at an appropriate rate, taking into account the bandwidth constraints and delay tolerance requirement of the clients.

2.4.2 Transcoding mechanisms

Typically transcoding refers to a process that converts media from one format to another. Transcoding functionality also includes reducing the encoding rate of a media stream, also referred to as *transrating*. For our analysis, we require only the transrating functionality at appropriate relay nodes in order to serve the clients with different delivered rates. As discussed in Section 1.2.3 in Chapter 1, we use transcoders when the link bandwidths are static and layer encoders when the link bandwidths are varying. We assume the following two properties for the transcoders:

- *Pipelining property*: When a transcoding enabled relay node is a RTP/RTSP client as in [35][36], as the bits are streamed, they go into an in-buffer (large enough to hold bits required for transcoding) and the transcoded bits are pushed into the out-buffer. This process is referred to as *pipelining*. The processing rate is defined as the number of bits processed by the transcoder per second. As long as this rate is more than *Min(bandwidth of the incoming link , bandwidth of outgoing link)*, there is no significant latency introduced due to the transcoding process [40]. Thus, the transcoding process can be assumed to be *real-time*. Also, compressed domain transcoding [42] can improve the speed of transcoding as it reuses the motion information. Thus, we include a small constant transcoding latency, the worst-case delay that may result from deploying transcoders in the relay nodes.

- *Concurrent-processing property*: A transcoder is capable of transcoding a single incoming stream encoded at r_i to multiple outgoing streams simultaneously, having different rates $r_j, r_k, \ldots$, where each outgoing rate is less than or equal to r_i. The advances in transcoder technologies have resulted in many products that are capable of real-time transcoding, such as Dynamic Bandwidth Manager (DBM) from RGB [34] and Explorer II card [7]. Using compressed domain based approaches concurrent rate adaptations can be efficiently performed. In compressed domain adaptation, the incoming video is only partially decompressed; rate adaptation is performed in this compressed domain while the motion information is reused [42]. Compared to the conventional decoding-transcoding-recoding approach, this approach improves the speed of rate adaptation considerably.

2.4.3 Layering mechanisms

When the link bandwidths are varying, transcoders would be required at every relay node serving multiple clients, in order to serve them with appropriate rates. A better mechanism would be to encode the contents in layers such that based on the available bandwidth, clients add appropriate layers. Typically, a *base layer* is mandatory which contains the crucial media data and the *enhancement layers* are add on to the base layer, that improve the quality of playout. Note that (i) the maximum number of layers is bounded and (ii) a minimum amount of data is required to construct an enhancement layer. Due to these two reasons, in layering, it is difficult to provide the exact rate (the optimal rate) to every client. There could be some under-utilization of link bandwidths.

There are two approaches to layering:

- *Sender-driven*: In this approach, the source is responsible for constructing the layers and adjusting the layers based on feedback on link bandwidths, from network nodes. Examples of algorithms that fall in this approach include the source adaptive multi-layered multicast (SAMM) [47][48][3][2] algorithms, the probabilistic feedback algorithm [5], and some unicast adaptive algorithms [10] [20].

  Sender-driven algorithms have the ability to adjust the encoding rates of the layers and the number of layers, using congestion feedback based on loss experienced by the client. This allows for the available bandwidth to be used efficiently. In our analysis for the scheduled streaming, varying bandwidth case, we assume an end-to-end source adaptive multi-layered multicast (SAMM) algorithm as discussed in [2]. Suppose bandwidths are varying over *prediction intervals* spanning the session duration, for each prediction interval based on the link bandwidths, the source can adjust the number of layers and the encoding rate of the layers to best serve the clients. One disadvantage of the SAMM algorithm is that the algorithm may negatively impact other competing flows, as the layers are priority marked. Given the context of our problem, a subscription based provisioned network controlled by a CSP, this is not a major concern. This is because, when link bandwidths are static, constant bandwidths are assumed to be provisioned for the stream for the duration of the session; when bandwidths are varying, bandwidths are assumed to remain constant over each prediction interval.

- *Receiver-driven*: In this approach, the source sends a fixed number of layers encoded at fixed rates; receivers perform *join experiment* to add new layers; if they experience loss, the experiment is a failure. Similarly, a receiver receiving multiple layers may have to *prune* itself from the distribution tree for a layer, when bandwidth drops. In this algorithm, receivers have to periodically conduct join experiments, and whenever loss is experienced, need to perform pruning. Examples of receiver-driven algorithms include: Receiver-driven Layered Multicast (RLM) [25], Layered Video Multicast with Retransmission (LVMR) [29], and TCP-like congestion control for layered data [49].

  In a best effort network, such as the Internet, receiver-driven algorithms can be deployed as they are friendly to competing data traffic. However, these algorithms have high overheads due to the join experiments and pruning.

2.4.4 Caching mechanisms

In our work related to on-demand streaming for the static link bandwidth case, we place streaming servers with transcoding capability at appropriate nodes, termed *streaming points*. We also assume caching capability at these streaming points.

Similar research has been done in the context of delivering contents encoded at appropriate format and rate to end users having different devices such as laptops and mobile phones, having different capabilities to receive multimedia data. Work presented in [39] [40] [41] use a Transcoding-enabled proxy system to achieve efficient delivery of multimedia contents to diverse end-users; Several caching algorithms are proposed. Note that in our work we propose a simple caching strategy to hold the content in cache for the duration of playout; if the content is accessed before expiry, its TTL (Time to Live of the content) is reset to the playout duration.

Any of the caching algorithms proposed in these papers can be used in our streaming servers. These algorithms are especially relevant to our work presented in Chapter 9, where we deal with buffer management issues when the client devices are memory-constrained.

2.4.5 Placement of resources within a dissemination network

There is a large body of work on the optimal placement of resources. We consider some of these, focusing on works that deal with resources such as transcoders and caches, and discuss the relationship between them and our solutions.

In our work, in scheduled streaming, static link bandwidth case, we find the placement for a given number of transcoders, when the number of transcoders is limited. Our objective is to place the transcoders such that the delivered rates at the clients are maximized. A related problem is addressed in [9]. This work shows that in a multicast streaming environment, performing transcoding at intermediate nodes is more efficient than transcoding at the source or clients. In this respect, this work justifies our claim that placement of transcoding capability at appropriate relay nodes would increase the effectiveness of the resource usage.

The objective in [9] is to optimally multicast a media stream while delivering it in a suitable format to each client. This translates into finding optimal placement for transcoders in a multicast tree, considering the costs involved. Three cost components considered are: (i) Communication cost of transmitting the streams over the network with appropriate formats, (ii) Cost of the transcoding operation, and (iii) Storage cost of the multimedia contents. The authors pose this as an optimization problem, formally described using a novel graph model termed *multi layered mixed graph* (MLMG) which is developed in order to better describe the flow of data over the heterogeneous network. Note that in our work, we consider rate conversions rather than format conversions. This is in line with our context, where a CSP is responsible for generating and disseminating content to its subscribed clients. Since we are dealing with a Closed User Group (CUG) with all client information available to the CSP, we do not worry about format conversions. However, in Chapter 9, we deal with resource constrained devices, which may require format conversions to adapt the contents to the device capabilities. Typically, format conversions are more expensive in terms of processing time compared to rate conversions; however, such delays can be taken into account by using appropriately reduced δ values, while computing the delivered rates at the clients.

In summary, [9] deals with the problem of transcoder placement at intermediate network nodes in order to serve the clients with appropriate format and rate. This work considers a graph and finds the path with minimal cost to decide placement of transcoders. In contrast, in our work, given a multicast tree, where the paths from the server to each client are known, our objective is to maximally utilize the available bandwidth and the delay tolerance to maximize rates delivered to the clients.

In [13], placement of caches at network nodes with the goal of reducing network traffic and server load is discussed. One of the cases considered is a multicast tree with a single server, which is the same as the multicast topology we consider. However, the objective of this work is to minimize the delay at the clients requesting contents. In contrast, we proactively utilize the delay specified by the client to maximize the delivered rates at the clients. Thus, the methodlogy used in [13] cannot be used in our context.

Authors of [45] deal with the problem of placing a given number of replicas within a network, to deliver content to clients within a specified delay bound. Whereas this delay relates to the time for downloading the contents, in our case, it connotes the delay until the start of playout of the continuous media content; once playout at the client begins at a certain rate, it continues without loss of content or reduction in quality till the end of the playout; Further, our objective is to place the transcoders such that the delivered rates at the clients are maximized. Even though in specific scenarios, for example, when playout duration is very short, i.e., when all the content is downloaded before playout starts, the two models converge, the problem we have at hand calls for solutions that explicitly take the delivery needs for continuous media into account.

2.5 Summary

In this chapter, we started with a discussion of the factors that affect multimedia dissemination. We discussed some real-world examples which fall under the delay-tolerant multimedia applications category to understand the concepts and mechanisms that are relevant. Given the vast expanse of research performed in this area, we have discussed existing literature relevant to the context of delay-tolerant applications and mechanisms that can be used in serving the clients in such applications.

Chapter 3

System model and detailed problem definition

We start this chapter with definitions of the terms and the set of key assumptions that are used throughout this thesis. We then present the problem definition in detail and trace the flow of data through the nodes, to understand the functionality required at the nodes to support delay-tolerant applications. This leads us to specify the architecture of the nodes required to service delay-tolerant applications.

3.1 Definitions and key assumptions

3.1.1 Definitions

Multicast tree: The network for multimedia dissemination with S as the source serving a set of geographically distributed clients C_is connected through intermediate relay nodes R_is is termed as the multicast tree. Multicast tree is denoted by Λ. Figure 3.1 illustrates a multicast tree and its different components.

Source: At the root of a multicast tree is the source, denoted by S.

Relay node: Any intermediate node in the multicast tree is termed relay node, denoted by R_i, where i is the numerical index of the node.

Client: The leaf nodes in the multicast tree are the clients, denoted by C_k, where $k = 1, 2, \ldots m$.

Nodes: The source, relay nodes, and clients make up the nodes in the network. Each node has

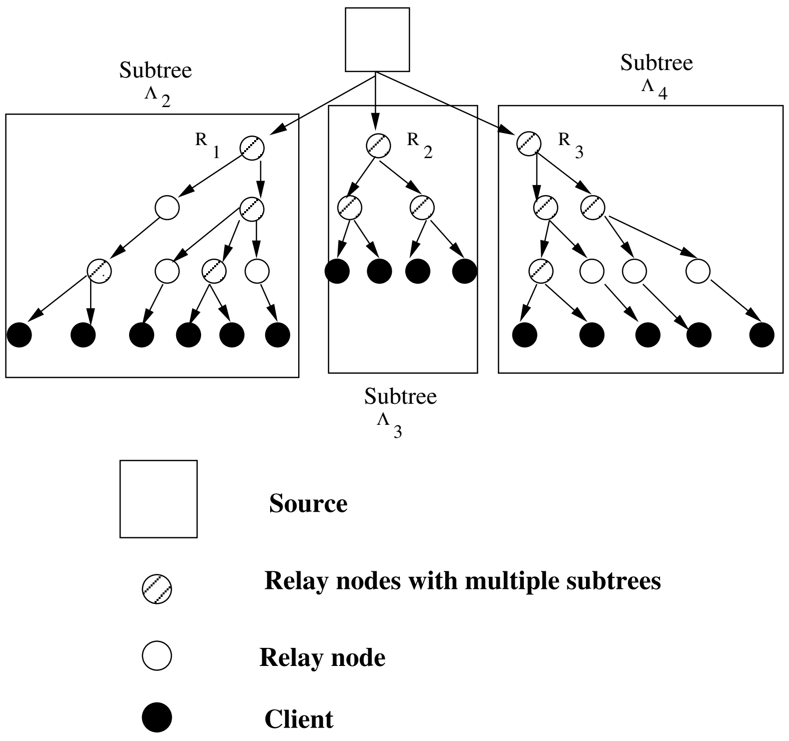


Figure 3.1: Multicast tree

a unique id.

Links: Links connect nodes to each other. A link that terminates at a node n_i is referred to as the *incoming link* of n_i and a link emanating out of a node n_i is referred to as an *outgoing link* of n_i.

Subtree: A subtree is a portion of a tree that can be viewed as a tree in itself [44]. Any node in a multicast tree Λ, together with all the nodes below it, comprises a subtree of Λ. The subtree corresponding to the root node $\mathcal{S}$ is the entire tree. Subtrees are referred to by Λ_m, where m is the node id of the root of the subtree. With reference to Figure 3.1, there are three distinct subtrees emanating from $\mathcal{S}$ which are marked as Λ_2, Λ_3, and Λ_4.

Path: A set of nodes and links from $\mathcal{S}$ to a given client C_i defines the path of C_i, denoted by $p(C_i)$.

Link bandwidth: Link bandwidth is the actual physical bandwidth available on a given link. Each link is assigned a unique numerical id. The link bandwidth on link l_i is denoted by b_i.

Network parameters: Network parameters, which are given as input to the algorithms developed in this thesis, refer a multicast tree (network topology) with specific link bandwidths.

Client arrivals: In the context of on-demand streaming, where clients request for contents at any time, the arrival pattern of client requests are referred to as client arrivals. This is a required input for the algorithms developed for on-demand streaming.

Base encoding rate: Base encoding rate, denoted by Γ, is the highest rate at which the stream is encoded and available at the source. In the context of layered encoding, base encoding rate determines the upper bound on the total encoding rate of all the layers.

Stream rate: Given a stream flowing through a link l_j, the rate at which the stream is encoded currently is referred to as the stream rate, denoted by r_j.

Maximum deliverable rate: Maximum deliverable rate is the stream rate that can be delivered to a client by considering its path from the source in isolation. For a given client C_i, its maximum deliverable rate is denoted by γ_i^{max}.

Delivered rate: The actual stream rate that is delivered at a client C_i (by considering other clients that share links in its path and other constraints such as the capability of a node in its path to transcode or not) is defined as its delivered rate denoted by γ_i. Note that γ_i at best is equal to γ_i^{max}.

Minimum required rate: Each C_i specifies its requirement for a minimum encoding rate for the stream, denoted by γ_i^{min}.

Delay tolerance: Each C_i specifies its delay tolerance requirement, the time it is willing to wait for the transmission to start, after connection establishment, denoted by δ_i.

Start of connection: For a given client C_i, the time t_i when the client connects to $\mathcal{S}$ and requests for content is defined as the start of connection.

Start of playout: The start of playout at each C_i is given by (t_i + δ_i), where δ_i is the delay tolerance of the client.

Playout duration: The time elapsed between the start of the playout till the end of the playout at a client C_i is the playout duration, denoted by $\mathcal{T}$. Note that irrespective of the rate at which a file is encoded, the playout duration remains constant.

End of playout: End of playout at a given C_i happens when the content finishes playing at C_i

given by $t_i^e = (t_i + \delta_i) + \mathcal{T}$, where $\mathcal{T}$ is the playout duration.

Connection duration: The time elapsed between the start of connection t_i till the end of the playout at a client C_i, $(t_i + \delta_i) + \mathcal{T}$, is the connection duration. This is given by $\delta_i + \mathcal{T}$.

Start of streaming session: For a given client C_i, the time t_i^s when transmission starts from $\mathcal{S}$ is the start of the streaming session.
Note that t_i^s is later than the start of connection t_i, when links in $p(C_i)$ are busy; When the links from $\mathcal{S}$ to C_i are free when C_i connects, t_i^s is equal to t_i. Also, t_i^s is earlier than the start of playout at C_i. Note that if t_i^s is greater than (t_i + δ_i), C_i can not be serviced as its requirement is violated.

Connection busy period: Suppose a client C_i connects at time t_i and requests for content. However, since link/s from the source in the path of C_i is/are busy serving another client, transmission starts from $\mathcal{S}$, say at t_j to C_i. Connection busy period is given by $(t_j - t_i)$.

Session duration: For each C_i the session duration is the time elapsed from the start of the streaming session t_i^s to the end of playout at $((t_i + \delta_i) + \mathcal{T})$. In other words,
session duration = connection duration - connection busy period$= \delta_i + \mathcal{T} - (t_i^s - t_i)$.

Prediction interval: A prediction interval is the duration of time over which bandwidths available on links in the network are assumed to remain constant. Note that the prediction interval spans the session duration when the bandwidth is assumed to be static/provisioned.

Consider an example where link 1 and link 2 are the links shared by clients C_1 and C_2. Suppose these two links are busy serving C_1, C_2 waits for these links to become free. Figure 3.2 illustrates the definitions related to connection and session. Table 3.1 and Figure 3.3 illustrate the actions along the timeline and the connection parameters for the two clients.

3.1.2 Key assumptions

We make the following assumptions:

- *Network topology is known and static.*

  We assume a subscription based network where all the nodes in the network are known. Examples include networks used for distance education, remote corporate training, movie

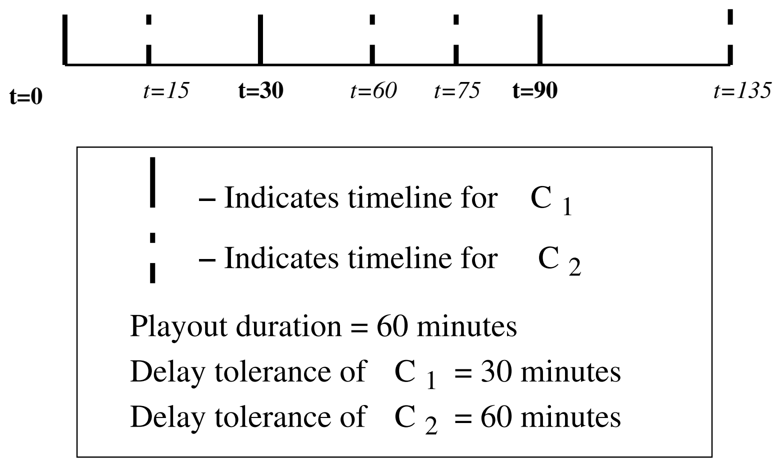

t=0
t=15
t=30
t=60
t=75
t=90
t=135
– Indicates timeline for C 1
– Indicates timeline for C 2
Playout duration = 60 minutes
Delay tolerance of C 1 = 30 minutes
Delay tolerance of C 2 = 60 minutes

Figure 3.2: Example: Illustration of definitions related to connection, session, and playout

Time	Action	Comment
t = 0	C_1 connects C_1's transmission begins	Start of connection for C_1 Start of streaming session for C_1
t = 30	Playout at C_1 begins	Start of playout at C_1
t = 90	Playout at C_1 ends	End of playout at C_1 End of streaming session for C_1 End of connection for C_1
t = 15	C_2 connects	Start of connection for C_2
t = 60	Links 1 and 2 are freed; C_2's transmission can begin	Start of streaming session for C_2
t = 75	Playout at C_2 must start	Start of playout at C_2
t = 135	Playout at C_2 must end	End of playout at C_2 End of streaming session for C_2 End of connection for C_2

Table 3.1: Illustration of timeline

Connection duration for C_1 = Session duration of C_1 = 90 minutes;

Connection busy period for C_2 =(60 − 15)= 45 minutes;

Connection duration for C_2 =(135 − 15)= 120 minutes;

Session duration of C_2 =(120 − 45)= 75 minutes;

Figure 3.3: Connection parameters

clubs, etc. Note that different streaming sessions may have different clients participating, thus the resulting distribution network or multicast tree for each session may be different.

- *Buffers are available at all network nodes.*

  We assume unlimited buffers at the network nodes for the analysis presented in this thesis. We outline the buffer management issues in Chapter 9.

- *The source $\mathcal{S}$ has all the information about the network parameters, client arrivals, and client requirements.*

- *All nodes have accurate information on the rate at which data is flowing into the node and the rate at which data is flowing out of the node through each outgoing link.*

- *All links in the multicast tree support the minimum rate requirement of every client admitted.*

  For all our analysis we assume that all n links in the multicast tree have a minimum bandwidth equal to the minimum rate required by any client in the tree. i.e.,

  $$\forall i, i = 1, 2, \ldots, n, \, b_i >= \gamma_i^{min}.$$

- *For the duration of a streaming session, each $\Lambda_{\mathcal{S}}$ (subtree rooted at $\mathcal{S}$) can have only one stream flowing through its links.*

  For the on-demand streaming, static link bandwidths case, we assume a *provisioned* distribution network, where bandwidths sufficient to support a multimedia stream is available on every link in the distribution network. While the bandwidth available on the links may be higher than base encoding rate of a stream, they may not be high enough to support multiple streams simultaneously. Given that bandwidth is a constrained resource, this assumption reflects reality.

- *Size of multimedia file is (Γ * $\mathcal{T}$).*

  Maximum size of a content file encoded at a rate Γ having playout duration $\mathcal{T}$ is given by: (Γ *$\mathcal{T}$). Note that this is the worst case assumption. In reality, the size would be lesser which may lead to under-utilization of the available bandwidths on the links.

- *End-to-end transmission delays are negligible.*

  We assume that transmission delay, propagation delay, and receiver delay which constitute the end-to-end transmission delay are negligible.

Typical values for end-to-end delays are of the order of msecs; Client delay tolerance values are typically several magnitudes higher, in the order of minutes. In practice, when these delays add up to a significant value, we can reduce the delay tolerance values of the clients appropriately to take these delays into account. The idea is as follows: The CSP estimates the end-to-end delay that would be experienced by the clients. This delay is subtracted from the delay tolerance values specified by the clients to produce updated delay tolerance values, using which delivered rates are calculated using our algorithm. If it is not possible to deliver the minimum required rate for any client, that client is not admitted for the session.

- *Source $\mathcal{S}$ has transcoding capability. This, as we shall see, is necessary to service clients with optimal rates without loss.*
- *Transcoders are capable of transcoding a stream into multiple streams at different rates simultaneously.*
- *Client delay tolerance requirements are known.*

When a client desires to join a session it sends a request with its requirements: a minimum acceptable rate γ_i^{min} and its delay tolerance δ_i, the time it is willing to wait for the transmission to start *while staying connected.*

3.2 Detailed problem definition

In this section, we consider the four combinations identified in Chapter 1, Section 1.2.3, and discuss the problems solved for each of these cases in detail.

3.2.1 Problem 1A: Scheduled streaming, static link bandwidths

We depict the solution approach for the *scheduled streaming, static link bandwidths* case in Figure 3.4.

For a given topology and client requirements, quality of playout at the clients can be improved by leveraging the clients' delay tolerance. Optimal rates delivered at clients depend on the (i) number of transcoders and (ii) options available for the placement of these transcoders.

Transcoder placement option: refers to the combination of nodes in the network which can have transcoders. Since $\mathcal{S}$ is always assumed to have a transcoder, the basic transcoder placement option is **Source only Transcoding (ST)**, where other than $\mathcal{S}$, no network node can have a transcoder. When all the relay nodes have transcoders, the resulting placement option is **Anywhere Transcoding (AT)**. When only a chosen subset of the relay nodes has transcoders, we term it as **Selected Node Transcoding (SNT)**.

1. *Determining optimal rates delivered at clients for a given transcoder placement option:* Our objective is to develop solutions to find the optimal rates delivered at clients for any given transcoder placement option, ST, AT, or SNT.

   (a) **Optimization-based approach:** We formulate the problem as an optimization function, subject to the following three sets of constraints:

      i. **Rate constraints** which bound the lower and upper limits of the encoded rate,

      ii. **Transcoder constraints** that enforce the property of transcoding, viz., the encoded rates can only be non-increasing, and

      iii. **Delay tolerance constraints** which enforce the client specified delay tolerance,

      to find the maximum rate at which the stream should be played out for each client.

   (b) **Algorithm based on properties of the network:** Given the exponential search space, as we experimentally show, the optimization based approach can incur computation overheads which make it impractical. We show that it is possible to develop a two-pass algorithm *find_opt_rates_I* with O(DC) complexity where D is the number of levels in the multicast tree and C is the number of clients. This algorithm finds the optimal delivered rates at clients, for any transcoder placement option, ST, AT, or SNT. We prove the optimality of the algorithm *find_opt_rates_I*.

2. *Determining optimal number of transcoders and their placement (for transcoder placement option AT)*: When all relay nodes are transcoder capable, *find_opt_rates_I* determines the stream rates flowing through each link in the network and the optimal rates delivered at the clients. We call these the *best delivered rates*, when there is no constraint on the number of transcoders. Even though all relay nodes have transcoders, all of them are not enabled to serve the clients with the best delivered rates. Our objective is to find the

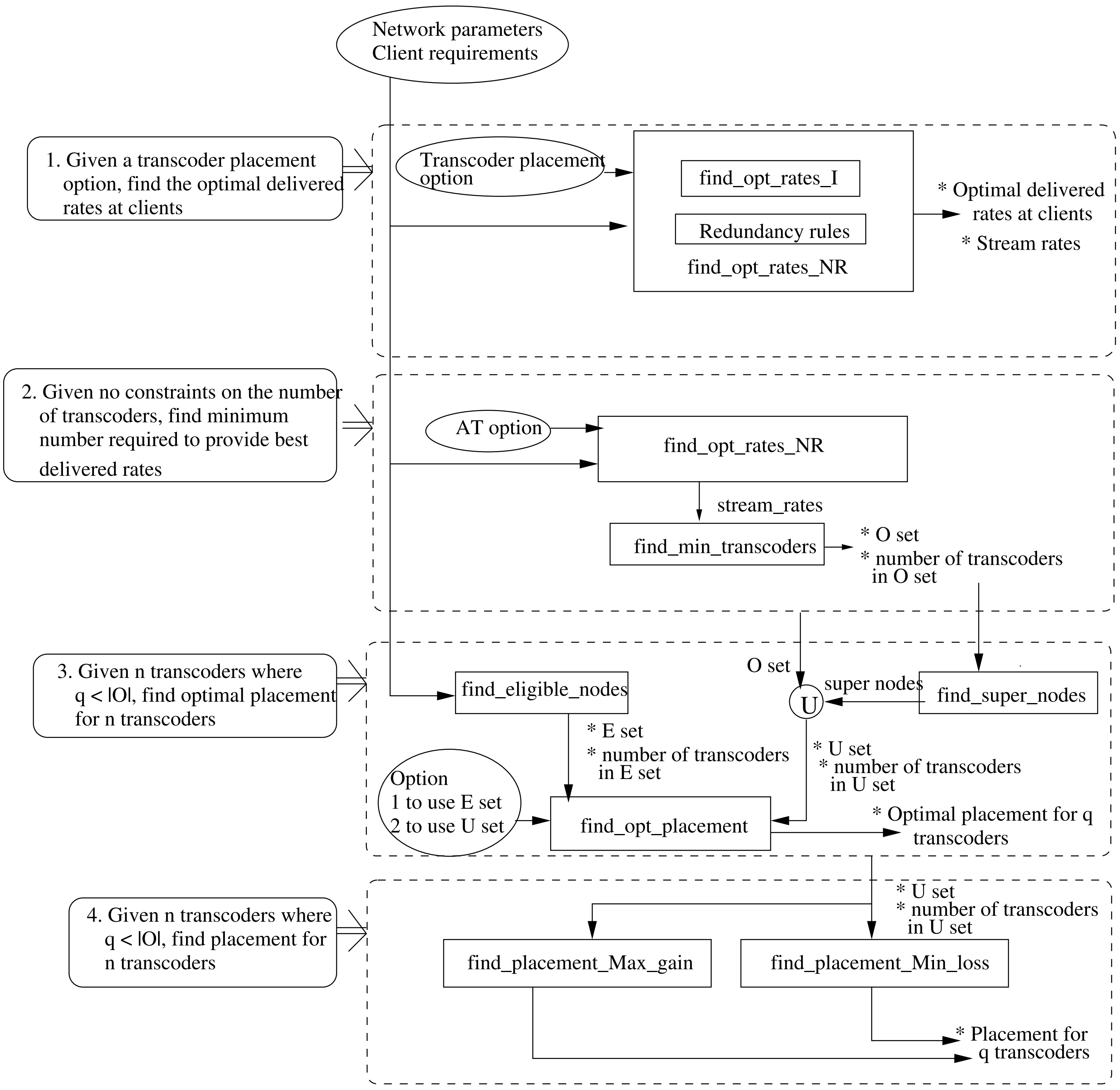

Network parameters
Client requirements
1. Given a transcoder placement option, find the optimal delivered rates at clients
Transcoder placement option
find_opt_rates_I
Redundancy rules
find_opt_rates_NR
* Optimal delivered rates at clients
* Stream rates
2. Given no constraints on the number of transcoders, find minimum number required to provide best delivered rates
AT option
find_opt_rates_NR
stream_rates
find_min_transcoders
* O set
* number of transcoders in O set
3. Given n transcoders where q < |O|, find optimal placement for n transcoders
find_eligible_nodes
O set
super nodes
U
find_super_nodes
* E set
* number of transcoders in E set
* U set
* number of transcoders in U set
Option
1 to use E set
2 to use U set
find_opt_placement
* Optimal placement for q transcoders
4. Given n transcoders where q < |O|, find placement for n transcoders
* U set
* number of transcoders in U set
find_placement_Max_gain
find_placement_Min_loss
* Placement for q transcoders

Figure 3.4: Overview of solution approach: Scheduled streaming, Static link bandwidths

optimal transcoder placement option required to serve the clients with the best delivered rates.

(a) Using *find_opt_rates_I* and network properties related to transcoder placement that detect redundant transcoders, we develop an algorithm *find_min_transcoders* that determines the optimal transcoder placement option required to deliver best delivered rates to clients. The optimal set of transcoders is referred to as the *optimal set*, denoted by $\mathcal{O}$.

(b) We prove that $\mathcal{O}$ is the optimal set of transcoders required to provide best delivered rates at all clients.

3. *Determining optimal placement for a given number of transcoders:* In practical scenarios, resources available are limited. Given a number of transcoders $q < |\mathcal{O}|$, we know that the delivered rates at some clients would be less than their best delivered rates. Our objective is to place the transcoders such that the decrease in delivered rates across clients is minimal; i.e., given a limited number of transcoders, we find the placement option that delivers the best possible rates to clients.

   (a) **Optimization-based approach:** We formulate the problem as an optimization function subject to the rate constraints and delay tolerance constraints as explained in 1(a). Instead of the transcoder constraints, we specify **number of transcoder constraint** to limit the number of transcoders deployable to the specified value q.

   (b) **Algorithm based on properties of the network:** We also develop an algorithm that considers all combinations of all the relay nodes in the network for placing a given number of transcoders. Suppose there are N nodes in the network. The total number of eligible nodes E that have to be considered for transcoder placement is given by the expression: $E = N-(C+1)$, where C is the number of clients. Since we always have a transcoder at $\mathcal{S}$, we do not include it in the set of eligible nodes. Let $|\mathcal{O}|$ be the optimal number of transcoders required for best delivered rates at the clients as determined by algorithm *find_min_transcoders* discussed in the previous problem. Given q transcoders, where $q < |\mathcal{O}|$, our objective is to seek a placement which delivers the best possible rates to clients. Number of possible unique placements is given by:

$$\begin{pmatrix} E \\ q \end{pmatrix} = \frac{E!}{q!(E-q)!}$$

We show that the number of combinations to be considered for determining the optimal placement for q transcoders can be reduced to nodes in $\mathcal{O}$ under certain conditions. We also determine when nodes outside of $\mathcal{O}$, which are termed *super nodes*, must be considered for finding optimal placement for q transcoders. Algorithm *find_super_nodes* determines the additional nodes $R_1, R_2, \ldots, R_k$ to be considered. We show that optimal placement for q transcoders can be determined by considering all combinations of q nodes from the set,
$\mathcal{U} = \mathcal{O} \cup \{R_1, R_2, \ldots\ R_k\}$, termed the *useful set*.

(c) **Greedy algorithms:** Given the exponential complexity of the optimal algorithm since the number of nodes in the network can be large, we develop two greedy alternatives: a *max-gain* algorithm and a *min-loss* algorithm and evaluate these algorithms.

3.2.2 Problem 1B: Scheduled streaming, varying link bandwidths

When link bandwidths vary over the session duration, we assume a bandwidth prediction module that provides an *advance estimate* of the available link bandwidths over prediction intervals spanning the session duration. Suppose t_0 is the scheduled start of the streaming session; client requirements are given. The advance estimate provides predicted bandwidths over prediction intervals spanning (t_0 + max(δ_i)) where δ_is are the delay tolerance values of the clients. We summarize the contributions made by this thesis for this case in Figure 3.5.

- *Determining optimal rates delivered at clients when the link bandwidths are varying*:

  1. **Optimization-based approach:** We formulate the problem as an optimization function, subject to the following three sets of constraints:

     (a) **Rate constraints** which bound the lower and upper limits of the encoded rate,

     (b) **Layer constraints** that enforce the property of layering, viz., the stream rates flowing through two consecutive links must be such that the stream rate through the first link is greater than or equal to the rate flowing through the next one;

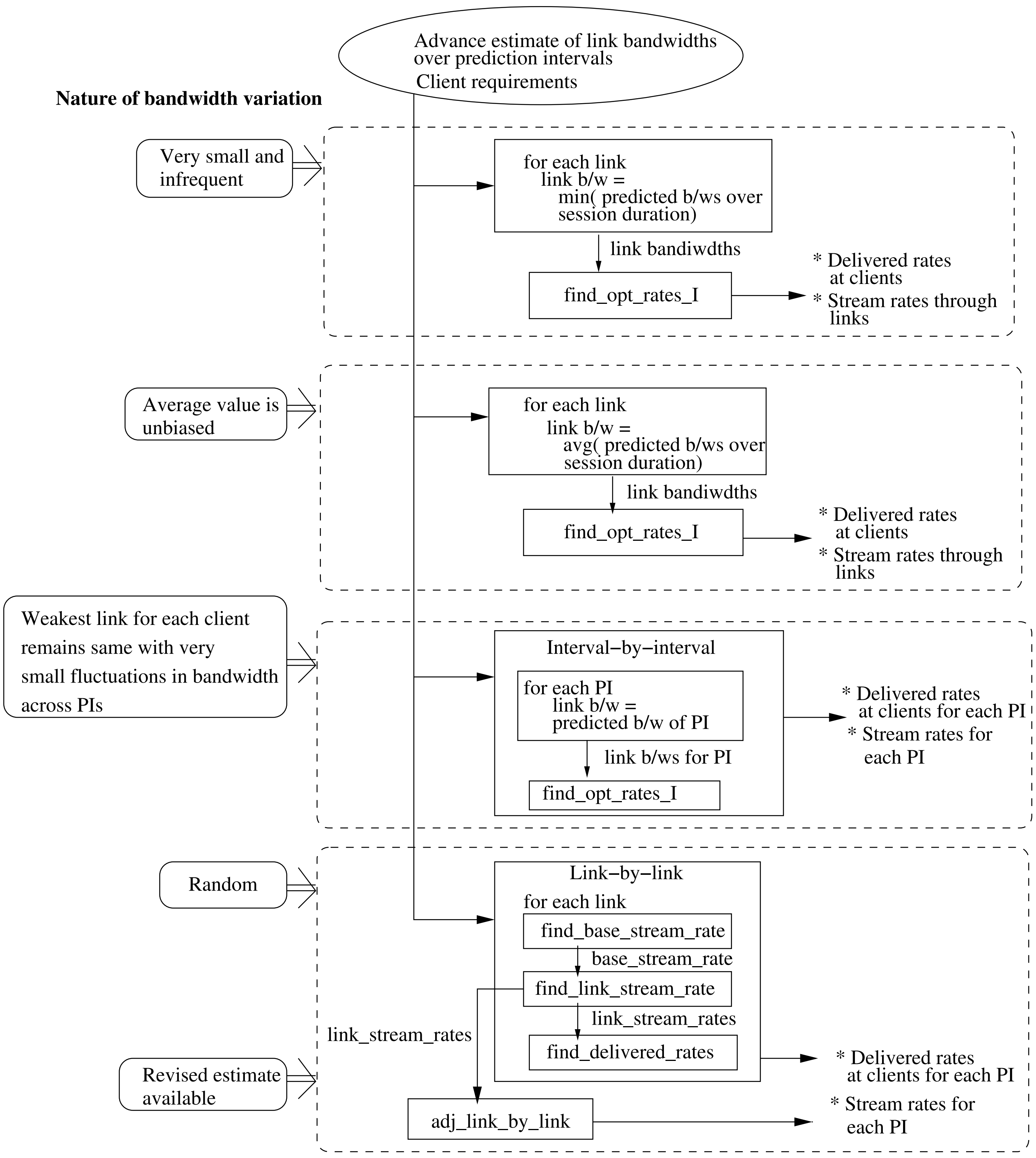

Advance estimate of link bandwidths over prediction intervals
Client requirements
Nature of bandwidth variation
Very small and infrequent
for each link link b/w = min(predicted b/ws over session duration)
link bandiwdths
find_opt_rates_I
* Delivered rates at clients
* Stream rates through links
Average value is unbiased
for each link link b/w = avg(predicted b/ws over session duration)
link bandiwdths
find_opt_rates_I
* Delivered rates at clients
* Stream rates through links
Weakest link for each client remains same with very small fluctuations in bandwidth across PIs
Interval−by−interval
for each PI link b/w = predicted b/w of PI
link b/ws for PI
find_opt_rates_I
* Delivered rates at clients for each PI
* Stream rates for each PI
Random
Link−by−link
for each link
find_base_stream_rate
base_stream_rate
find_link_stream_rate
link_stream_rates
find_delivered_rates
link_stream_rates
Revised estimate available
adj_link_by_link
* Delivered rates at clients for each PI
* Stream rates for each PI

Figure 3.5: Overview of solution approach: Scheduled streaming, Varying link bandwidths

note that these constraints are the same as the transcoder constraints as given in 1(a), and

(c) **Delay tolerance constraints** which enforce the client specified delay tolerance,

to find the maximum rate at which the stream should be played out for each client.

2. Given the exponential complexity of the optimization approach, we explore different solutions based on the nature of bandwidth variation. We start with simple ones that convert the problem to the static bandwidth case such as (i) using the minimum value of bandwidth predicted over the session duration for each link (ii) using the average bandwidth over the session duration for each link, and (iii) treating each prediction interval as an instance of static bandwidth case, termed *interval-by-interval algorithm*. For each solution we propose, we outline the usefulness of that solution and its limitations.

   Our objective is to maximize the use of available bandwidth to provide clients with the best possible loss free delivered rates. To this end, we develop a *link-by-link algorithm* that considers each link to find the maximum rate supported by the link without loss. Considering the links in each client's path, the algorithm determines the delivered rates at the clients.

   We show that the link-by-link algorithm with linear complexity performs close to the optimal algorithm in finding the maximum loss-free delivered rates at the clients.

3. *Adjusting stream rates if revised bandwidth estimates are available*: We show how to use the *revised estimate* of link bandwidths, if available at the beginning of every prediction interval, to adjust the delivered rates at the clients; note that the delivered rates are calculated by the link-by-link algorithm using the *advance estimate* across all prediction intervals. We show that this algorithm is effective in exploiting the available bandwidth on the links over each prediction interval compared to just using the advance estimate.

3.2.3 Problem 2A: On-demand streaming, static link bandwidths

In the previous two problems, we considered scheduled streaming, where streaming of a given content starts at time t_0 for all clients. Now we consider on-demand streaming where client requests are handled as and when they arrive. Unlike the scheduled streaming case, transmission

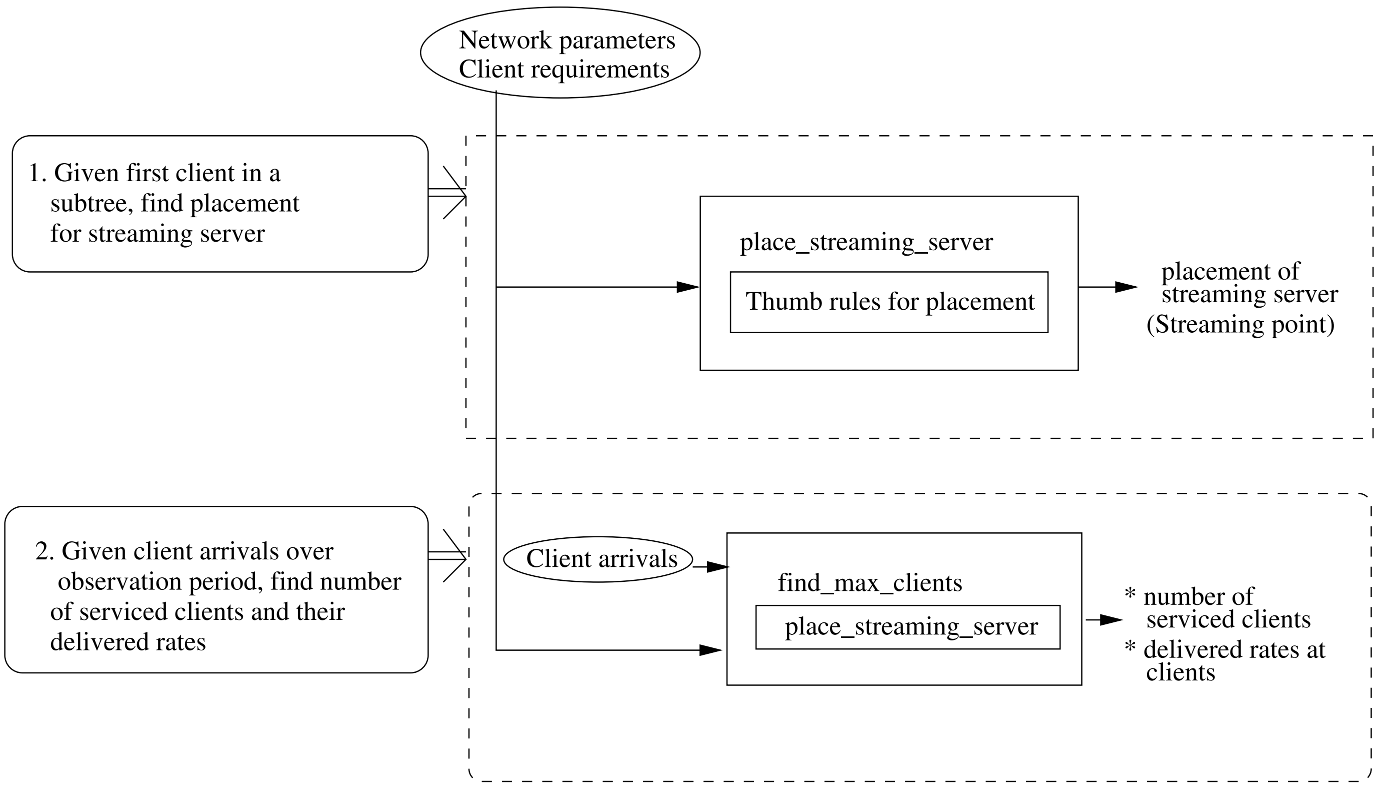

Network parameters
Client requirements
1. Given first client in a subtree, find placement for streaming server
place_streaming_server
Thumb rules for placement
placement of streaming server (Streaming point)
2. Given client arrivals over observation period, find number of serviced clients and their delivered rates
Client arrivals
find_max_clients
place_streaming_server
* number of serviced clients
* delivered rates at clients

Figure 3.6: Overview of solution approach: On-demand streaming, Static link bandwidths

from $\mathcal{S}$ may not start immediately, if links are busy serving requests from other clients. Thus, a client C_i connects to the source $\mathcal{S}$ at time t_i and requests for contents with base encoding rate Γ and playout duration $\mathcal{T}$; C_i specifies the time it is willing to wait for the playout to start, its delay tolerance δ_i. $\mathcal{S}$ admits C_i and services its request with an appropriate rate r_i such that playout at C_i starts at (t_i+δ_i) and ends at $((t_i + \delta_i) + \mathcal{T})$, if $r_i >=$ the client's minimum required rate γ_i^{min}. We summarize the contributions made by this thesis for this case in Figure 3.6.

- *Determining optimal rates delivered at clients when the clients request for contents over a period of time:*
  - We assume an observation period over which client request arrivals are monitored. Network characteristics and client requirements are given.
  - Given highly provisioned links in the distribution network, we propose an algorithm that combines data transfer and streaming mechanisms to efficiently handle requests for the same contents over a period of time.

Determining appropriate placement for streaming server: Given the arrival pattern of client requests over the observation period, streaming servers have to be placed at appro-

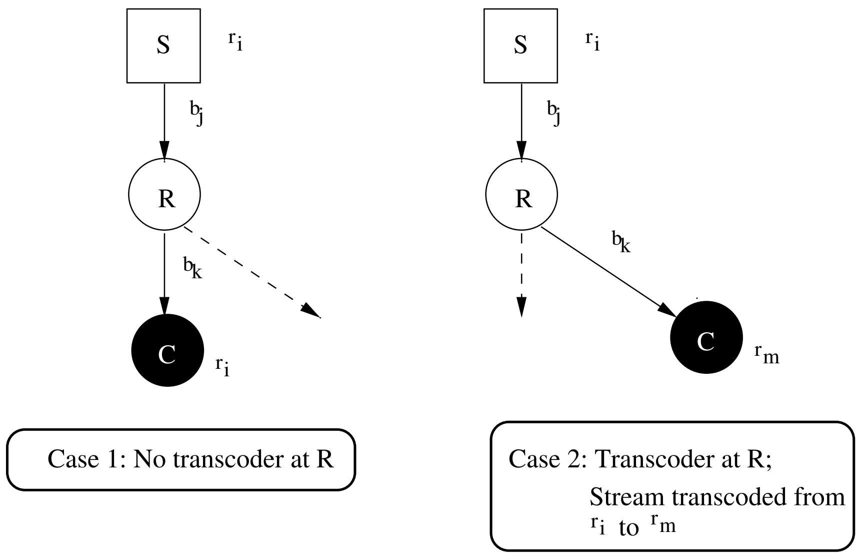

S
r_i
b_j
R
b_k
C
r_i
Case 1: No transcoder at R
S
r_i
b_j
R
b_k
C
r_m
Case 2: Transcoder at R; Stream transcoded from r_i to r_m

Figure 3.7: Example to illustrate data-flow: Scheduled streaming

priate network nodes in order to maximize number of clients serviced. Using observations of the network characteristics related to data transfer and streaming, we develop rules for placement of the streaming servers. These rules are applied in the algorithm for finding the number of serviced clients and the rates delivered at the clients.

3.2.4 Problem 2B: On-demand streaming, varying link bandwidths

Determining optimal rates delivered at clients over a observation period, given the network characteristics and an arrival pattern, when the distribution network links are highly provisioned and access network links have varying bandwidths: Using the insights gained from the analysis for solving Case 1B and Case 2A, algorithms can be designed for this case. Detailed analysis is not in the scope of this thesis.

3.3 Flow of data through the nodes

In this section, we consider the flow of data for the two service types: scheduled streaming and on-demand streaming. Based on this discussion, we propose the node architectures to support these two service types in Section 3.4.

3.3.1 Scheduled streaming

Consider the nodes in the network as depicted in Figure 3.7. A stream encoded at r_i originates from the source node $\mathcal{S}$. b_j and b_k are bandwidths on the links connecting $\mathcal{S}$ to the relay node R and R to client C respectively, as indicated in the figure. We have three types of nodes in the network: (i) Source node, (ii) Relay node, and (iii) Client node. In the scheduled streaming case, a relay node can be of two types: (i) capable of transcoding/layering and (ii) not capable of transcoding/layering.
With reference to the Figure 3.7, we trace the flow of data through the nodes below:

- Let r_i be the maximum encoded rate of the stream required by clients in the multicast tree, as computed by *find_opt_rates_I*. Note that r_i is at best equal to Γ. When $r_i < \Gamma$, the source $\mathcal{S}$ transcodes the stream to r_i. Thus, data rate of the stream is r_i for $\mathcal{S}$ and the outgoing data rate depends on the bandwidth of the outgoing link from $\mathcal{S}$. In our example, b_j is the bandwidth of the outgoing link from $\mathcal{S}$.

- Consider the relay node R. Data flows into R at b_j kbps and flows out of R at b_k kbps. When R has no transcoder, the stream rate r_i remains unchanged, as shown in Case 1 in Figure 3.7. Based on the bandwidths on the incoming and outgoing links, b_k kbps is sent out of the buffer at R and the remaining data is maintained in the buffer.

  Suppose the transcoder at R transcodes the stream from r_i to r_m, where r_m is the delivered rate at C, as shown in Case 2 in Figure 3.7.

- Data flows at a rate b_k kbps into the client C. The data is played out at the encoded rate r_i starting at $(t_0 + \delta)$, where t_0 is the start time of the transmission and δ is the delay tolerance of C. Hence the data is buffered at C till the start of playout. The stream is played out using a media player application at the client.

Note that every node requires buffering capability. The source and some relay nodes have transcoding/layering capability. The client needs capability to re-assemble the stream to play it out at the encoded rate. We look at each of these capabilities and discuss the components that provide the capability to the node.

Buffer: Storing capability

The buffer is a simple First in First out (FIFO) mechanism which is available on every node in the network. All the data flowing into the node passes through the buffer. The amount of data supported by the outgoing link is continuously sent out and the remaining data stays in the buffer. When bandwidths are varying, the encoding rate of the stream varies over the prediction intervals. Encoding rate of the stream over a given prediction interval may be less than, equal to, or more than the bandwidth available on the outgoing link. Irrespective of these variations, the buffer follows the first in first out order to send data at a maximum rate sustainable, given the bandwidth available on the outgoing link.

Transcoder/Layer encoder: Transrating capability

As defined in Chapter 1, *transrating* refers to decreasing the encoding rate of a stream without changing its encoding format. The maximum amount of data that can flow into a node depends on the bandwidth of its incoming link and maximum data that can flow out of a node depends on the bandwidth of its outgoing link. However, a stream can be encoded at a rate higher than the bandwidths of a node's incoming and outgoing links. When a node serves multiple clients with different delivered rates, the node can reduce the stream rate appropriately to serve the clients with their delivered rates. When n_i is transcoder capable, we assume a real-time transcoder that appropriately reduces the stream rate before sending it through the outgoing link. When n_i is layer encoding capable, we assume a layering mechanism that appropriately drops layers to ensure loss free transmission through the outgoing link.

Suppose n_i serves multiple clients C_1 and C_2. Let r_i be the delivered rate at C_1 and r_j be the delivered rate at C_2, where $r_i > r_j$. When node n_i has a transcoder or layer encoder functionality built in, C_1 is served with a stream encoded at r_i and the rate of the stream is reduced from r_i to r_j to serve C_2.

Media player: Streaming capability

Let node n_i be a client; suppose data is flowing into n_i at a rate b_k. Let r_i be the delivered rate at n_i. Data flowing into n_i is buffered till the start time of playout given by: $(t_0 + \delta)$, where t_0 is the start time of the transmission and δ is the delay tolerance of the client. Data is stored in the buffer in the order of arrival as discussed in the previous section. Starting at $(t_0 + \delta)$ the stream

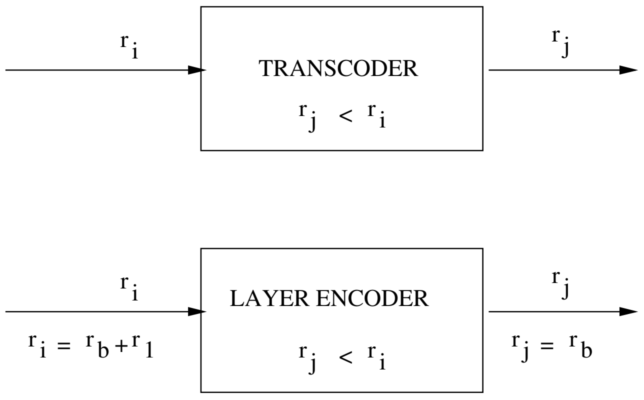


Figure 3.8: Resource used

is played out encoded at rate r_i and transmission ends at $(t_0 + \delta + \mathcal{T})$. The media player fetches data from the buffer and streams the data at the appropriate encoded rate.

Figure 3.8 depicts the architecture of the transcoder/layer encoder component; buffer and media player are presented as part of the node architectures.

3.3.2 On-demand streaming

Consider the nodes in the network as depicted in Figure 3.9. A stream encoded at r_i originates from the source node $\mathcal{S}$. b_j and b_k are bandwidths on the links connecting $\mathcal{S}$ to the relay node R and R to the streaming relay node R^{st} respectively, as indicated in the figure. We have four types of nodes in the network: (i) Source node, (ii) Relay node, (iii) Relay node with streaming server, and (iv) Client node. With reference to Figure 3.9, we trace the flow of data through the nodes below:

- Let r_i be the maximum encoded rate of the stream, as computed by *find_opt_rates_I*. Note that r_i is at best equal to Γ. When $r_i < \Gamma$, the source $\mathcal{S}$ transcodes the stream to r_i. Thus, data rate of the stream is r_i for $\mathcal{S}$ and the outgoing data rate depends on the bandwidth of the outgoing link from $\mathcal{S}$. In our example b_j is the bandwidth of outgoing link. Suppose b_j is much greater than Γ. $\mathcal{S}$ does the following:
  - invokes algorithm *place_streaming_server* and identifies the relay node where streaming server is placed. With reference to Figure 3.9, let R^{st} be the relay node chosen for streaming server placement.
  - invokes the data transfer mechanism and sends the data at the best possible loss-free

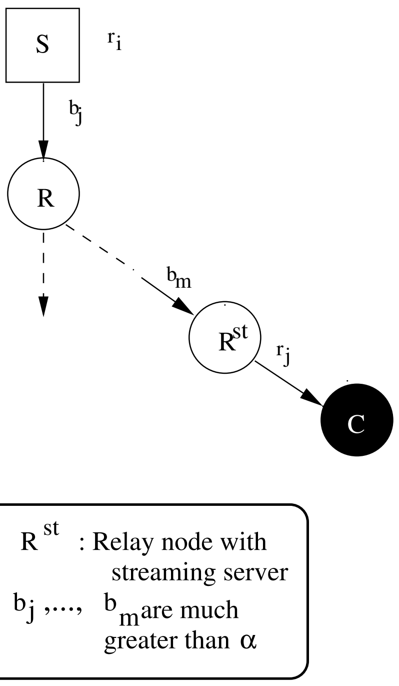

S
r_i
b_j
R
b_m
R^{st}
r_j
C
R^{st} : Relay node with streaming server
$b_j, \ldots, b_m$ are much greater than α

Figure 3.9: Example to illustrate data-flow: On-demand streaming

rate based on the bandwidths of links from $\mathcal{S}$ to R^{st}. In our example data transfer happens at a rate $= min(b_j, \ldots, b_m)$. Let this rate be b_k.

- Consider the relay node R^{st}. Data flows into R^{st} at b_k kbps. R^{st} streams the data at r_i to C_i. In addition, R^{st} maintains the contents in a cache to serve future requests for the same contents when the links from $\mathcal{S}$ to R^{st} are busy. Note that R^{st} requires transrating capability to serve requests for the same content from other clients.

Based on the above discussion, we outline the additional capabilities required at the nodes for on-demand streaming in the next sections.

Data transfer mechanism: Packetizing capability

Since $\mathcal{S}$ has the information on the placement of streaming server and the available bandwidths on the links from itself to the node with streaming server, it invokes a data transfer mechanism to send the data at an appropriate loss-free rate. The data transfer mechanism packetizes the data such that the highly provisioned links from $\mathcal{S}$ to R^{st} are utilized maximally.

Storage for future use: Caching capability

Each relay node chosen for streaming server placement termed a *streaming point*, has caching capability. The content stream is cached at this node while being streamed to the requesting client. A simple Time To Live of Content (TTL) mechanism can be used to evict the contents from the cache. Details are provided in Chapter 7.

3.4 Node architectures

In this section, we present the architectures of the network nodes based on our discussion of dataflow through these nodes. We present the node architectures for the scheduled streaming case and then provide the architecture for the nodes for on-demand streaming.

3.4.1 Node architectures for scheduled streaming

Considering the dataflow for the scheduled streaming as discussed in Section 3.3.1, architecture of the nodes to support the dataflow are presented in this section.

Source node architecture for scheduled streaming

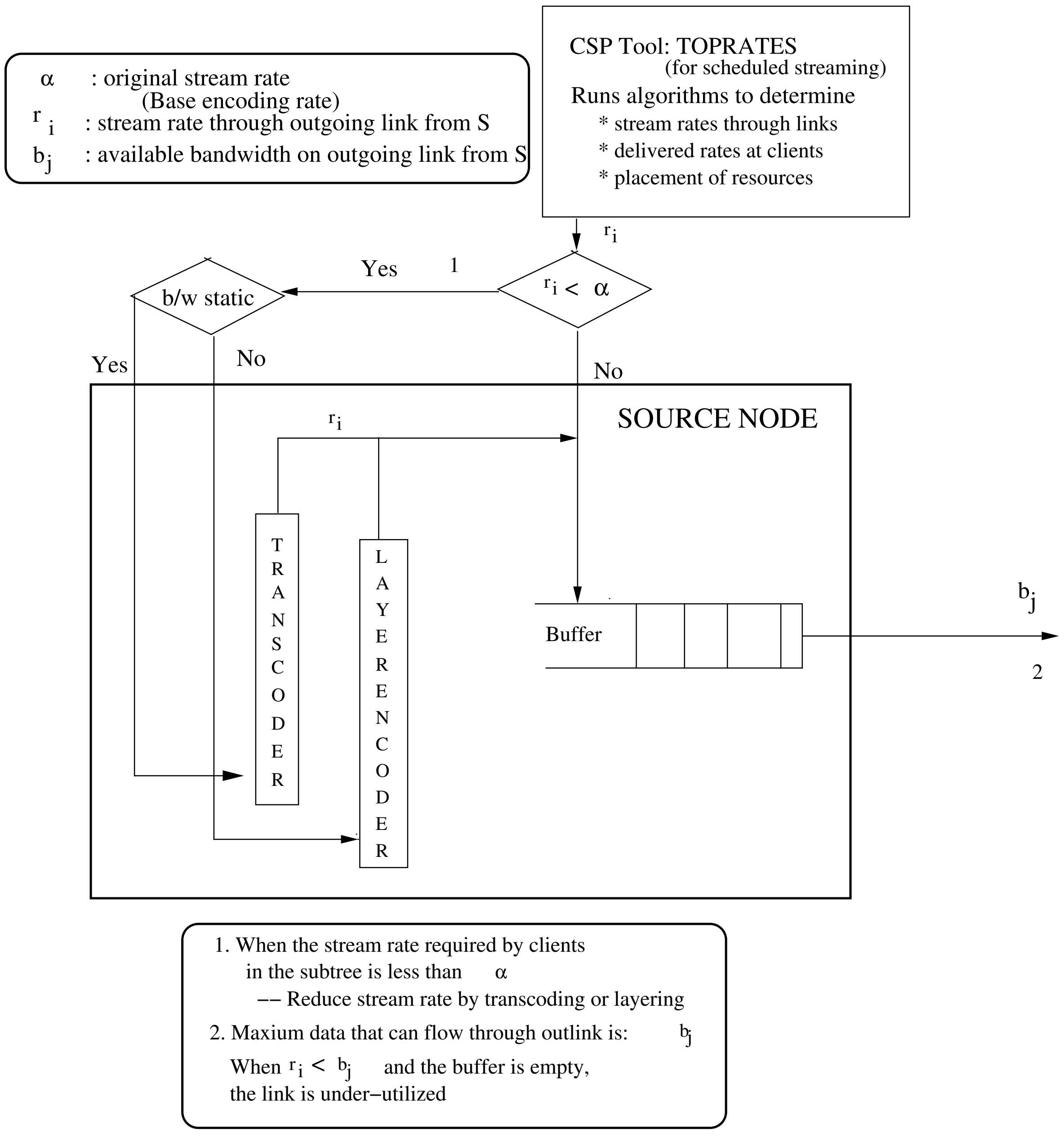
A source node uses the following basic components: (i) Transcoder/Layer encoder and (ii) Buffer. Architecture of source node is presented in Figure 3.10.

Relay node architecture

A relay node that has no transcoder/layer encoder consists of the buffer to transmit the data without loss. When a relay node has the additional resource of a transcoder or a layer encoder, it can reduce the encoded rate of the stream before sending the data through its outgoing link. Relay node architectures with and without the transcoder/layer encoder functionality are presented in Figures 3.11 and 3.12 respectively.

Client node architecture

Playout at a client starts only after δ time units, the delay tolerance specified by the client from the start of transmission. Thus, even though data starts flowing into the client node at time t_0,



α : original stream rate (Base encoding rate)
r i : stream rate through outgoing link from S
bj : available bandwidth on outgoing link from S
CSP Tool: TOPRATES (for scheduled streaming)
Runs algorithms to determine
* stream rates through links
* delivered rates at clients
* placement of resources
ri
ri < α
Yes
1
b/w static
Yes
No
No
SOURCE NODE
ri
TRANSCODER
LAYERENCODER
Buffer
bj
2
1. When the stream rate required by clients in the subtree is less than α
-- Reduce stream rate by transcoding or layering
2. Maxium data that can flow through outlink is: bj
When ri < bj and the buffer is empty, the link is under-utilized

Figure 3.10: Source node architecture for scheduled streaming

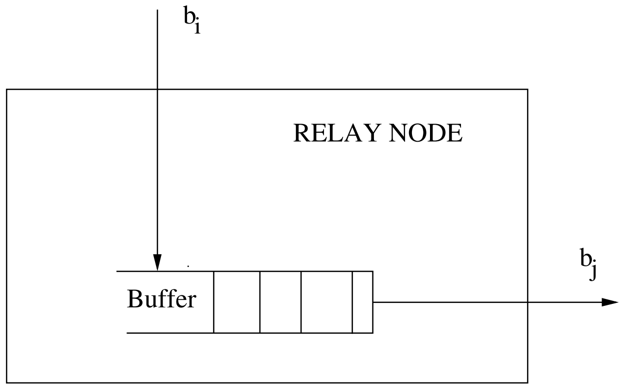
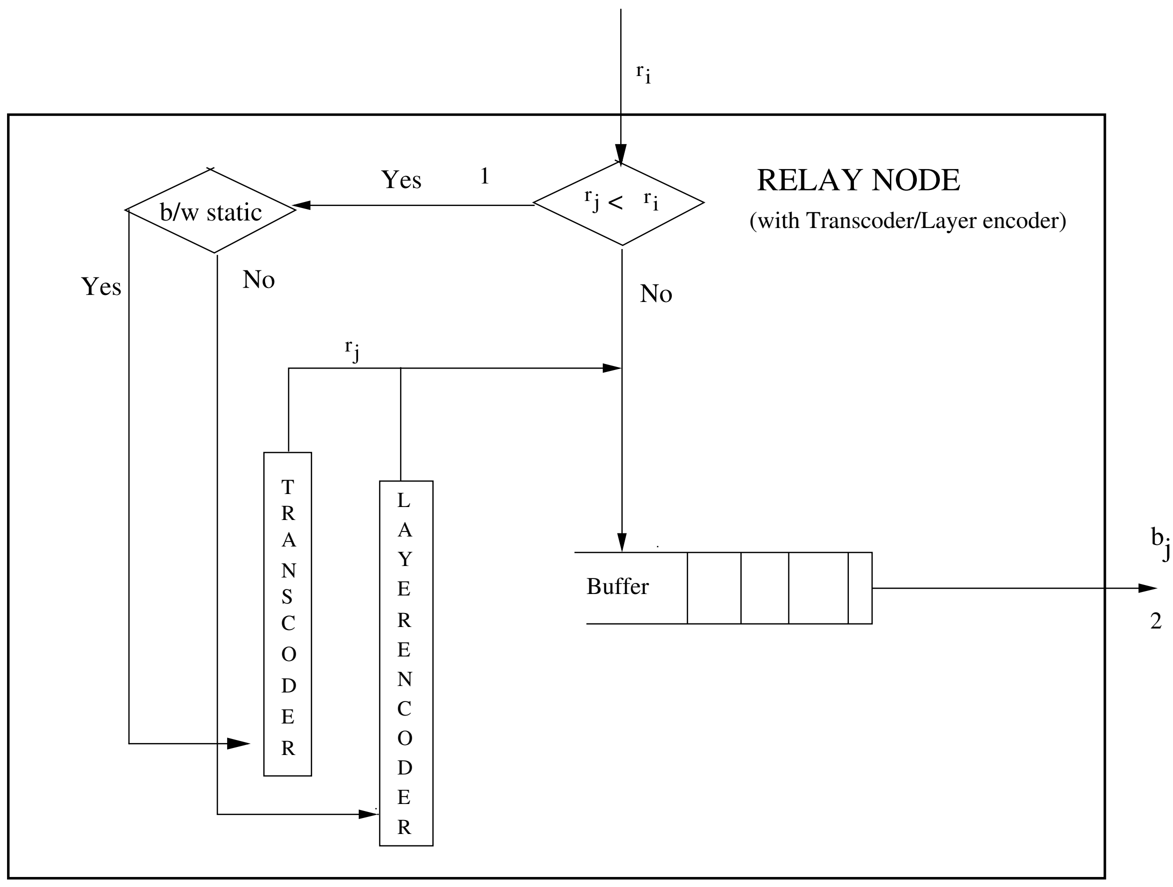


Figure 3.11: Relay node architecture



1. When the stream rate required by clients in the subtree r_j , is < r_i

   -- Reduce the stream rate
2. Maxium data that can flow through outlink is: b_j

   when $r_j < b_j$ and the buffer is empty, the link is under_utilized

Figure 3.12: Relay node with transcoder/layer encoder

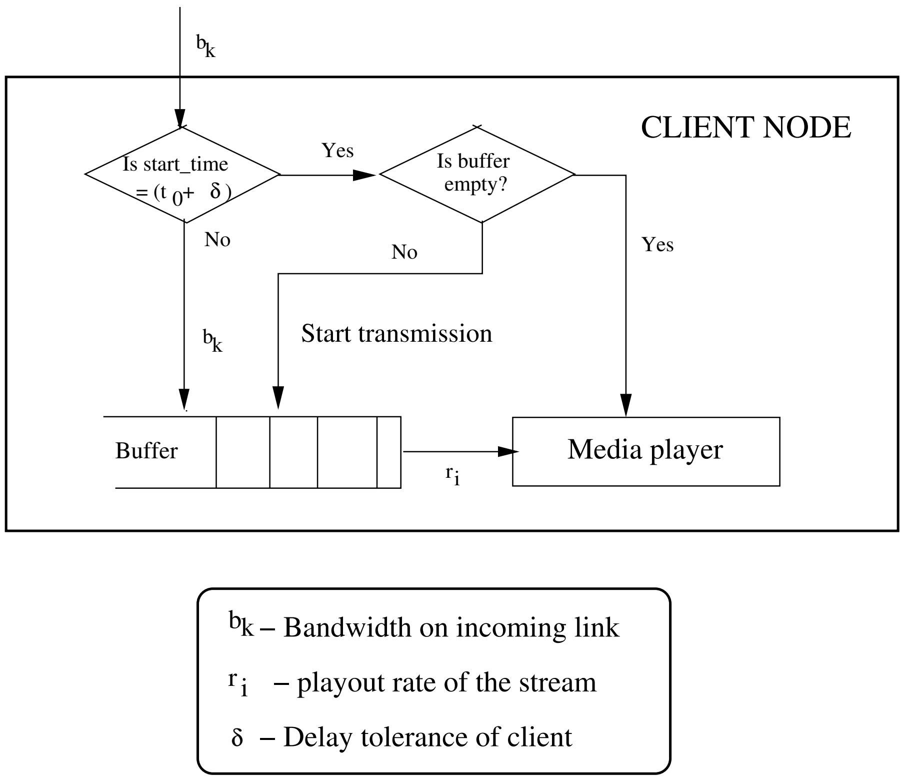

b_k
CLIENT NODE
Is start_time = (t_0+ δ)
Yes
Is buffer empty?
No
No
Yes
b_k
Start transmission
Buffer
r_i
Media player
b_k – Bandwidth on incoming link
r_i – playout rate of the stream
δ – Delay tolerance of client

Figure 3.13: Client node architecture

data is buffered till (t_0+δ). Playout at the client node starts at (t_0+δ) using the player application, which streams the data at the appropriate encoded rate.

Figure 3.13 depicts the architecture of a client node.

3.4.2 Node architectures for on-demand streaming

As discussed in Section 2.2.2, for on-demand streaming, additional data transfer functionality is required at the source node. Note that relay nodes have architectures as presented in Figure 3.11. However, there are special relay nodes where streaming servers are placed. Each of these nodes has caching capability. We present the architecture for the source node and relay nodes with streaming server in this section. The client node architecture is similar to the scheduled streaming case as presented in Figure 3.13.

Source node architecture for on-demand streaming

In on-demand streaming, the source node has an additional function to packetize the data appropriately to maximally utilize the bandwidths on the links from S to the streaming point without loss. Source node architecture required to support on-demand streaming is given in Figure 3.14.

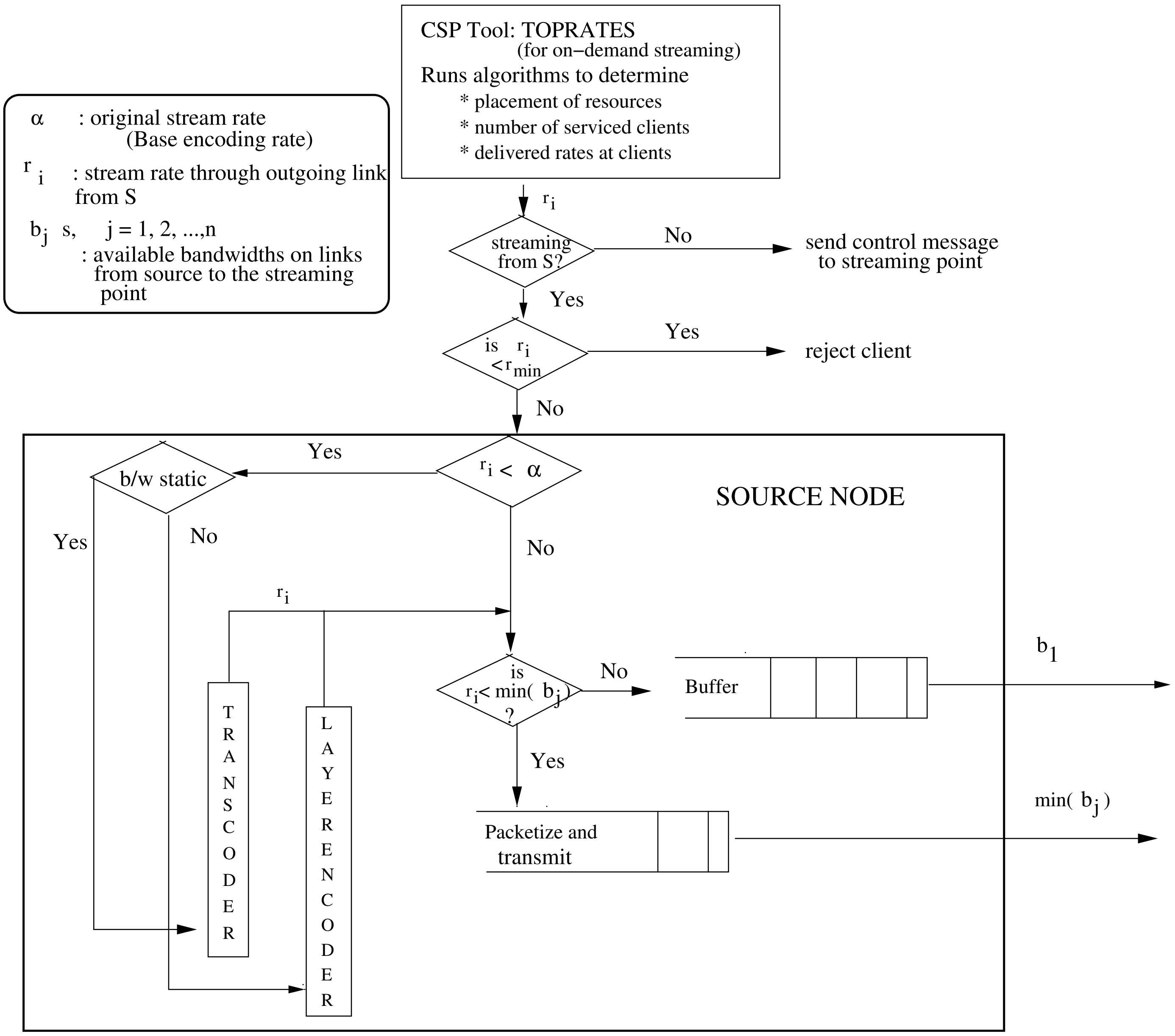

CSP Tool: TOPRATES
(for on−demand streaming)
Runs algorithms to determine
* placement of resources
* number of serviced clients
* delivered rates at clients
α : original stream rate (Base encoding rate)
r i : stream rate through outgoing link from S
bj s, j = 1, 2, ...,n : available bandwidths on links from source to the streaming point
ri
streaming from S?
No
send control message to streaming point
Yes
is ri < rmin
Yes
reject client
No
SOURCE NODE
ri < α
Yes
b/w static
Yes
No
No
ri
TRANSCODER
LAYERENCODER
is ri< min(bj) ?
No
Buffer
b1
Yes
Packetize and transmit
min(bj)

Figure 3.14: Source node architecture for on-demand streaming

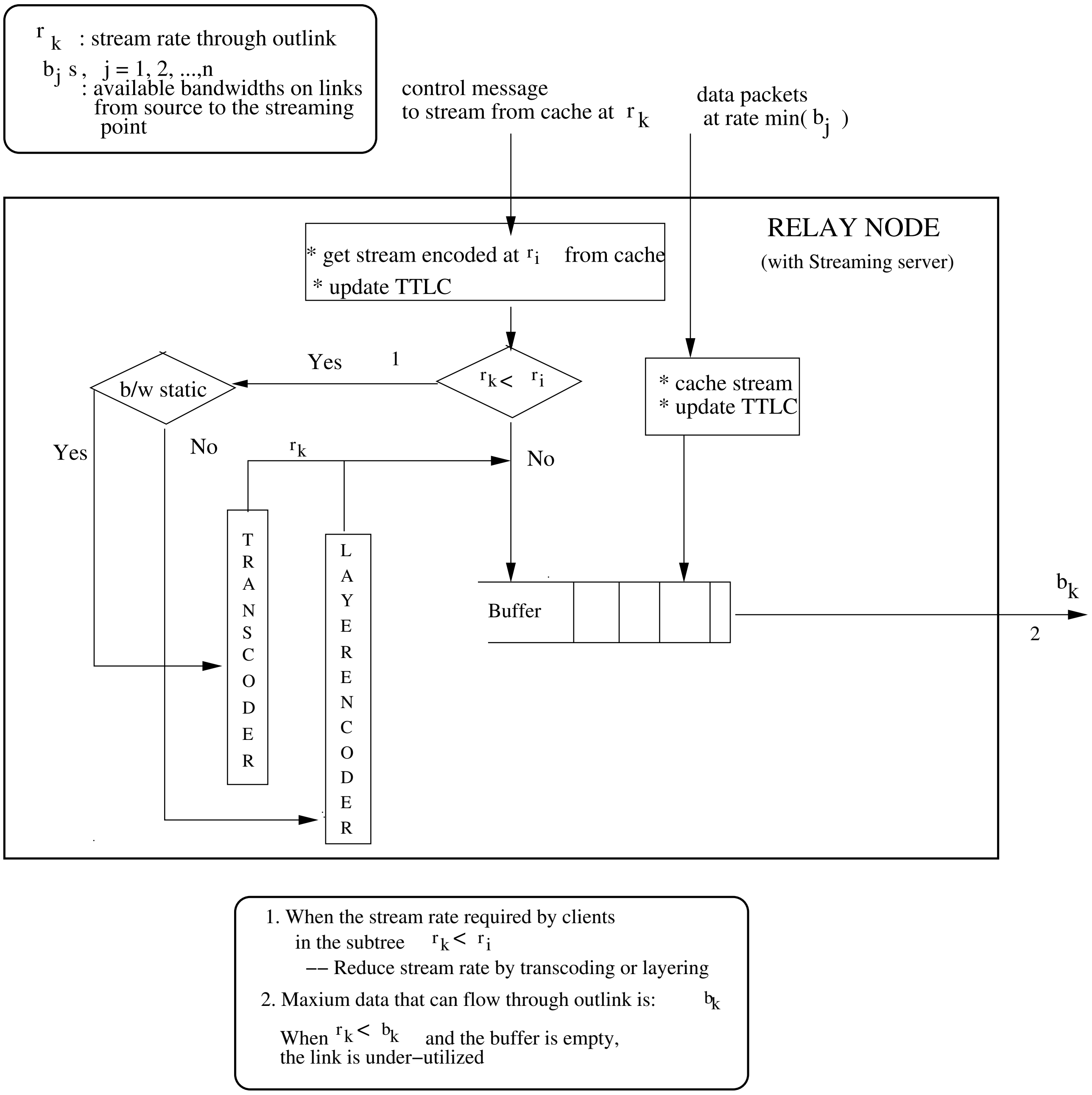

r_k : stream rate through outlink
b_j s, j = 1, 2, ...,n : available bandwidths on links from source to the streaming point
control message to stream from cache at r_k
data packets at rate min(b_j)
RELAY NODE
(with Streaming server)
* get stream encoded at r_i from cache
* update TTLC
Yes
1
$r_k < r_i$
b/w static
* cache stream
* update TTLC
Yes
No
r_k
No
TRANSCODER
LAYERENCODER
Buffer
b_k
2
1. When the stream rate required by clients in the subtree $r_k < r_i$
-- Reduce stream rate by transcoding or layering
2. Maxium data that can flow through outlink is: b_k
When $r_k < b_k$ and the buffer is empty, the link is under-utilized

Figure 3.15: Relay node with streaming server

Architecture of relay node with streaming server

Relay node with streaming server requires the following additional functionality: (i) stream the packetized data flowing in, at appropriate rate through the outlink, while caching it (ii) when a control message is received, stream the data at appropriate rate from the cache. Figure 3.15 depicts the architecture of a relay node with a streaming server.

3.5 Summary

In this chapter, we provided the definitions of the terms used throughout this thesis and listed the key assumptions made. We provided detailed definition of the problem and presented an overview of the data flow for the two service types we consider: Scheduled streaming and On-demand streaming. We discussed the architecture of the nodes required to support the two service types. This chapter provides the system model in a nutshell before we proceed to discuss each sub-problem in detail in the next sections.

Chapter 4

Determining optimal delivered rates at clients

(Scheduled streaming, Static link bandwidths)

4.1 Introduction

For a given topology and client requirements, optimal rates delivered at clients depend on the (i) number of transcoders and (ii) placement of these transcoders. As discussed in Section 1.3.1, there are three transcoder placement options, viz. Source only Transcoding (ST), Anywhere Transcoding(AT), and Selected Node Transcoding (SNT) [17]. These refer to the combination of nodes in the network which can have transcoders. Note that we can have the following cases:

1. Unlimited number of transcoders, all nodes available for placement – This case is equivalent to AT. Note that when all network nodes are capable of transcoding, the delivered rates at the clients are the *best delivered rates* for the given network topology and link bandwidths. Our objective is to find these best delivered rates at clients and the minimal placement of transcoders to deliver the best delivered rates.

2. Unlimited number of transcoders, Limited nodes available for placement – This case arises when the CSP does not have access to some of the relay nodes in the network or when some relay nodes cannot be equipped with transcoders. Our objective is to find the optimal delivered rates at clients for a specific choice of relay nodes having transcoders. This case is an instance of SNT.

3. Limited number of transcoders, all nodes available for placement – In this option, for a

given number of transcoders, we can decide on their placement anywhere among the relay nodes. Our objective is to find an optimal placement that delivers the optimal rates. This case is also an instance of SNT.

4. Limited number of transcoders, Limited nodes available for placement – In this case the CSP has access only to some relay nodes to place a specific number of transcoders. Our objective is to find an optimal placement among the available nodes for placement that maximizes the delivered rates across clients. This case is handled as a special case of (3).

We convert the objectives defined for these cases into the following questions that define the scope of Problem 1A:

1. Given a multicast tree and client requirements, what are the optimal delivered rates at which the clients can be serviced for a given transcoder placement option?
2. In order to provide the clients with the best delivered rates
    (a) How many transcoders are needed?
    (b) Where should the transcoders be placed?
    (c) What rate conversions should these transcoders perform?
3. Given a limited number of transcoders, where should these be placed such that the delivered rates across all clients is maximized?

We address the first question in this chapter. Questions 2 and 3 are discussed in Chapters 5 and 6 respectively.

Our initial focus is on establishing that client specified delay tolerance can be leveraged to improve the delivered rates at clients. We develop an optimal algorithm to find the delivered rates at the clients. By modifying the algorithm slightly to eliminate redundant transcoders, we find the optimal transcoder placement to deliver the optimal delivered rates. In all our formulations, we first devise an optimization function for the following reasons: (i) to see if a readily available optimization framework can be used (ii) to compare the results of our algorithm. Whenever we found the optimal solution to be computationally expensive, we have devised other greedy alternatives.

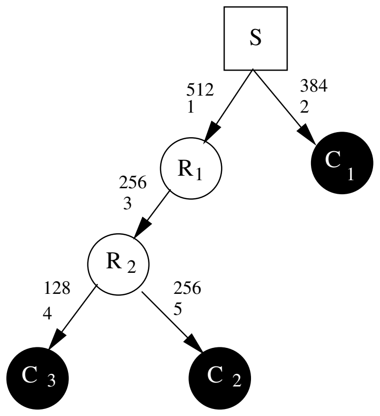


Figure 4.1: An illustrative example

4.1.1 Optimal rates for different placement options: An illustrative example

Source $\mathcal{S}$ is streaming contents to clients C_1, C_2, and C_3 connected through links 1 to 5, with bandwidths in kbps as indicated in Figure 4.1. Let the base encoding rate of the file, Γ be 512 kbps and the duration of the playout, $\mathcal{T}$ be 1 hour. We assume the same minimum rate requirement for all three clients: a minimum rate of 128 kbps. Our objective is to provide the best possible rates to clients in this network without any loss. We consider the following cases:

1. **Source Transcoding (ST), where only the source is capable of transcoding, assuming zero delay tolerance**: Clients require immediate playout without loss (delay tolerance = 0). In this case, the bandwidth of the weakest link in a client's path determines the maximum deliverable rate at that client. Bandwidths of the weakest links in the paths from $\mathcal{S}$ to C_1, C_2, and C_3 are 384 kbps, 256 kbps, and 128 kbps respectively. C_1 gets 384 kbps while both C_2 and C_3 get 128 kbps as they share the first link, and since no transcoding is possible in the relay nodes.

2. **Source Transcoding (ST) considering clients' delay tolerance of 30 minutes**: Suppose buffers are available in the network nodes. While the bandwidth of the weakest link in a client's path determines the rate at which data can be sent, now extra time is available to collect the data at this rate. Thus, when the clients are willing to wait for 30 minutes, the following are the maximum deliverable rates at the clients: C_1: 512 kbps, C_2: 384 kbps and C_3: 192 kbps (We find the amount of data that can be buffered given the extra time of 30 minutes to arrive at the maximum deliverable rates at the clients. A formal discussion

Case	Delivered rates		
	C1	C2	C3
ST when $\delta = 0$	384	128	128
ST when $\delta = 30$	512	192	192
AT when $\delta = 30$	512	384	192
SNT when $\delta = 30$	512	384	192

Table 4.1: Delivered rates for different transcoder placement options

is presented in Section 4.3.1). C_1 gets 512 kbps, being a client directly connected to the source. However, in ST, as no transcoding is possible in the relay nodes, both C_2 and C_3 get 192 kbps.

3. **Anywhere Transcoding (AT), where transcoding capability is available at all relay nodes, considering clients' delay tolerance of 30 minutes**: C_1 still gets 512 kbps, its maximum deliverable rate, and C_2 gets 384 kbps as R_1 transcodes the stream from 512 kbps to 384 kbps for C_2. C_3 gets 192 kbps. However, when transcoders are available both at R_1 and R_2, there may be redundant use of transcoders to deliver 192 kbps to C_3. In this example, two transcoders are used; R_1 converts stream from 512 to 384 kbps, R_2 from 384 to 192 kbps.

4. **Selected Node Transcoding (SNT), where transcoding capability is provided only at selected nodes, considering clients' delay tolerance of 30 minutes**: By restricting transcoding capability only to R_1, we can still deliver the same rates at the clients as in (3) while reducing costs, assuming that R_1 can simultaneously transcode the incoming stream at 512 kbps to two streams encoded at 384 kbps and 192 kbps to serve C_2 and C_3 respectively.

Delivered rates at the clients for the four transcoder placement options discussed above are presented in Table 4.1.

From the above example, it can be seen that the delivered rates at clients depend on the placement of transcoders. Our objective is to develop a solution that determines the optimal delivered rates at clients in a multicast tree when a specified subset of the relay nodes is enabled with transcoders.

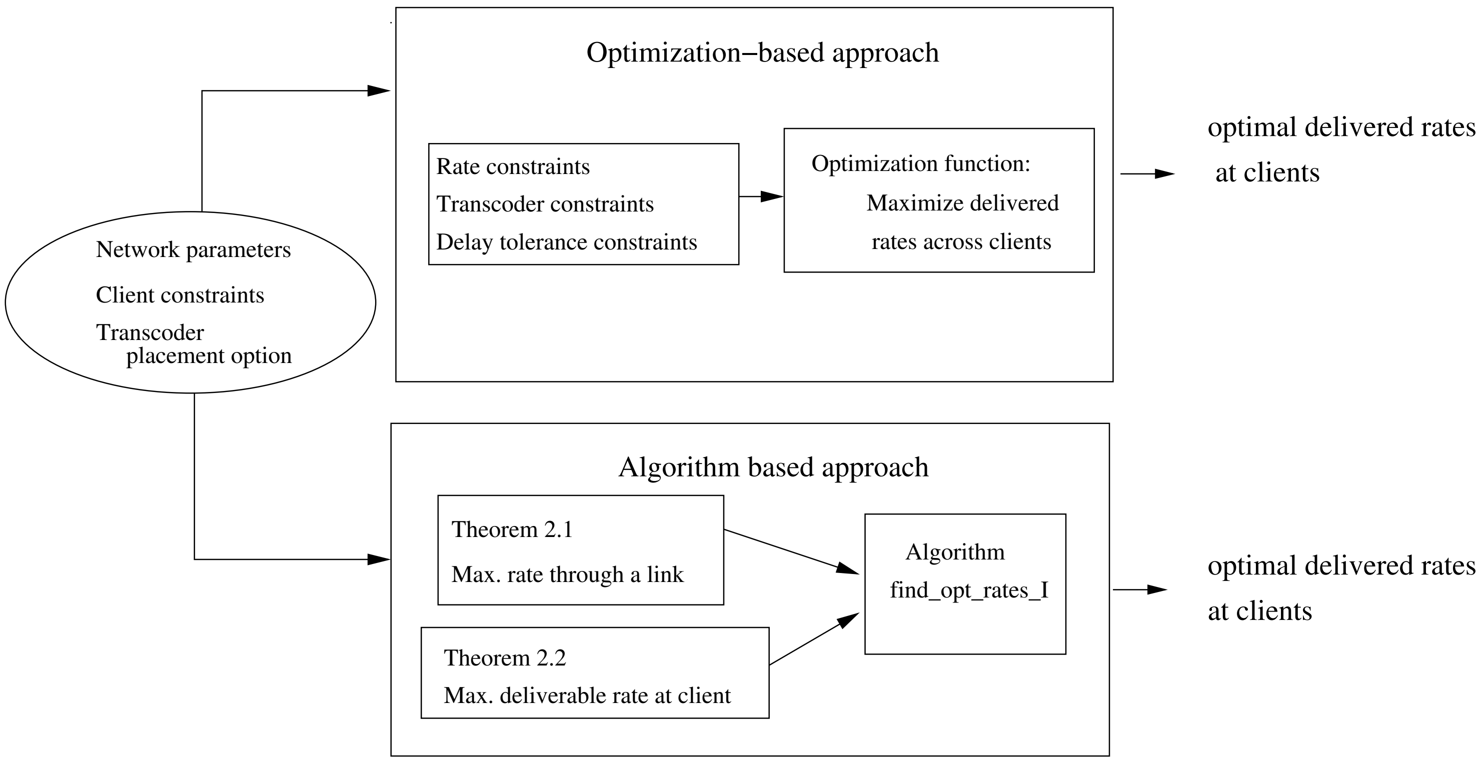

Optimization−based approach
optimal delivered rates at clients
Rate constraints
Transcoder constraints
Delay tolerance constraints
Optimization function:
Maximize delivered rates across clients
Network parameters
Client constraints
Transcoder placement option
Algorithm based approach
Theorem 2.1
Max. rate through a link
Theorem 2.2
Max. deliverable rate at client
Algorithm
find_opt_rates_I
optimal delivered rates at clients

Figure 4.2: Overview of solution approaches: Chapter 4

4.1.2 Overview of solution approaches

We start with an optimization formulation to find the optimal delivered rates at clients, given a specific transcoder placement option. We solve it using the optimization toolbox of **Matlab** [24]. We experimentally show that the optimization based approach can incur computation overheads which make it impractical. Hence we develop algorithmic alternatives.

To this end, we present two theorems; Theorem 4.1 determines the maximum stream rate that can flow through a link and Theorem 4.2 quantifies the maximum deliverable rate at a client when the client is considered in isolation. Using these theorems, we develop an iterative algorithm *find_opt_rates_I* that computes the optimal delivered rates at clients for any given transcoder placement option. We prove the optimality of this algorithm. Figure 4.2 provides an overview of solution approaches presented in this chapter.

4.2 Optimization-based approach

We formulate our objective of maximizing delivered rates across all clients in the network as an optimization problem, where paths from source to every client in the network are considered.

Objective function: Minimize $\forall k, k = 1, 2, \ldots m$, $\sum_k (\Gamma - r_k)^2$,

where m is the number of clients in the multicast tree.

Constraints:

Rate constraint: $\forall i, i = 1, 2, \ldots n$, $r_i <= \Gamma$,

where n is the number of links in the multicast tree.

Transcoder constraint: $\forall n_i$: n_i has a transcoder and r_i is the incoming stream rate and

r_j is the outgoing stream rate, $r_i >= r_j$.

Delay tolerance constraint: $\forall k, k = 1, 2, \ldots m$, $\mathcal{L}_k <= \delta_k$, where $\mathcal{L}_k$ is the

latency incurred in the path from $\mathcal{S}$ to C_k, due to buffering and transcoding.

Figure 4.3: Optimization function

Let n be the number of links in the network and m be the number of clients. Suppose Γ is the base encoding rate and $\mathcal{T}$ is the playout duration of the content.

The optimization function formulated using the *fmincon* function from the optimization toolbox of **Matlab** is presented in Figure 4.3. We explain the optimization function and the constraints below.

Design variables: Stream rates r_is flowing through links l_is.
Objective function: Maximize delivered rates at the clients, written as:
Minimize $\forall k, k = 1, 2, \ldots m$, $\sum_k (\Gamma - r_k)^2$, where and $r_1, r_2, \ldots r_m$ are the rates delivered at the clients.

Choice of objective function: Since transcoding is a process where encoded rates can only be reduced, we choose to maximize the stream rates through every link in the network. Our objective is to minimize the variance between the chosen values while finding the optimal delivered rates at the clients. Hence, we use the square function rather than the linear function which will minimize the mean.

Constraints:

- **Rate constraint**: This constraint ensures that the stream rate flowing across any link in the network is bounded by Γ, which is the highest possible stream rate. $\forall i, i = 1, 2, \ldots n$, $r_i <= \Gamma$.

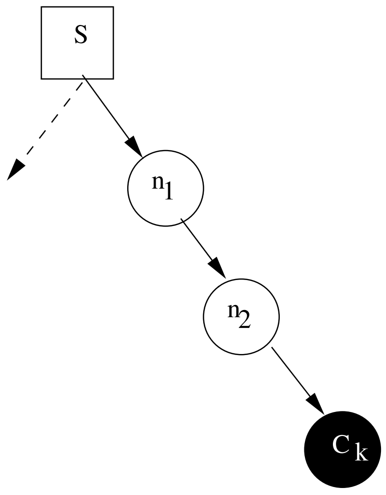


Figure 4.4: Example to illustrate transcoder constraint

- **Transcoder constraint**: This constraint captures the property of transcoding, i.e., the incoming stream rate at a transcoder is greater than or equal to its outgoing rate.

  Consider a client C_k having links l_1, l_2, l_3 in its path from the source $\mathcal{S}$ connected by nodes n_1 and n_2 as shown in Figure 4.4. Let r_1, r_2, r_3 be the stream rates flowing through links l_1, l_2, l_3 respectively. Let both the nodes n_1 and n_2 in the path of C_k from $\mathcal{S}$ have transcoders. In this case, the following conditions hold: $r_1 >= r_2$; $r_2 >= r_3$.

  Consider the case when only one relay node in $p(C_k)$ from $\mathcal{S}$ is capable of transcoding. Suppose node n_1 is capable of transcoding and node n_2 is not capable of transcoding. In this case, the following conditions hold: $r_1 >= r_2$ and $r_2 = r_3$.

  Thus, through an appropriate combination of the equality and inequality constraints on the stream rates flowing through the links, we can emulate any transcoder placement option.

- **Delay tolerance constraint**: This constraint ensures that the client specified delay tolerance δ is not exceeded while delivering enhanced rate to the client. It is specified as: $\forall k, k = 1, 2, \ldots\ m$, $\mathcal{L}_k <= \delta_k$, where $\mathcal{L}_k$ is the latency incurred in the path from $\mathcal{S}$ to C_k, due to buffering and transcoding. We derive the expression for $\mathcal{L}_k$ below.

As explained in Section 2.1.2, we assume the end-to-end transmission delays to be negligible; when they are significant, the delay tolerance value of the client can be reduced appropriately to take these delays into account. There are two other significant factors that contribute to $\mathcal{L}_k$, the latency incurred in the path from $\mathcal{S}$ to a client C_k : (i) the latency introduced due to buffering, L_k^b, when a stream encoded at a higher rate is flowing through a link having lesser bandwidth, and (ii) latency introduced due to transcoding in the relay nodes, L_k^t. Thus, we have the expression,

$$\mathcal{L}_k = L_k^b + L_k^t \tag{4.1}$$

We consider each component of $\mathcal{L}_k$.

In order to provide loss-free transmission to C_k, the highest rate at which data can be sent depends on the bandwidth of the weakest link b_w, in $p(C_k)$ from $\mathcal{S}$. The total amount of data that can be delivered to the client per second, r_k is: $(b_w + (b_w * \delta_k)/\mathcal{T})$, where δ_k is the delay tolerance of C_k. (A formal theorem is presented in Section 4.3.3) . Given the delivered rate r_k at a client C_k and the bandwidth of the weakest link in its path b_w, the expression for latency due to buffering can be derived by substituting L_k^b for δ_k in the above expression and solving for L_k^b.

$$L_k^b = ((r_k - b_w)/b_w) * \mathcal{T} \tag{4.2}$$

As explained in Chapter 2, due to the real-time nature of the transcoding process and the pipelined property of streaming, irrespective of the number of transcoders used in the path of a client, we assume the transcoding latency to remain constant. Thus, we include a small constant transcoding latency L_k^t, the worst-case delay that may result from deploying transcoders.

As mentioned, we implemented the optimization function using **fmincon** from the optimization toolbox of **Matlab** to find the optimal delivered rates at the clients. Function **fmincon** uses *Sequential Quadratic Programming* (SQP) optimization method [38]. It is a gradient descent based search algorithm in the continuous search space. It starts from an initial point and usually converges to a constrained local optimum close to the initial point.

Complexity of the optimization formulation

Considering our optimization formulation, suppose we have an average of n links in the path of each client, in the worst case the optimizer solves the problem by trying all possible values for the q distinct values corresponding to the stream rates that flow across the links. A total of n^q computations are needed to find the delivered rates at a given client. For example, if there are 20 links in the path a client, with the stream rates flowing through each link taking at minimum one of 15 possible values, the search space is of the order of 20^{15}.

4.3 Effect of delay tolerance on stream rates through links

Our objective is to find the loss-free delivered rates at clients in a multicast tree given: (i) client requirements and (ii) a transcoder placement option. The following parameters affect the delivered rates at clients:

1. Bandwidth of the weakest link in a client's path.
2. Delay tolerance of the client.
3. Nature of links in a client's path – whether a given link is shared or not.
4. Resources available in the nodes in a client's path – whether a given node is capable of transcoding or not.

In order to develop an optimal algorithm that finds the delivered rates at clients for any transcoder placement option, we need to understand the impact of these parameters on the stream rates that flow through the links in the multicast tree.

In this section, we present an analysis on the impact of delay tolerance on the rates delivered at the clients. We first present a lemma that captures the impact of delay tolerance on the stream rate that flows through a link that is not shared. We then present two theorems: the first generalizes the lemma to find the upper bound on the stream rate that can flow through any link in a multicast tree; the second finds the maximum deliverable rate at a client considering the client in isolation.

4.3.1 Maximum stream rate through an unshared link

We first consider a link that is in the path of a single client, to understand the effect of delay tolerance on the maximum stream rate that can flow through the link without loss.

Lemma 4.1: Consider a link l_i having bandwidth b_i in the path of a client C_k from the source $\mathcal{S}$ as shown in Figure 4.5. The maximum stream rate that can flow through l_i for a loss-free transmission across that link is given by:

$$r_i = b_i * (1 + (\delta_k/\mathcal{T})) \tag{4.3}$$

where δ_k is the delay tolerance of C_k and $\mathcal{T}$ is the playout duration of the contents.

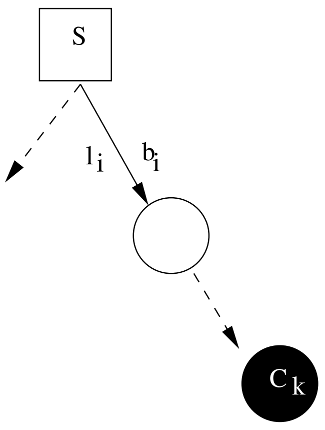


Figure 4.5: Determining stream rate through an unshared link

Proof:

In order for the stream to flow across l_i without loss, the stream has to be encoded at most at b_i, when $\delta_k = 0$. Given the additional time δ_k, $(\delta_k \times b_i)$ additional bits can flow across l_i over time $\mathcal{T}$. Thus, the additional amount of data that can flow through l_i per second is $(\delta_k \times b_i)/\mathcal{T}$. Hence, the maximum rate that can flow through l_i without loss is given by: $r_i = b_i \times (1+(\delta_k/\mathcal{T}))$.
□

4.3.2 Maximum stream rate through any link

Lemma 4.1 provides the stream rate that can flow through a link, considering the link in isolation. When a link is shared, maximum stream rate that can flow through the link depends on the delay tolerance values of the clients sharing the link, to provide loss-free playout at the clients. We generalize Lemma 4.1 to find the maximum stream rate that can flow through any link, shared or not, to provide loss-free playout at the clients. Although this theorem is intuitively obvious, we provide a formal statement and proof, for the sake of completeness.

Theorem 4.1:

Consider a link l_i having bandwidth b_i in the path of one or more clients $C_1, C_2, \dots C_m$. r_i^{max}, the maximum stream rate that can flow through l_i is given by:

$$r_i^{max} = b_i * (1 + (\delta_p/\mathcal{T})) \tag{4.4}$$

where $\delta_p = min(\delta_k)$, δ_ks are the delay tolerance values of C_ks, $k = 1, 2, \dots m$.

Proof:

We use induction to prove this theorem.

- *Base case*: Consider the case when l_i is in the path of a single client C_k. Let δ_k be the delay tolerance of C_k. By Lemma 4.1, the maximum stream rate that can flow through l_i is given by:
  $b_i * (1 + (\delta_k/\mathcal{T}))$. Since C_k is the only client served by link l_i, $\delta_p = min(\delta_k) = \delta_k$.
- *Hypothesis*: Now consider the case when l_i is the shared link in the path of clients $C_1, C_2, \dots C_k$. Equation 4.4 holds when clients $C_1, C_2, \dots C_k$ share the link.
- *To prove*: We need to prove that Equation 4.4 also holds when the link is shared by clients $C_1, C_2, \dots C_{k+1}$.
- Let δ_{k+1} be the delay tolerance of client C_{k+1}. If $\delta_{k+1} > \delta_p$ the equation is not affected. Let $\delta_{k+1} < \delta_p$. Suppose the stream is encoded at $b_i * (1 + (\delta_p/\mathcal{T}))$, client C_{k+1} would experience lossy playout. In order for all clients to experience loss-free playout, the stream needs to be encoded at $min(\delta_1, \delta_2, \dots \delta_{k+1}) = \delta_{k+1}$. Hence the maximum stream rate that can flow through l_i is bounded by $b_i * (1 + (\delta_p/\mathcal{T}))$. $\square$

4.3.3 Maximum deliverable rate at a client – when considered in isolation

We considered a single link thus far. In this section we consider a set of links that are in the path of a client from the source. We derive the expression for the upper bound on the delivered rate at a client, *maximum deliverable rate*, considering the links in its path in isolation. Given that we have three types of nodes, source, relay, and client, we first consider a simple topology with one of each type of node. We refer to this three-node topology as the *basic block* presented in Figure 4.6.

Theorem 4.2:
Given a source $\mathcal{S}$ serving a client C_k connected through a set of relay nodes $R_1, R_2, \dots R_n$ and links $l_1, l_2, \dots l_{n+1}$ having bandwidths $b_1, b_2, \dots b_{n+1}$. Maximum deliverable rate γ_k^{max} at C_k is given by:

$$\begin{aligned} \gamma_k^{max} &= b_w * (1 + (\delta_k/\mathcal{T})) && if\ \ b_w * (1 + (\delta_k/\mathcal{T})) < \Gamma \\ &= \Gamma && otherwise; \end{aligned} \tag{4.5}$$

where $b_w = min(b_1, b_2, \dots b_{n+1})$, bandwidth of the weakest link in $p(C_k)$ and $\mathcal{T}$ is the playout duration of the content having base encoding rate Γ.

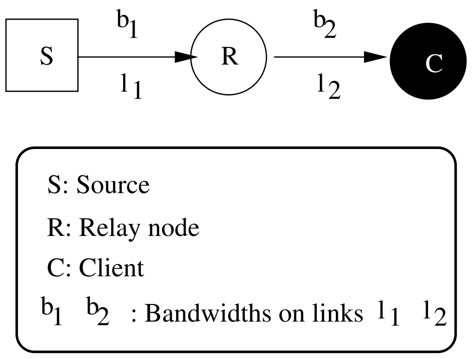


Figure 4.6: Basic block

Proof:

We use induction to prove this theorem. We consider Part 1 of the equation.

- *Base case*: Consider the basic block, having a path $\mathcal{S}$–R-C as shown in Figure 4.6. Without loss of generality, let l_1 be the weakest link such that $min(b_1, b_2) = b_1$;

  To provide loss free playout, the stream has to be encoded at b_1, when the delay tolerance is zero. Given δ_k, by Lemma 4.1, a maximum rate of $b_1 * (1 + (\delta_k/\mathcal{T}))$ can flow through link l_1 to provide loss-free playout to C_k. Hence,
  $\gamma_k^{max} = b_1 * (1 + (\delta_k/\mathcal{T}))$.

- *Hypothesis*: Consider a path $\mathcal{S}$–R_1–R_2–$\ldots$-R_n–C_k, having bandwidths $b_1, b_2, \ldots\ b_{n+1}$. Part 1 of equation 4.5 holds for this path.

- *To prove*: We need to prove that part 1 of equation 4.5 also holds for the path: $\mathcal{S}$–R_1–R_2–$\ldots$-R_n– R_{n+1}–C_k.

  Let **min**$(b_1, b_2, \ldots\ b_{n+1}) = b_w$.
  When an additional node is inserted into the path, it introduces one additional link, l_{n+2}. Let b_{n+2} be the bandwidth of this link. If $b_{n+2} >= b_w$, part 1 of equation 4.5 does not change since b_w remains the bandwidth of the weakest link. However, when $b_{n+2} < b_w$, if b_w is used to calculate γ_k^{max}, this rate cannot be supported by l_{n+2} without loss. Thus, in order for C_k to receive loss-free transmission, the maximum encoded rate is bounded by b_{n+2}, which is **min**$(b_1, b_2, \ldots\ b_{n+2})$, the new b_w. Hence we have,
  $\gamma_k^{max} = b_w * (1 + (\delta_k/\mathcal{T}))$.

- Suppose $\gamma_k^{max} > \Gamma$. Given that Γ is the best possible rate at which the stream can be delivered, $\gamma_k^{max} = \Gamma$, as in part 2 of equation 4.5. $\square$

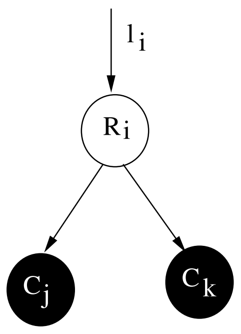


Figure 4.7: Clients sharing a link

Corollary 4.1:
The maximum deliverable rate γ_k^{max} determined using Theorem 4.1 is the upper bound on the delivered rate to client C_k. i.e., γ_k^{max} is the upper bound on r_k.
Proof:
Consider clients C_j and C_k sharing a link l_i as shown in Figure 4.7. Let r_j and r_k be the maximum deliverable rates at C_j and C_k as calculated by Theorem 4.1. Without loss of generality, let $r_j < r_k$. Note that a single stream serves both C_j and C_k flowing through the common link l_i.

- When there is no transcoder in relay node R_i, r_k cannot be chosen as the encoding rate for the stream as C_k will experience lossy playout. Hence r_j is chosen so that both the clients are served without loss. Note that while C_j gets its maximum deliverable rate r_j, C_k also gets r_j, which is less than its maximum deliverable rate r_k.

- When a transcoder is available at R_i, stream encoded at r_k is delivered to C_k. Using the transcoder at R_i stream is transcoded to r_j and delivered to C_j.

Extending the logic to any number of clients sharing one or more links, a client at best can get the maximum deliverable rate as calculated by Theorem 4.1. Thus, for any client C_k, $r_k <= \gamma_k^{max}$. $\Box$

The theorems discussed so far help us to develop an efficient iterative algorithm *find_opt_rates_I* for finding the optimal rates delivered at the clients for a given transcoder placement option.

4.4 An iterative algorithm to find optimal rates delivered at clients

In this section, we discuss algorithm *find_opt_rates_I* in detail. Before we discuss the algorithm, we take the same example as in Figure 4.1 and illustrate the variables and data structures used in the algorithm. These variables and data structures are also used for all the algorithms that we have developed in this thesis.

4.4.1 Variables and data structures used

The network topology is specified as a $n \times n$ matrix, where n is the number of nodes in the network. An entry M(i,j) provides the bandwidth available on the link between nodes i and j. M(i,j) is 0, when nodes i and j are not directly connected. Client requirements include the minimum rate requirement and delay tolerance requirement of each client. In addition to these two global parameters, transcoder placement option is given as input to the algorithm. The algorithm determines the stream rates flowing through the links and the delivered rates at the clients. The input and output variables and the global constants are described in Figure 4.8.

We take the example of Figure 4.1 to illustrate the input and output variables and the global constants, presented in Figure 4.9. Data structures accessed as global variables are described in Figure 4.10. Illustrations of these data structures for the example are presented in 4.11.

4.4.2 Algorithm for pass 1

The algorithm uses two passes. In the first pass, the algorithm starts with links from the leaf nodes of the multicast tree. For each link, the maximum stream rate that can flow through that link without loss is determined. Note that the maximum stream rate that can flow through a link depends on the following three factors:

1. *Bandwidth available on that link*: Given the bandwidth available on the link, maximum rate that can be supported by the link is calculated using Lemma 4.1; this rate provides the upper bound on the stream rate that *can* be supported by the link without loss.

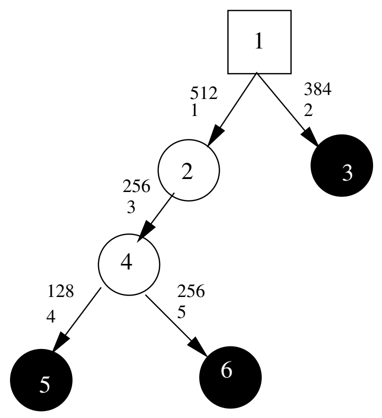
```
Input:
% the following global variables are used as input by the algorithm %
M: Multicast tree represented as a n x n matrix, where n is the
   number of nodes in the network
cli_req: three-columned matrix, indexed by the client-id, having
          the node-id in the first column, values of minimum rate
          required in the second column, and delay tolerance in
          the third.

% the following input parameter is passed to the algorithm %
transinfo: Boolean vector of size n, where n is the number of nodes
           in the network, that indicates nodes having transcoders

Output:
actual_rates: Vector of stream rates flowing through each link
optim_rates: Vector of optimal rates delivered at clients

Global constants:
TRANSDURATION: Playout duration of the contents
BASEENCODINGRATE: The best rate at which the contents are encoded
```

Figure 4.8: Input/output variables and global constants



GLOBAL CONSTANTS:

TRANSDURATION: 3600

BASEENCODINGRATE: 512

INPUT:

$$M = \begin{bmatrix} 0 & 512 & 384 & 0 & 0 & 0 \\ 0 & 0 & 0 & 256 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 128 & 256 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

cli_req =

nodeid	minrate	delay tol.
3	128	1800
5	128	900
6	128	1800

$$\text{transinfo} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

OUTPUT:

$$\text{actual\_rates} = \begin{bmatrix} 384 \\ 512 \\ 384 \\ 160 \\ 384 \end{bmatrix}$$

$$\text{optim\_rates} = \begin{bmatrix} 512 \\ 160 \\ 384 \end{bmatrix}$$

Figure 4.9: Illustration of variables and global constants

```
numnodes: Scalar value of number of nodes in the network
numlinks: Scalar value of number of links
numclients: Scalar value of number of links
clients: Matrix that provides following information
on each client in the multicast tree:
  client id, node id, minimum required rate, delay tolerance,
  number of nodes in path, weakest link in path.

This matrix is created by concatenating two more columns to
the client constraint table.

clientinfo: a three-dimensional matrix that provides the node ids
and the bandwidths on links that connect the nodes in the paths
of each client.

linkinfo: Matrix that provides the following information
on each link in the multicast tree:
  from node, to node, number of clients sharing the link,
  clients sharing the link

pathinfo: Matrix that provides link ids in the path of
each client in the multicast tree.

subtreeinfo: Matrix that provides the following information
for each node in the network:
    nodes connected to it, number of subtrees emanating from it,
    depth, clients served by the node
```

Figure 4.10: Global variables used in the algorithms

numnodes = 6;
numlinks = 5;
numclients = 3;

	nodeid	minrate	delay tol.	nodes in path	weakest link b/w
clients =	[3	128	1800	2	384
	5	128	900	4	128
	6	128	1800	4	256]

		nodeid	link b/w
clientinfo =	(:,:,1)	[3	0
		1	384]
	(:,:,2)	[5	0
		4	128
		2	384
		1	512]
	(:,:,3)	[6	0
		4	256
		2	384
		1	512]

	from node	to node	num clients sharing link	clients		
				1	2	3
linkinfo =	[1	2	2	0	1	1
	1	3	1	1	0	0
	2	4	2	0	1	1
	4	5	1	0	1	0
	4	6	1	0	0	1]

	clients		
	1	2	3
pathinfo =	[2	4	5
	0	3	3
	0	1	1]

	node id	next node (var. length)		num subtrees	level num	clients		
						1	2	3
subtreeinfo =	[1	2	3	2	0	1	1	1
	2	4	0	1	1	0	1	1
	3	0	0	0	1	1	0	0
	4	5	6	2	2	0	1	1
	5	0	0	0	3	0	1	0
	6	0	0	0	3	0	0	1]

Figure 4.11: Illustration of data structures used in the thesis

2. *Maximum of the maximum deliverable rates at the clients served by that link*: For each client served by a link, its maximum deliverable rate is calculated using Theorem 4.2. Maximum of these rates provides the upper bound on the stream rate that *needs to be* supported by the link without loss.

3. *Whether the node at the terminating end of the link has a transcoder or not*: In addition to the above two factors, the stream rates at which the clients having the link in their paths are served without loss depends on availability of transcoders at the nodes.

The first two factors enforce the *delay tolerance constraints* to ensure loss-free playout at the clients. The third parameter enforces *transcoder constraints*.
To understand the first pass of the algorithm, we consider the instances of links that occur in the multicast tree as shown in Figure 4.12. Note that a link can be directly attached to a client as in Figure 4.12 (i) or it can be attached to a relay node having one outgoing link as in Figure 4.12 (ii) or many outgoing links as in Figure 4.12 (iii).

1. In the first case, the maximum rate that can flow through the link l_j is determined by the maximum deliverable rate at the client C_a, the link is attached to. (Note that finding the maximum rate supported by the link calculated using Lemma 4.1 is redundant in this case).

2. In the second and third cases, the maximum rate that can flow through the link l_j depends on whether the node n_j has a transcoder or not in addition to the bandwidth available on that link. Note that since we compute the maximum rate flowing through links from the bottom to top of the tree, when transcoder is available at n_j, maximum of the rates computed for the outgoing links from n_j can flow through l_j provided it can be supported by l_j without loss. when there is no transcoder at n_j, only minimum of the rates can flow through l_j.

We provide the algorithm for Pass 1 in Figure 4.13. Note that when there is only one outgoing link from n_j, rate of the stream flowing through that link is assigned to l_j, irrespective of whether there is a transcoder at n_j or not.

Thus, in the first pass, the algorithm determines the maximum rate that can flow through each link in the multicast tree.

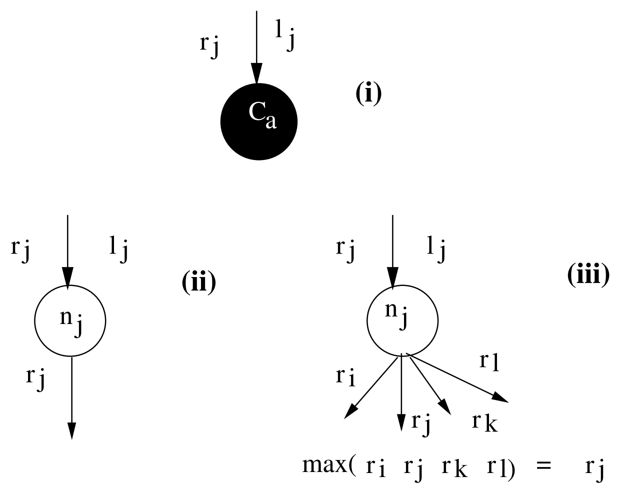


Figure 4.12: Instances of links

4.4.3 Algorithm for pass 2

In the second pass, the algorithm starts with links from the top of the tree. The algorithm traverses the tree from the top to bottom tracing the path of each client in the tree. It checks that the stream rate flowing through a link in the client's path is less than or equal to the stream rate flowing through the previous link in the path, as the stream can only be transcoded down. This step enforces the *rate constraints*. At the end of this step, the *actual rates* flowing through the links are determined. For each client, the actual rate flowing through the last link in its path determines the delivered rate at that client.

We present the algorithm for Pass 2 in Figure 4.14. In the next section, we prove that this algorithm determines optimal delivered rates at clients for any transcoder placement option.

4.4.4 Optimality of *find_opt_rates_I*

Claim 4.1:

Algorithm *find_opt_rates_I* finds the optimal rates delivered at the clients for a given transcoder placement option.

To prove: Let r_k be the delivered rate at client C_k as determined by *find_opt_rates_I*. A rate $r_k+\varepsilon$, $\varepsilon > 0$ can be delivered to C_k, only if the constraints of one or more clients are violated.

In other words, we need to prove that in each pass of the algorithm, the stream rates assigned to each link is the maximum that can flow through that link without violating any constraint.

Proof: By contradiction

```
function find_opt_rates_I (transinfo);
Initialize stream_rate [numlinks]
% stream_rate is a vector of dimension numlinks %
Initialize optim_rates [numclients]
% optim_rates is the output vector of dimension numclients %
% start pass 1 %
for each link lj % Starting from the leaf nodes of the tree %
    if link is directly attached to a client
       max_stream_rate =  find_max_deliverable_rate (Ci)
        % Calculated using Theorem 4.2 %
    else
   max_rate_thru_link  =  linkbandwidth(1+ bj/T);
       if there is a transcoder at node ni
          % at the terminating end of the link %
          max_stream_rate  =
          max(stream rates flowing through
          the outgoing links from ni);
       else
           max_stream_rate  =
           min(stream rates flowing through
           the outgoing links from ni);
       end
    end
    stream_rate [li] =
              min(max_rate_thru_link, max_stream_rate);
    % Apply Theorem 4.1 to assign appropriate rate to link %
end
```

Figure 4.13: Pass 1 of algorithm: *find_opt_rates_I*

```
% start pass 2%
for each client C
    for link L in Cs path starting from S
    if L is the first link in path
     max_rate  =  stream_rate [L]
     % Traverse path from source to client and
     assign stream rate of first link to max_rate %
    else
      if stream_rate [L] is greater than max_rate
           actual_rate [L] = max_rate;
      else
           actual_rate [L] = stream_rate [L];
           max_rate = stream_rate [L];
      end
    end
    optim_rates [C] =
            actual_rate [last link in C's path]
end
```

Figure 4.14: Pass 2 of algorithm: *find_opt_rates_I*

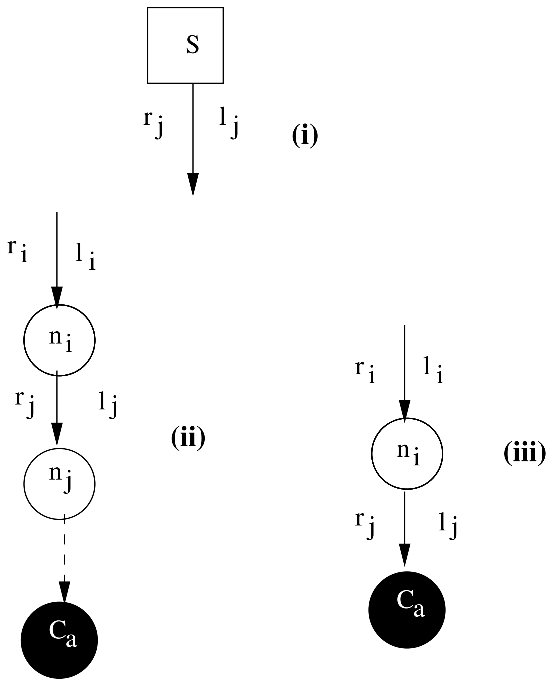


Figure 4.15: Instances of links in the path of a client

Let us consider any link l_j in the path of any client C_k in the multicast tree. Let l_j be the incoming link to node n_j.

- By Theorem 4.2, the maximum deliverable rate at C_k is given by $\gamma_k^{max} = b_1 * (1 + (\delta_k/\mathcal{T}))$ where b_1 is the bandwidth of the weakest link in $p(C_k)$. Thus, any link in $p(C_k)$ requires to support γ_k^{max} at maximum.
- By Theorem 4.1, the maximum rate that can flow through l_j is bounded by $r_j^{max} = b_j * (1 + (\delta_p/\mathcal{T}))$ where $\delta_p = min(\delta_k)$, δ_ks are the delay tolerance values of clients sharing the link l_j.

Consider Pass 1 of the algorithm with the following possibilities for link l_j as illustrated in Figure 4.15:

- Let l_j be directly connected to C_k. Two possibilities exist: (i) l_j is weakest link in $p(C_k)$ (ii) l_j is not the weakest link in $p(C_k)$. In the first case, γ_k^{max} is the maximum stream rate that can flow through l_j. In the second case, r_j^{max} is the maximum rate that can be supported by l_j. However r_j^{max} which is greater than γ_k^{max} will violate the requirement of loss-free playout at C_k. Thus, in either case, γ_k^{max} is the maximum rate that can flow through l_j as determined by the first pass of the algorithm.

- Let l_j be any intermediate link occurring in the multicast tree. Let r_j be the rate assigned to l_j. We have to prove that any rate higher than r_j assigned to l_j will violate one/more constraints or would lead to redundant resource use without having any effect on the delivered rates at the clients it is serving.

  Note that r_j is assigned to l_j by considering the following three parameters: (i) b_j, the actual available bandwidth on l_j, (ii) $\delta_p = min(\delta_k)$, where δ_ks are the delay tolerance values of clients sharing the link l_j, and (iii) availability of transcoder at node n_j. According to the algorithm, maximum value of $r_j =$ max(rates flowing through outgoing links from n_j). We need to consider the following cases:

  - With reference to Figure 4.12 (ii), suppose $r_j^{max} > r_j$. When there is no transcoder at n_j, assigning r_j^{max} to l_j would violate the requirement of loss-free delivery at the clients served by l_j.

    When there is a transcoder at n_j, r_j^{max} *can* be assigned to l_j. In this case r_j^{max} will be flowing through l_j while r_j is the rate flowing through the outgoing link from n_j. Note that in this case the transcoder at n_j is redundant as both the links are part of a string, serving the same set of clients. Hence having r_j^{max} flow through l_j does not affect the delivered rates at the clients it is serving.

  - With reference to Figure 4.12 (iii), when there is no transcoder at n_j, assigning any rate greater than r_j to l_j would violate the requirement of loss-free delivery at the clients.

    When there is a transcoder at n_j, r_j^{max} *can* be assigned to l_j. In this case r_j^{max} will be flowing through l_j while r_j, max(stream rates through outgoing links from n_j) is less than r_j^{max}. The transcoder at n_i would reduce the stream rate to r_j and other rates less than r_j, in order to serve all the clients without violating their requirements. Note that having r_j^{max} flow through l_j does not affect the delivered rates at the clients.

  Thus, the first pass of the algorithm assigns stream rates to the links that would maximize the delivered rates at the clients.

Consider Pass 2 of the algorithm. This step considers links in the path of each client from source to adjust the rates flowing through the links. The rate flowing through the last link in a client's path is its delivered rate. Note that in the first pass, last link in every client's path is assigned the client's maximum deliverable rate. Let us consider the three possibilities for link l_j in the path of a client C_k as illustrated in Figure 4.15. Let r_j be the stream rate assigned to l_j in pass 1.

1. Let l_j be the first link in $p(C_k)$ as shown in 4.15 (i). As proven above, this is the maximum rate that can flow through l_j without violating any requirement and without affecting delivered rate at any client. Thus the actual rate flowing through l_j is r_j.

2. Let l_j be any intermediate link occurring in $p(C_k)$ as shown in 4.15 (ii). Let r_i be the stream rate flowing through the incoming link l_i into node n_i. The following three possibilities exist:

   (a) $r_j < r_i$; in this case as r_j is the maximum stream rate that can flow through l_j, increasing r_j would violate the requirement for C_k. Thus r_j remains unchanged in this case.

   (b) $r_j = r_i$; in this case also r_j remains unchanged.

   (c) $r_j > r_i$; in this case since the incoming stream rate r_i is less than r_j, the maximum rate that can flow through l_j is limited by r_i. Hence the actual rate that flows through l_j is changed to r_i. Note that irrespective of the rate assigned to the link in the first pass, the highest rate that can actually flow through the link depends on the actual rate of the incoming stream.

3. Let l_j be the last link in $p(C_k)$. Using the same argument in the previous case, the actual rate flowing through l_j depends on the actual rate r_i, flowing through its previous link in $p(C_k)$. When $r_j <= r_i$, r_j is the delivered rate at C_k; else r_i is the delivered rate at C_k.

Extending the same logic to links in every client's path, the delivered rates at the clients as computed by Pass 2 of the algorithm are the maximum rates that can delivered to the clients without violating any requirement.
Thus, algorithm *find_opt_rates_I* finds the optimal delivered rates at the clients. Since the presence or absence of a transcoder at every node is considered while computing the maximum stream rate that can flow through each link, this algorithm finds the optimal delivered rates at clients for any given transcoder placement option. □

4.4.5 Complexity of *find_opt_rates_I*

Consider a multicast tree having L links and C clients. Let D be the number of levels in the multicast tree, also referred to as its *depth*. Let us consider the steps in the algorithm to estimate the number of computations required and the order of complexity of the algorithm:

In the first pass, starting from the leaf nodes of the tree, for each link the maximum possible rate that can flow through the link is determined:

- When the link is directly connected to the client, the maximum deliverable rate of the client is found. This step involves finding the weakest link in the client's path. Given a tree with D levels, the number of computations is of $O(DC)$ (which is $>$ L).
- When the link is any intermediate link connected to a node n_i, the stream rates through all the out going links from n_i are considered. Let F be the maximum number of out going links from any node in the tree. The number of computations required is $O(F)$, where $F << L$.
- The maximum rate supported by the link without violating the delay tolerance requirement of any client having the link in its path is found. Let K be the maximum number of clients having the link in their paths. The number of computations required is $O(K)$, where $K < C$.

Thus, computations required for the first pass of the algorithm is $O(DC)$.
In the second pass, links are traversed from top to bottom, to ensure appropriate rates are assigned to each link. The number of computations for this step is $O(L)$.
Thus, algorithm *find_opt_rates_I* requires $O(DC)$ computations.

4.5 Experimental demonstration of effect of delay tolerance: Gnutella Peer Network

In this section, we present the results of experiments to study the effect of delay tolerance on the delivered rates at the clients. We have implemented the algorithm *find_opt_rates_I* using **Matlab** and have conducted extensive experiments to understand the implication of clients' delay tolerance and link bandwidths, on the delivered rates at the clients.

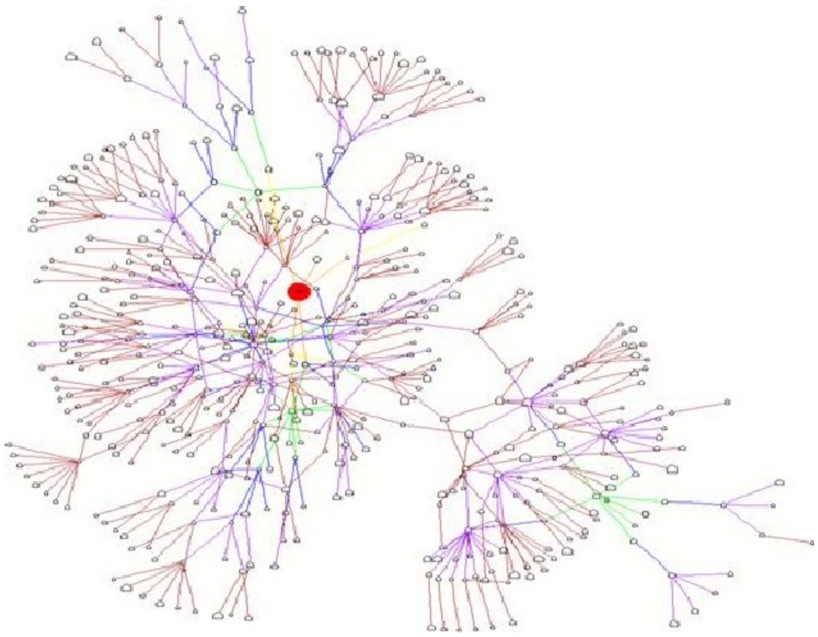


Figure 4.16: An instance of GNU peer network

In our performance analysis, we have assumed a constant minimum rate requirement of 128 kbps for all the clients and concentrated on studying the effects of different delay tolerance values of the clients. Note that using our algorithm we can devise an admission control module, when clients specify different minimum rates. Given the knowledge of nodes with transcoders, the admission control module would run the algorithm to see if any clients minimum rate requirement is not satisfied. If so, that client will not be admitted for service.

We present a case study on the Gnutella Peer Network [8] (See Figure 4.16). We simplify the original network topology to a tree-based network by removing cycles in the topology. Our approximated Gnutella Peer distribution network contains 510 nodes and 10 levels. In this simulation, we set the clients' minimum rate requirement to 128 Kbps. We run algorithm *find_opt_rates_I*, for 40 sets of delay tolerance values with the AT transcoder placement option. Delay tolerance values of the clients for each run are randomly chosen from $(0 * \mathcal{T} - 0.1 * \mathcal{T})$, $(0.1 * \mathcal{T} - 0.2 * \mathcal{T})$, ..., $(0.9 * \mathcal{T} - \mathcal{T})$, ..., $(3.9 * \mathcal{T} - 4 * \mathcal{T})$ respectively.

We plot the range of delay tolerance values for clients (as a fraction of the playout duration) along the X-axis and the average delivered rates at the clients along the Y-axis as shown in Figure 4.17. As seen from the graph, as the delay tolerance values of the clients increase, the average delivered rates increase. When the delay tolerance values are chosen from $(3.0 * \mathcal{T} - 3.1 * \mathcal{T})$, the average delivered rate is 512 kbps which is the base encoding rate of the file. Note that increasing the delay tolerance values beyond this range does not improve the average

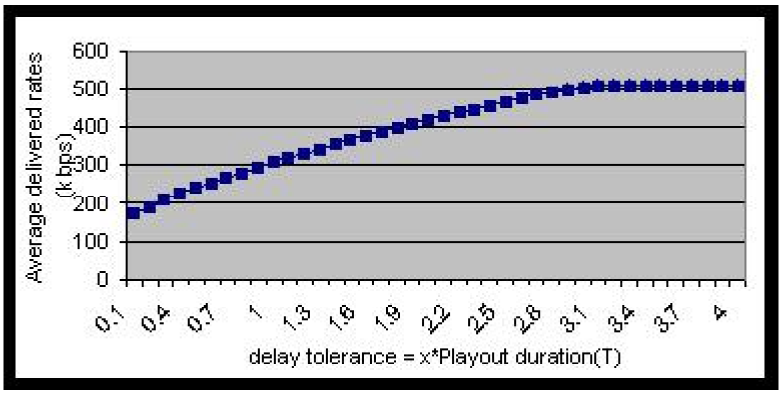


Figure 4.17: Effect of delay tolerance on delivered rates

delivered rates, as the best possible rate that can be delivered to any client is bounded by the base encoding rate, 512 kbps.

4.6 Conclusions

Given a subscription-based service for disseminating contents synchronously to clients, where each client specifies the time it is willing to wait for transmission to start, the first question that needs to be answered is: what are the optimal rates that can be delivered to these clients, given a transcoder placement option. In this chapter, we show that the client specified delay tolerance can be leveraged to enhance the delivered rates at the clients. Links that are shared by clients and the availability of transcoders at relay nodes are the other factors that determine the delivered rates at clients. We analyze network properties affected by these factors. Using the properties we develop an algorithm to find the optimal delivered rates at clients for a given transcoder placement option and prove the optimality of this algorithm. The analysis and algorithm presented in this chapter are very relevant for a CSP catering to a diverse set of subscribed clients who may have bandwidth constraints along their path from the source. This analysis is also useful for a CSP to introduce various classes of service (say, Platinum, Gold, and Silver memberships) as, given a specific playout start time at a client, the CSP would be able to guarantee a loss-free rate.

When all the relay nodes are transcoder capable, algorithm *find_opt_rates_I* finds the optimal rates delivered at the clients which are the *best* rates that the clients can get given the network characteristics and the clients' delay tolerance requirements. We discuss the optimal placement of transcoders for providing best delivered rates to clients in the next chapter.

Chapter 5

Determining optimal placement of transcoders

(Scheduled streaming, Static link bandwidths)

5.1 Introduction

As discussed in the last chapter, transcoder placement option refers to the combination of nodes in the network which have transcoders. There we presented algorithms to compute the optimal rates delivered at clients in a multicast tree for a given transcoder placement option. Before we pose the question that we address in this chapter, we recap some definitions and introduce some new ones which are used to state the question precisely:

Maximum deliverable rate: Maximum deliverable rate is the stream rate that can be delivered to a client considering its path from the source in isolation. For a given client C_i, its maximum deliverable rate is denoted by γ_i^{max}.
Recall that this rate depends on the bandwidth of the weakest link in the client's path and the client's delay tolerance.

Optimal rate: Optimal rate is the highest loss-free rate that can be delivered to a client in the multicast tree for a given transcoder placement option.

Best delivered rate: The optimal rate delivered at a client, when all relay nodes are capable of transcoding is referred to as its best delivered rate.
Note that for a given network with static link bandwidths and specific client requirements, best

delivered rates provide the upper bound on the delivered rates at the clients.

Eligible set: If there are N nodes and C clients in the tree, the maximum number of transcoders that can be placed, assuming that the source always has a transcoder, is bounded by: $(N - (C + 1))$. This set is referred to as the eligible set.

Optimal set: The set of nodes where transcoders have to be placed to obtain the best delivered rates for all clients such that the number of transcoders deployed is minimal. The optimal set is denoted by $\mathcal{O}$.

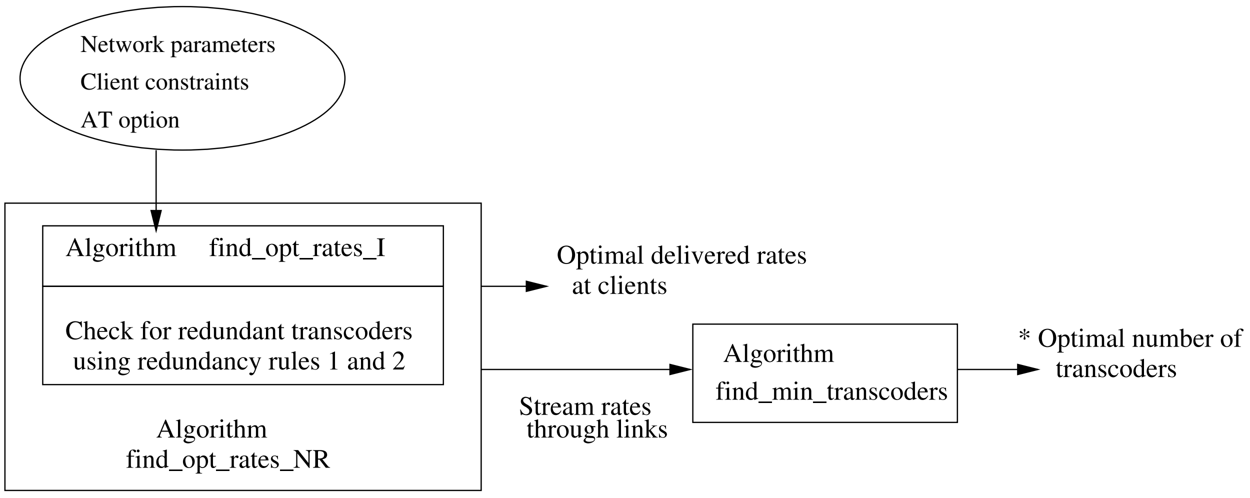
In this chapter we find the optimal set $\mathcal{O}$, i.e., address the following question:
What is the minimum number of transcoders required and where should they be placed to provide the best delivered rates to clients?

In a multicast tree with specific link bandwidths and client requirements, when AT placement option is used (when all nodes in the eligible set are transcoder capable), the optimal rates computed are the best delivered rates at the clients. Even though all relay nodes are transcoder capable, not all of them are required to provide the best delivered rates to clients. Thus, there is a set of transcoders which are necessary and sufficient for providing the best delivered rates to clients termed the **Optimal set** denoted by $\mathcal{O}$. Our objective in this chapter is to develop an algorithm to determine $\mathcal{O}$.

5.1.1 Overview of solution approach

As explained in the previous chapter, *find_opt_rates_I* determines the actual stream rates flowing through each link in the multicast tree and the optimal rates delivered at clients, for any given transcoder placement option. When there is no restriction on the number of transcoders, this algorithm finds the best delivered rates at clients; these rates can not be improved with any additional transcoders, given the multicast tree and client requirements. We can find the set of transcoders that are enabled for providing the best delivered rates using *find_opt_rates_I*. However, as we shall see, *find_opt_rates_I* might employ some transcoders that can be shown to be redundant.

We analyze the properties of the network related to placement of transcoders and come up with observations to eliminate redundancy when all relay nodes are capable of transcoding. Using this insight, we modify *find_opt_rates_I* to include checks to ensure that only the nec-



Network parameters
Client constraints
AT option
Algorithm find_opt_rates_I
Check for redundant transcoders using redundancy rules 1 and 2
Algorithm find_opt_rates_NR
Optimal delivered rates at clients
Stream rates through links
Algorithm find_min_transcoders
* Optimal number of transcoders

Figure 5.1: Overview of solution approaches: Chapter 5

essary and sufficient set of transcoders (required to provide best delivered rates) are enabled. We call this algorithm *find_opt_rates_NR*, where NR stands for No Redundancy. Note that this algorithm also finds the stream rates flowing through the links and the delivered rates at the clients as *find_opt_rates_I*, when there is no restriction on the number of transcoders. Using the stream rates as determined by *find_opt_rates_NR*, function *find_min_transcoders* finds the optimal placement of transcoders and the rate conversions they perform. Figure 5.1 depicts the relationship between the algorithms and function *find_min_transcoders* and presents an overview of the contributions made in this chapter.

We prove that *find_min_transcoders* in fact finds $\mathcal{O}$, the minimal set of transcoders required to provide the best delivered rates to clients. We also demonstrate optimality of $\mathcal{O}$ through experimental results.

5.2 Observations related to redundancy in transcoders

In this section, we explore the properties related to placement of transcoders in the network nodes to develop rules for eliminating redundancy in transcoder placement. We modify *find_opt_rates_I* to include these redundancy rules. The resulting algorithm is called *find_opt_rates_NR*.

5.2.1 Redundancy in strings: Redundancy rule 1

We consider paths from the source to clients and identify instances where placing a transcoder at a node is redundant. We define a *string* as a set of nodes connected to each other such that each node in the string has only one outgoing link. Note that a string can occur in the following sequences:

1. starting at the source, ending at a client.
2. starting at the source, ending at any relay node.
3. starting at any relay node, ending at a client.
4. starting at any relay node, ending at any relay node.

All occurrences of strings in a multicast tree are depicted in Figure 5.2.

Lemma 5.1:

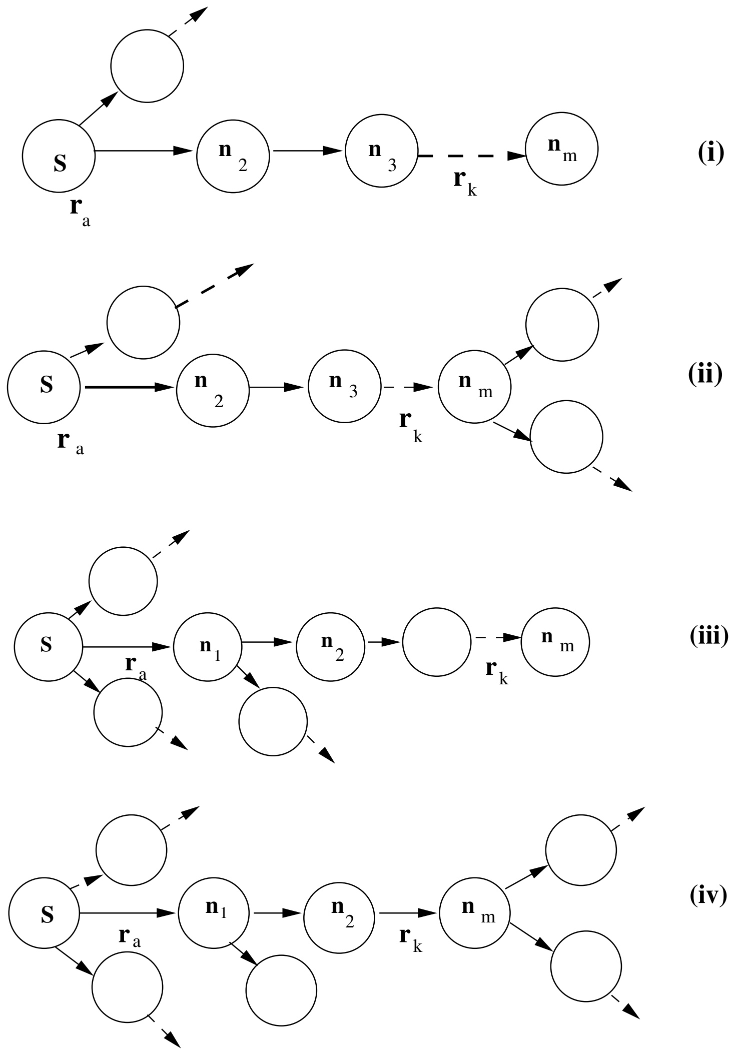
Consider various instances of a string $n_1, n_2, \dots n_m$ as shown in Figure 5.2. Let r_a be the stream rate flowing into n_1. Let r_k be the rate flowing into n_m. If $r_k < r_a$, i.e., transcoders are needed in this string, it is sufficient to place a transcoder at n_1.

Proof:

Given a string $n_1, n_2, \dots n_m$. With reference to any instance of a string as shown in Figure 5.2, we consider the following:

- Without loss of generality, let n_i, n_j, n_k be some intermediate nodes transcoding the stream from r_a to r_i to r_j to r_k where $r_a > r_i > r_j > r_k$.
- Let n_k be the last node with transcoding capability, i.e.,nodes n_{k+1} to n_m do not have transcoders. n_m receives the stream encoded at rate r_k.
    1. None of the clients can receive rate r_j flowing from n_j to n_k without loss, as r_k is the maximum encoded rate required in this subtree. So, n_j can transcode the stream at the rate r_k and send it to n_k.
    2. $n_1, n_2, \dots n_m$ is a string; hence no other branches exist.
    3. Similarly, none of the clients can receive the rate r_i without loss, which is greater than r_j and r_k.
- Extending the same logic, instead of encoding a stream with higher rate from n_1 to n_i, n_1 can transcode the stream to r_k.

Thus, only one transcoder is needed at the first node of a string. $\square$



(i) S is first node in the string; n_1 is the same as S; n_m is the client

(i) S is first node in the string; n_1 is the same as S; n_m is root of a subtree

(iii) n_1 is the first node in the string; n_m is the client

(iv) n_1 is first node in the string; n_m is the root of a subtree

→	Directly connected
⇢	Connected to other nodes

Figure 5.2: Strings in a multicast tree

Corollary 5.1: Redundancy rule 1

In a multicast tree, placing transcoders only at relay nodes having multiple subtrees, $\mathcal{R}_i$s, yields the same delivered rates at clients as placing transcoders at all relay nodes.

Proof:

This proof directly follows from observation 5.1. $\Box$

When there is no constraint on the number of transcoders, we apply Redundancy rule 1 and select all the relay nodes which have multiple outgoing links for placing transcoders. Our next step is to identify any redundant transcoders placed at these relay nodes.

5.2.2 Redundancy in trees: Redundancy rule 2

In this section, we present an observation that detects redundant transcoder at the root of a subtree. We first present a lemma which is used in the observation.

Lemma 5.2:

Let R_k be a node with m outgoing links $l_1, l_2, \ldots, l_m$. Let r_a be the stream rate flowing in to n_i. Suppose $r_1, r_2, \ldots, r_m$ are the maximum stream rates through $l_1, l_2, \ldots, l_m$ as calculated by Theorem 4.2 as shown in Figure 5.3. If $r_1, r_2, \ldots, r_m$ are greater than or equal to r_a, there is no need to transcode the stream at R_k.

Proof:

With reference to Figure 5.3, consider nodes R_k and R_1. A stream encoded at r_a can be delivered without loss from a node R_k to R_1 under the following conditions: (i) when the maximum stream rate through l_1 is greater than or equal to r_a or (ii) when a transcoder at R_k transcodes the stream to the maximum stream rate that can flow through l_1. Since R_k has multiple outgoing links, the above conditions have to be applied to each of these links. Since $r_1, r_2, \ldots, r_m$ are all greater than or equal to r_i, the stream can be delivered without loss across $l_1, l_2, \ldots, l_m$. Hence there is no need to transcode the stream at R_k.$\Box$

Corollary 5.2:

Let Λ be a subtree rooted at a relay node R_k. r_a is the stream rate flowing into R_k as shown in Figure 5.3. Suppose relay nodes $R_1, R_2, \ldots\ R_m$ connected to R_k have transcoders. If maximum stream rates that can be supported without loss by each outgoing link from R_k are $r_1, r_2, \ldots\ r_m$, such that each $r_i >= r_a$, it is redundant to have a transcoder at R_k.

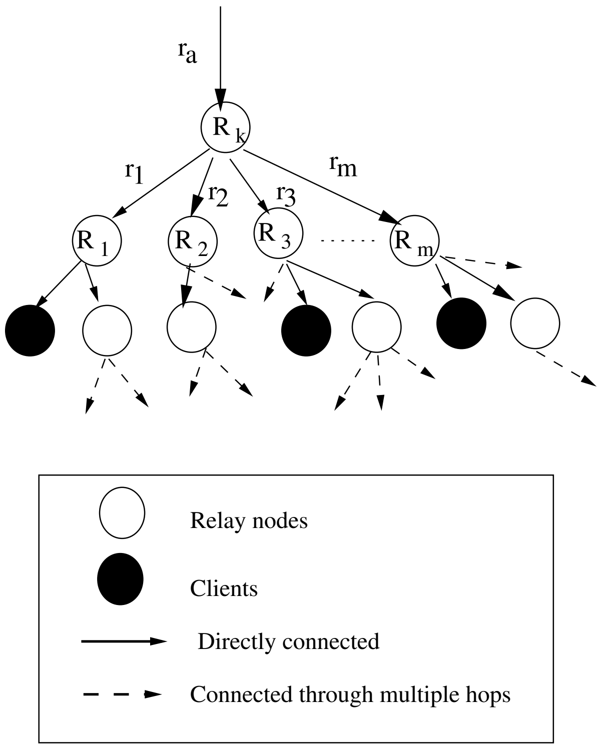


Figure 5.3: Illustration of a redundant node

Proof:

By Lemma 5.2, to send the stream to $R_1, R_2, \ldots, R_m$ without loss, there is no need for a transcoder at R_k. However, $R_1, R_2, \ldots, R_m$ are roots of subtrees, each serving multiple clients. In order to serve each client without loss, Lemma 5.2 has to be applied to each of $R_1, R_2, \ldots, R_m$. But, since $R_1, R_2, \ldots, R_m$ have transcoders, these can be enabled to serve clients without loss. Hence there is no need for a transcoder at R_k. $\Box$

Redundancy due to equi-deliverable rate clients

In a multicast tree with specific link bandwidths and client requirements, even though clients may have different delay tolerance values and different weakest link bandwidths, they may still have same maximum deliverable rates. We define clients having the same maximum deliverable rates as *equi-deliverable rate clients*.

Identifying equi-deliverable rate clients

We find the maximum deliverable rates at clients using Theorem 4.2. When all the clients in a subtree have the same maximum deliverable rate, γ_{eq}^{max}, we identify these as the equi-deliverable rate clients. Note that the maximum deliverable rate provides the upper bound on the delivered rates at these clients.

Determining delivered rate r_{eq} at the equi-deliverable rate clients

Let clients $C_1, C_2, \ldots C_n$ be the equi-deliverable rate clients having delivered rates $r_1, r_2, \ldots r_n$. The following two cases have to be considered to find the delivered rates at $C_1, C_2, \ldots C_n$:

1. *When none of the shared links is the weakest link on any client's path*: In this case, since the shared links do not have any impact on the delivered rates at the clients, for each client C_i, its delivered rate = its maximum deliverable rate. Hence, $r_1, r_2, \ldots r_n$ = γ_{eq}^{max}.

2. *When one of the shared links is the weakest link in the path of one or more clients*: Let a shared link l_k having bandwidth b_k be the weakest link in one of the clients, say C_j's path. By Theorem 4.1, the maximum stream rate that can flow through $l_k = b_k * min(\delta_i)$, where δ_is are the delay tolerance values of clients sharing this link. Note that since l_k is not their weakest link, all other clients sharing l_k, must have weakest links in their paths having bandwidth $< b_k$. The only way these clients can have the same maximum deliverable rate as C_j is by having a higher delay tolerance value than C_j. Thus, the delivered rates at $C_1, C_2, \ldots C_n$ = maximum rate that can flow through the shared link l_k = maximum deliverable rate at C_j = γ_{eq}^{max}.

From the above discussion it is clear that when we find an instance of equi-deliverable rate clients, all these clients have the same delivered rate equal to their maximum deliverable rate. i.e.,

$$r_{eq} = \gamma_{eq}^{max} \tag{5.1}$$

This important insight is used to eliminate redundant placement of transcoders as discussed below.

Lemma 5.3:

Consider a subtree Λ rooted at a relay node R_k as shown in Figure 5.4 having clients $C_1, C_2, \ldots C_n$ having maximum deliverable rates $r_1, r_2, \ldots r_n$, (as calculated by Theorem 4.2), where $r_1 = r_2 = \ldots = r_n = r_{eq}$. The incoming stream rate $r_a > r_{eq}$. In order to provide the optimal delivered rates to $C_1, C_2, \ldots C_n$, it is sufficient to place the only transcoder in Λ at R_k.

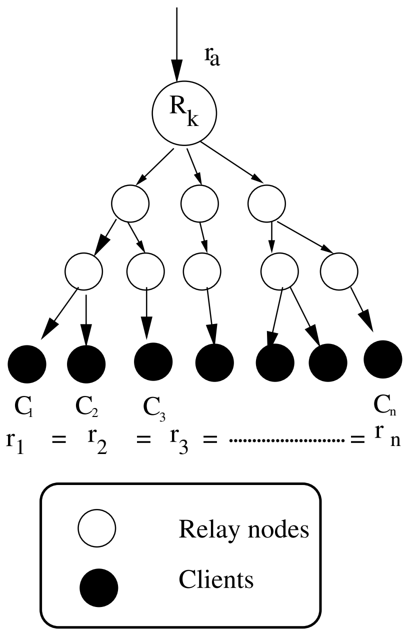


Figure 5.4: Equi-deliverable rate clients

Proof:
According to Equation 5.1, the delivered rates at $C_1, C_2, \dots C_n$ is also r_{eq}. Since r_{eq} is less than the incoming stream rate r_a, transcoding is required. However, since all clients have the same delivered rates, in order to serve the clients with minimum number of transcoders, it is sufficient to place a transcoder at R_k. □

Let us suppose that all relay nodes are capable of transcoding. With reference to Figure 5.3, according to Corollary 5.2, when all outgoing links from R_k are capable of supporting r_a, no transcoder is required at R_k. Thus, multiple transcoders in relay nodes $R_1, R_2, \dots R_m$ are enabled based on the delivered rates at the clients. However, this rule does not apply, when all clients are equi-deliverable rate clients as explained below.

With reference to Figure 5.4, consider the subtree Λ rooted at R_k. When the incoming stream rate $r_a > r_{eq}$ we consider the following:

- Let the maximum stream rates that can be supported without loss by each outgoing link from R_k, $r_1, r_2, \dots r_m$ are such that each $r_i >= r_a$. By Corollary 5.2, there is no need to transcode the stream at R_k as all the outgoing links from it are capable of supporting r_a.
- Since all the clients attached to R_k need r_{eq} which is less than r_a, transcoders are required at nodes in Λ to serve the clients with r_{eq}. This means that, by Corollary 5.2, multiple transcoders are required to serve the clients with the same rate.

- However, by placing a transcoder at R_k to transcode the stream to r_{eq}, all clients can be serviced with just one transcoder.

When $C_1, C_2, \ldots C_n$ in Λ rooted at R_k have the same maximum deliverable rate, the only transcoder enabled is at the root R_k, even when all the outgoing links from R_k are capable of supporting r_a.
Thus, in order to find the optimal number of transcoders, based on the maximum deliverable rates at the clients, appropriate lemma is applied in Redundancy rule 2.

5.3 Determining optimal transcoder placement

As discussed, when there is no constraint on the number of transcoders, algorithm *find_opt_rates_I* finds the best delivered rates at the clients. However, it may use more than the minimum number of transcoders required for providing best delivered rates to clients. For the algorithm to find the optimal number of transcoders, we have to ensure that all redundancy rules are applied so that only the minimal set of transcoders required for providing best delivered rates are enabled. This algorithm which enables the optimal transcoder placement for best delivered rates at the clients is called algorithm *find_opt_rates_NR*. We modify Pass 2 of algorithm *find_opt_rates_I* to assign appropriate stream rates to the links such that an optimal set of transcoders are enabled. We present the algorithm with appropriate comments to show the application of the redundancy rules in Figure 5.5.

Note that when there is only one outgoing link from n_i, Algorithm *find_opt_rates_NR* assigns the rate of the stream flowing through that link to l_k. Hence no transcoder is enabled at n_i. Thus, the algorithm ensures application of Redundancy rule 1. We provide an example to illustrate how algorithm *find_opt_rates_NR* ensures that Redundancy rule 2 is applied.

- Consider the network presented in Figure 5.6 (i). Stream rates assigned by algorithm *find_opt_rates_I* are indicated in the figure. Note that transcoders at both nodes R_1 and R_2 are enabled. In this case, if link l_1 has enough bandwidth to support 512 kbps, this *should* be assigned as the stream rate flowing through l_1 to avoid redundant enabling of a transcoder at R_1.

```
function find_opt_rate_NR(transinfo);
Initialize stream_rate [numlinks]; Initialize optim_rates [numclients];
for each link L %Starting from the leaf nodes of the tree
    if link is not shared
        max_stream_rate  =   find_max_deliverable_rate (Ci)
    else
         .
         .
  % same as for algorithm find_opt_rates_I %
    end
    stream_rate [L]  =  min (max_rate_thru_link, max_stream_rate);
    % Apply Redundancy rule 1  eliminates redundancy in strings %
end
for each client C
    for each link L in  C's path
        if link  L serves equi-deliverable rates clients
        % Case 1 of Redundancy rule 2  Corollary 5.2 %
           if link L is first link in client's path
              stream_rate [L]  =  delivered rate at client;
           else
              stream_rate [L]  =
              min(stream rate of prev. link, delivered rate at client)
           end
       else % Case 2 of Redundancy rule 2  Lemma 5.3 %
           max_deli_rate  =  stream_rate [L1]
           % Traverse path from source to client and
           % assign stream rate of first link to max_deli_rate
           if stream_rate [L] is greater than max_deli_rate
              actual_rate [L]  =  max_deli_rate;
           else
              actual_rate [L]  =  stream_rate [L];
              max_deli_rate =  stream_rate [L];
           end
        end
     end
     optim_rates [C]  =  actual_rate [last link in C's path]
end
```

Figure 5.5: Algorithm: *find_opt_rates_NR*

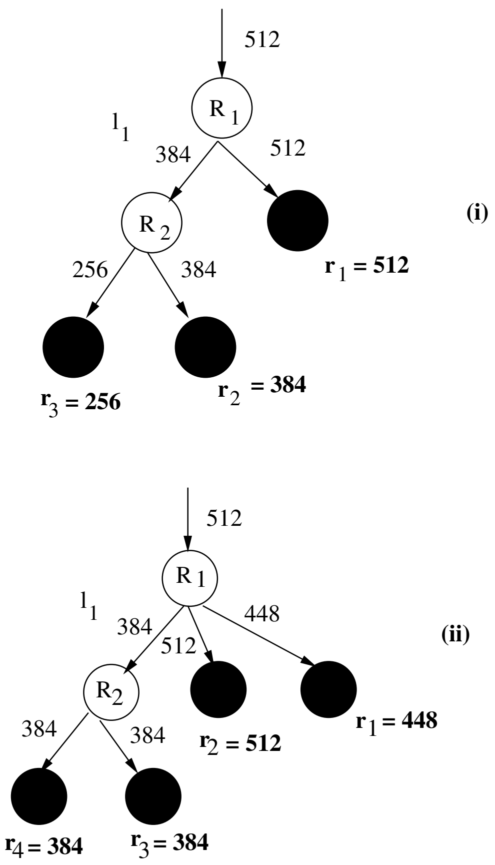


Figure 5.6: Example to illustrate application of redundancy rule 2

- Let us consider a similar network presented in Figure 5.6 (ii). Stream rates assigned by algorithm *find_opt_rates_I* are indicated in the figure. Note that only one transcoder at R_1 is enabled. In this case, even if link l_1 has enough bandwidth to support 512 kbps, this *should not* be assigned as the stream rate flowing through l_1 to avoid redundant enabling of a transcoder at R_2, as transcoder at R_1 can be used to transcode the stream to 384 kbps required to serve both the clients.

5.3.1 Finding the optimal set: *find_min_transcoders*

Algorithm *find_opt_rates_NR* computes the best delivered rates using the minimum number of transcoders. The algorithm finds the stream rates flowing through the links and the delivered rates at the clients. Traversing the links top to bottom in each client's path, when the incoming stream rate to a node is greater than the outgoing stream rate from the node, we can infer that the transcoder at that node is enabled. Function *find_min_transcoders* takes the stream rates as determined by *find_opt_rates_NR* as input and determines the minimum set of transcoders enabled to provide the best delivered rates to clients.

Thus, to find the optimal transcoder placement and the rate conversions the transcoders perform, we use the following two steps: (i) run *find_opt_rates_NR* with the AT transcoder placement option. This gives the stream rates flowing through the links as an output. (ii) Using these stream rates, run *find_min_transcoders* to find the minimum set of transcoders enabled and the rate conversions they perform.

5.3.2 Complexity of *find_opt_rates_NR*

The first pass of *find_opt_rates_NR* is the same as that of *find_opt_rates_I*. Hence, the number of computations is of the order of $O(DC)$, where D is the number of levels and C is the number of clients in the multicast tree, as derived in Section 4.4.5, in Chapter 4.

In the second pass, as in *find_opt_rates_I*, each link in a client's path is traversed. As in the case of *find_opt_rates_I*, the number of computations is O(L). Note that in *find_opt_rates_NR* there is a conditional check to verify whether the link serves equi-deliverable rates clients; thus, this algorithm requires less number of computations in this step than *find_opt_rates_I*, when there are equi-deliverable rates clients.

Algorithm *find_opt_rates_NR* requires O(DC) computations.

We have the following claim with respect to the optimal placement of transcoders discussed thus far:
The resulting set of transcoders after eliminating redundancy in placement, known as the *optimal set*, denoted by $\mathcal{O}$, is the minimal set that is required for providing the best delivered rates to clients (the redundancy rules identify *all* instances of redundant transcoders). We prove this claim below.

5.3.3 Optimality of $\mathcal{O}$

In order to show that $\mathcal{O}$ is the minimal set to provide best delivered rates to clients, we need to prove that all occurrences of redundant transcoders are identified and eliminated while selecting the nodes in $\mathcal{O}$.

Claim 5.1: $\mathcal{O}$ is the minimal set of transcoders required to provide best delivered rates to clients.
To prove:

1. $\mathcal{O}$ provides the best delivered rates to clients.

2. removing one or more transcoders from $\mathcal{O}$ and placing an equal number at nodes not in $\mathcal{O}$ does not improve the delivered rate for any client, i.e., even if $\mathcal{O}$ is not unique, any other combination of nodes would not provide higher delivered rates at the clients.

3. placing transcoders at nodes not in $\mathcal{O}$ in addition to $\mathcal{O}$ does not improve the delivered rate at any client.

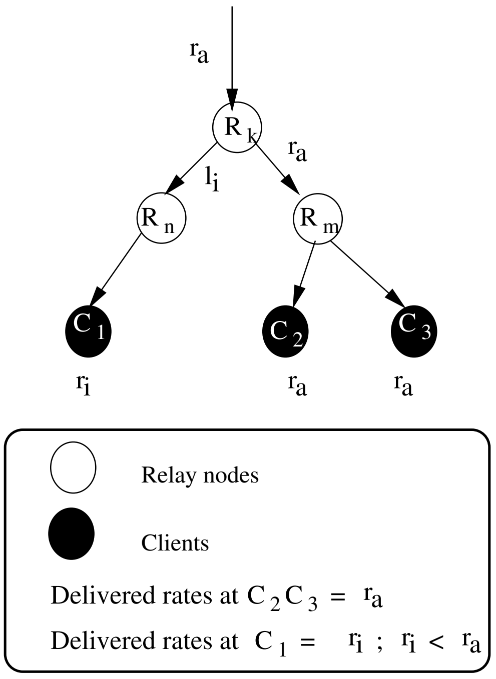
4. removing any transcoder from $\mathcal{O}$ results in reduction of the delivered rate for at least one client.

Proof:

1. When all the relay nodes are capable of transcoding, the rates delivered at the clients are the best delivered rates. Thus, to prove that $\mathcal{O}$ provides the best delivered rates to clients, we need to prove: delivered rates with transcoders placed at nodes in $\mathcal{O}$ are the same as delivered rates with AT option. We state this as: $\forall C_i, i = 1, 2, \ldots, m, r_i^{\mathcal{O}} = r_i^{AT}$.
   Under the AT option, all relay nodes have transcoders. For a given network bandwidths and client requirements, the delivered rates at the clients with this option can not be improved. As proven by the optimality claim in Section 4.4.1, delivered rates at clients using the AT option are the best delivered rates.

   Nodes in $\mathcal{O}$ are selected from the relay nodes by applying the redundancy rules. We have already proven that the two redundancy rules do not reduce the delivered rates for any clients. Hence, $\forall i, i = 1, 2, \ldots m, r_i^{\mathcal{O}} = r_i^{AT}$.

2. Consider a subtree Λ rooted at R_k with incoming stream rate r_a as shown in Figure 5.7. Delivered rates at the clients in this subtree are as indicated in the figure. Algorithm *find_opt_rates_NR* would choose node R_k for transcoder placement by Redundancy rule 1. i.e., $R_k \in \mathcal{O}$. However, we can remove the transcoder from R_k and place it at R_n, when link l_i has enough bandwidth to support r_a without loss. Note that the delivered rates at the clients do not change. This demonstrates that **the optimal set $\mathcal{O}$ need not be unique**. However, irrespective of the optimal set used for placement, the delivered rates at the clients do not change. Thus, more than one node may qualify for placing a transcoder in a string; however, delivered rates at the clients do not change with the



r_a
R_k
r_a
l_i
R_n
R_m
C_1
C_2
C_3
r_i
r_a
r_a
Relay nodes
Clients
Delivered rates at $C_2 C_3 = r_a$
Delivered rates at $C_1 = r_i$; $r_i < r_a$

Figure 5.7: Illustration of non-uniqueness of the optimal set

choice. Other options to move the transcoder from node k to node $l \notin \mathcal{O}$ include: (a) moving to a subtree root which has multiple outgoing links that support the incoming rate, with appropriate nodes in its subtrees having transcoders and (b) moving to a node in a subtree which has equi-deliverable rate clients. Note that both these cases are shown to be redundant by Corollary 5.2 and Lemma 5.3. Thus, the delivered rates can not be improved by the new placement of transcoders.

3. Suppose a transcoder is placed at node $q \notin \mathcal{O}$, in addition to the nodes in $\mathcal{O}$. Any additional transcoder can be placed only at a node which is proven to be redundant as per Lemma 5.1, Corollary 5.2, or Lemma 5.3. Thus, placing additional transcoders does not improve the rate for any client.

4. Suppose we remove a transcoder from a node k in subtree Λ_m. Since k is chosen for placement by applying Lemma 5.1, Corollary 5.2, or Lemma 5.3, removal of the transcoder would reduce the delivered rates at one or more clients.

Thus, $\mathcal{O}$ serves the best delivered rates to clients. □

Property of optimal set $\mathcal{O}$

Optimality of $\mathcal{O}$ as proven by Claim 5.1, can be stated mathematically in the following two ways:

1. The average delivered rates across all clients is a monotonically increasing function of the number of deployed transcoders from $\mathcal{O}$, when the transcoders are enabled one by one starting with the source stated as:

   - $f(q) >= f(q-1)$, where $f(q)$ finds the average delivered rates across clients for a given number of transcoders q from $\mathcal{O}$.
   - $f(q)$ is constant when $q > |\mathcal{O}|$ and all nodes in $\mathcal{O}$ are already selected for transcoder placement.

2. Let the number of transcoders in $\mathcal{O}$ be J. Suppose initially there are K relay nodes in the network which are enabled with transcoders, where $K > J$. There are $(K-J)$ nodes that are not in $\mathcal{O}$.

   When transcoders at the nodes that are not in $\mathcal{O}$ are disabled, the average delivered rate remains constant.

   With transcoders at all nodes in $\mathcal{O}$, when transcoders are disabled one by one starting at the leaf nodes of the tree, the average delivered rates across all clients is a monotonically decreasing function of the number of removed transcoders from $\mathcal{O}$. These are stated as:

   - $f(q)$ is constant when $q > |\mathcal{O}|$ and transcoders at $(K-J)$ nodes that are not in $\mathcal{O}$ are disabled.
   - $f(q-1) <= f(q)$, where $f(q)$ finds the average delivered rates across clients for a given number of transcoders q from $\mathcal{O}$.

The above statements capture the optimality claim as stated in Claim 5.1. These two properties can be used to prove the optimality of $\mathcal{O}$.

Experimental validation of optimality of $\mathcal{O}$

In this section, we demonstrate the optimality of $\mathcal{O}$ through experiments. We have implemented algorithm *find_opt_rates_NR* using **Matlab** and run several experiments to ensure that the algorithm always determines the optimal set of transcoders to provide best delivered rates to clients.

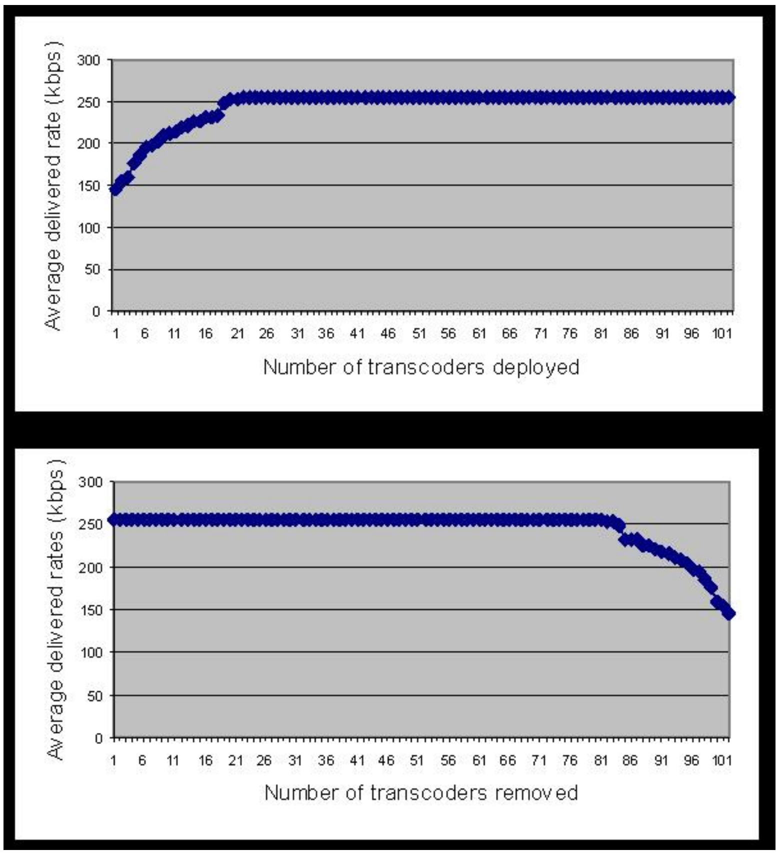


Figure 5.8: Validation of optimality of $\mathcal{O}$

We consider 100 randomly generated topologies of various sizes. We present one example in Figure 5.8. The network has 200 nodes and 10 levels. There are 98 clients. Hence there are $200 - (98 + 1) = 101$ eligible relay nodes for transcoder placement. Using *find_opt_rates_NR* we find that there are 21 nodes in the optimal set $\mathcal{O}$.

1. We first start with one transcoder at $\mathcal{S}$ and find the average delivered rate across all clients. This value is 149.2 kbps. We then place transcoders one by one at the nodes in the optimal set in the top-down order. Figure 5.8(a) plots the average delivered rates when transcoders are added to each node in $\mathcal{O}$ incrementally. When all the nodes in $\mathcal{O}$ are selected, the average rate is maximum at 254.3 kbps. When transcoders are added to other nodes, there is no change in the average rate as portrayed in the figure.

2. With all relay nodes enabled with transcoders, we start removing transcoders one by one from the nodes that are not in $\mathcal{O}$. Figure 5.8(b) plots the average delivered rates when transcoders are removed one by one. Note that the average rate remains constant at the maximum 254.3 kbps when the transcoders from nodes that do not belong to $\mathcal{O}$

are removed one by one. This trend continues till the point when all the transcoders from relay nodes which are not in $\mathcal{O}$ are removed. At this point, we remove transcoders from each node in the optimal set starting with the node in the bottom-up order. The average delivered rate decreases in a manner similar to the case when transcoders are incrementally added, as each transcoder from $\mathcal{O}$ is removed. When the last transcoder is removed, the average rate converges to the minimum average rate 149.2 kbps, the average delivered rate when transcoding is done only by $\mathcal{S}$.

5.4 Conclusions

A Content Service Provider (CSP) catering to the needs of clients dispersed around the world on a subscription based model, needs to understand the resources required to provide quality of service to the clients and appropriate placement of these resources [9] to maximize the delivered rates at the clients. The first question that needs to be answered is: how many transcoders are required if all clients are to be provided with optimal rates, assuming no constraints on the number of transcoders? Answer to this question would give the upper bound on the number of transcoders and their placement for providing the best delivered rates to clients. In this chapter, we presented an algorithm to find the optimal set $\mathcal{O}$ of transcoders for providing clients with the best delivered rates.

Our algorithm for finding the optimal rates and optimal transcoder placement is applied for one multicast tree at a time. This algorithm can be run for each instance of the multicast tree to determine the nodes that require transcoders. Note that since nodes may overlap across different trees, a node may require multiple transcoders from different runs of the algorithm; or a node chosen for placement in one run may not require a transcoder in another run. As discussed in the final section of the previous chapter, by providing various classes of service at different premiums, a CSP catering to a set of subscribed clients can enhance its revenue. Our algorithm can be readily used by the CSP to understand the benefits of investing in resources, given the cost of the resource and the revenue model from the classes of service offered. While such an analysis from a business perspective is beyond the scope of this thesis, we can claim that the analysis and algorithms presented in this chapter can be readily used by a CSP for efficient dissemination of multimedia contents.

In the next chapter, we look at the problem of finding optimal placement for a given number of transcoders, q, where $q < |\mathcal{O}|$. While we know that the delivered rates at clients would be less than the best delivered rates, we want to answer the following question: how can we place the transcoders such that the maximum possible rates are delivered at the clients?

Chapter 6

Determining optimal placement for a given number of transcoders

(Scheduled streaming, Static link bandwidths)

6.1 Introduction

In this chapter we address the following question: Given a limited number q of transcoders, where can these be placed such that the delivered rates are maximized across all the clients? Recall that the **Eligible set** $\mathcal{E}$ contains all the relay nodes in the multicast tree and **Optimal set** $\mathcal{O}$ is the smallest set of nodes where transcoders have to be placed to obtain the best delivered rates for all clients.

We define the following additional terms for the ease of understanding the solutions presented in this section:

Super node: A node other than $\mathcal{S}$, which is the common ancestor of two or more nodes in $\mathcal{O}$ and does not have a transcoder.

Useful set: The union of $\mathcal{O}$ and the set of super nodes. This set is denoted by $\mathcal{U}$.

We first formulate the placement for q transcoders as an optimization function. We also develop an algorithm, *find_opt_placement_E*, that finds the best combination of nodes for transcoder placement that delivers the maximum delivered rates to clients, considering all possible combinations of nodes from $\mathcal{E}$. We next show that by limiting the set of nodes considered for placement to the useful set $\mathcal{U}$, the complexity can be reduced by several orders of magnitude. We

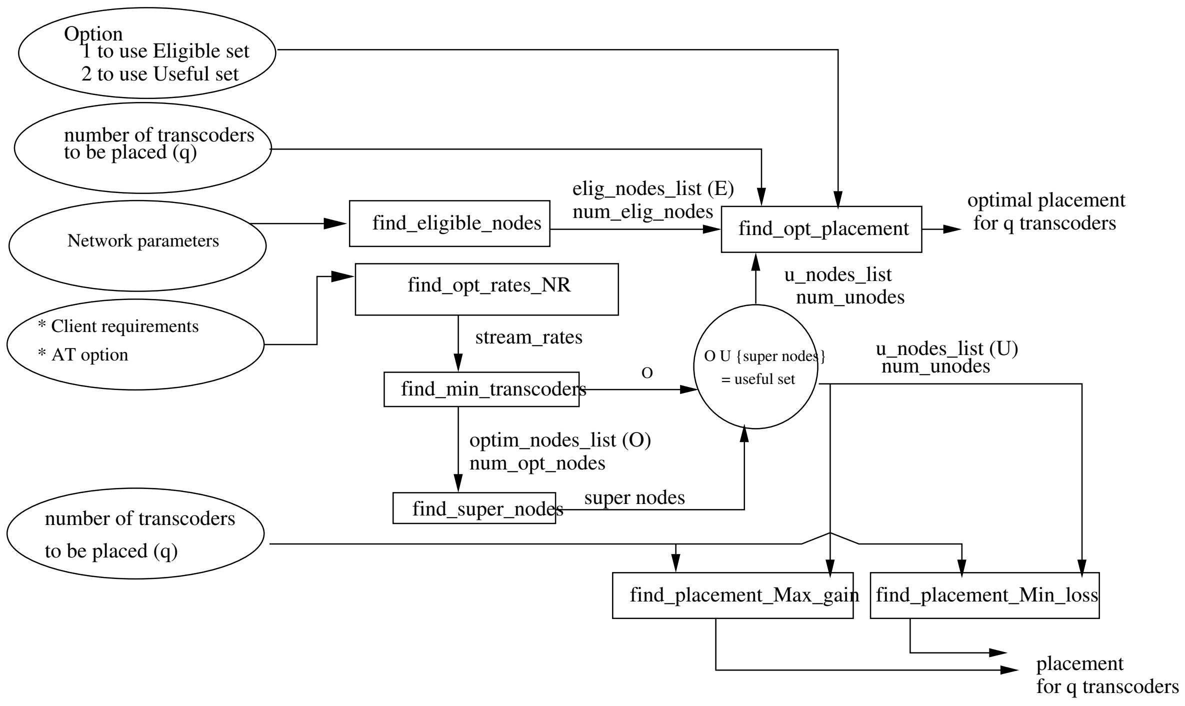


Figure 6.1: Overview of solution approaches: Chapter 6

prove that the resulting algorithm *find_opt_placement_U* is equivalent to *find_opt_placement_E*.

find_opt_placement_U narrows the search space thereby making the algorithm converge faster; however, the computation time required by this optimal algorithm may render it infeasible in practice. Hence we develop two greedy alternatives: *max-gain* and *min-loss*. Both the algorithms select nodes from $\mathcal{U}$. The max-gain algorithm starts with no transcoders (other than $\mathcal{S}$). It adds q transcoders choosing one node from $\mathcal{U}$ at a time, such that at each step, the improvement in delivered rates at clients from the added transcoders is maximum.

The min-loss algorithm starts with transcoders at all nodes in $\mathcal{U}$ and removes one transcoder at each step until q transcoders are left. The node chosen at a step is such that it has the least impact on the delivered rates at the clients. The algorithms are presented in detail in the next sections. Figure 6.1 provides an overview of the contributions of this chapter.

6.2 Optimization-based approach

We formulate our objective of maximizing delivered rates across all clients in the network for a given number q of transcoders as an optimization problem, where paths from source to every client in the network are considered. Note that we can use the same formulation presented in Section 4.2 in Chapter 4, with the following difference: In this case, instead of the **Transcoder**

Objective function: Minimize $\forall k, k = 1, 2, \ldots, m, \sum_k (\Gamma - r_k)^2$,

where m is the number of clients.

Constraints:

Rate constraint: $\forall i, i = 1, 2, \ldots\ n$, $r_i <= \Gamma$, where n is the number of links in the multicast tree.

Delay tolerance constraint: $\forall k, k = 1, 2, \ldots\ m$, $\mathcal{L}_k <= \delta_k$, where $\mathcal{L}_k$ is the latency incurred in the path from $\mathcal{S}$ to C_k, due to buffering and transcoding.

Number of transcoder constraint: $|k$: node k has an outgoing link with stream rate smaller than rate flowing into it $| <= q$, where q is the number of transcoders to be placed.

Figure 6.2: Optimization function

constraint which provides the information on whether a given node is transcoder capable or not, we use the **Number of transcoder constraint** that ensures that only the specified number of transcoders are deployed. The optimization function formulated using the *fmincon* function from the optimization toolbox of **Matlab** is presented in Figure 6.2.

Design variables: Stream Rates r_is flowing through links l_is
Objective function: Maximize delivered rates at the clients, written as: Minimize $\sum_k (\Gamma - r_k)^2$, where Γ is the base encoding rate and r_k is the rate delivered at client C_k.
Constraints:

- *Rate constraint*: This constraint ensures that the stream rate r_i flowing across any link l_i in the network is bounded by Γ which is the highest possible stream rate. We represent this constraint as:
  $r_i <= \Gamma$

- *Delay tolerance constraint*: This constraint ensures that the client specified delay tolerance δ_k is not exceeded while delivering enhanced rate to the client. It is specified as:
  $\mathcal{L}_k <= \delta_k$,
  where $\mathcal{L}_i$ is the latency incurred in the path from $\mathcal{S}$ to C_k, due to buffering and transcoding.

- *Number of transcoder constraint*: As explained before, when the stream rate flowing into a node is greater than the stream rate flowing out of it through any of its out going links, we can infer that the node has a transcoder. We want to constrain number of such nodes. We specify this constraint as:

$|k$: node k has an outgoing link with stream rate smaller than rate flowing into it $| <= q$, where q is the number of transcoders to be placed.

Results: We used the **fmincon** function from the optimization toolbox of **Matlab** for implementation. Note that the same objective function as in Section 4.2 is used, with the number of transcoder constraint replacing the transcoder constraint.

Given the eligible set $\mathcal{E}$. Given q transcoders, where $q < |\mathcal{O}|$, our objective is to seek a placement which delivers the best possible rates to clients. Number of possible unique placements is given by:

$$\begin{pmatrix} E \\ q \end{pmatrix} = \frac{E!}{q!(E-q)!} \tag{6.1}$$

We found the following two problems in computing the placement when any network having 15 or more nodes is given as input to the **fmincon** function: (i) due to the exponential search space, the algorithm takes a long time to converge or (ii) it converges to a local optimum; this is explainable, given the fact that *fmincon* may be trapped to local optima and may not be able to find a feasible solution when constraints are nonlinear [38]. Hence, we next explore ways to reduce the number of combinations to be considered in finding the optimal placement for a given number of transcoders.

6.3 Towards finding lower cost optimal algorithms

We first present an algorithm that finds the optimal placement of q transcoders, considering all possible subsets of q nodes of the eligible set $\mathcal{E}$. We refer to this algorithm as *find_opt_placement_E*, where E stands for the eligible set. As expected, while the algorithm converges in a reasonable amount of time for networks with small number of nodes, as the number of combinations increases exponentially, it is not a practical approach for networks with large number of nodes.

Given that the optimal set consists of the minimal set of nodes where transcoders have to be placed to provide best delivered rates to clients, we explore whether confining to nodes from the optimal set suffices in finding the optimal combination for a given number of transcoders. Let $\mathcal{O}$ be the optimal set which provides the best delivered rates to clients, as determined by *find_min_transcoders*. Suppose q transcoders are available for placement, where $q < |\mathcal{O}|$. We define *best possible rates for a given number of transcoders* as the rates that result in the minimal reduction in delivered rates at clients compared to the best delivered rates computed using *find_opt_rates_I*. Suppose $r_1, r_2, \ldots, r_m$ are the optimal delivered rates at the m clients computed using *find_opt_rates_I*. Suppose $v_1^1, v_2^1, \ldots, v_m^1$ are the delivered rates when q transcoders are placed in one way, $v_1^2, v_2^2, \ldots, v_m^2$ are the delivered rates when q transcoders are placed a second way, etc., our objective is to seek a placement which delivers best possible rates to clients, i.e., we want to select that placement i, such that $\Sigma(r_k - v_k^i)$ is minimum. We ask the following questions:

1. Is it sufficient to consider only the nodes in $\mathcal{O}$ for optimal placement of a given number of transcoders?
2. Under what conditions is it necessary to consider nodes that are not in $\mathcal{O}$?
3. When it is not sufficient to consider only the nodes in $\mathcal{O}$, how can we identify the additional nodes that must be considered?

We find answers to these questions in the sections that follow.

6.3.1 Insufficiency of considering only nodes in the optimal set

In this section, we explore the first question and prove that when the number of available transcoders for deployment is less than $|\mathcal{O}|$, it is *not* sufficient to consider only nodes in $\mathcal{O}$ to provide the best possible delivered rates at clients.

Lemma 6.1:
Consider the optimal set $\mathcal{O}$ as determined by algorithm *find_min_transcoders*. If the number of available transcoders for deployment is less than $|\mathcal{O}|$, then it is *not* sufficient to consider only nodes in $\mathcal{O}$ to provide the best possible delivered rates at clients.

Proof: By contradiction

With reference to Figure 6.3, let $\{R_1, R_2, \ldots R_k\} \in \mathcal{O}$ and $R_a \notin \mathcal{O}$. Let us assume that it is sufficient to consider the nodes in $\mathcal{O}$ for placing a given number of transcoders. Let $r_1, r_2, \ldots r_m$ be the best delivered rates at $C_1, C_2, \ldots C_m$. Let $r_{min} = min(r_1, r_2, ., r_m)$. Consider the case when each subtree rooted at $R_1, R_2, \ldots R_k$ has at least one client having delivered rate r_{min}. To

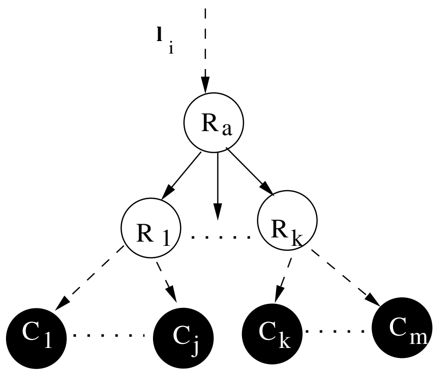


Figure 6.3: Redundancy in transcoders at relay nodes

place q transcoders where $q < |\mathcal{O}|$, a subset of $R_1, R_2, \ldots R_k$ has to be chosen. But, in order to serve all the clients without loss, r_{min} must flow through the shared link l_i when there are no transcoders at some nodes in $R_1, R_2, \ldots R_k$. Hence all clients get only r_{min}; Note that this is the rate the clients would get if there are no transcoders in any of the nodes in $R_1, R_2, \ldots R_k$. Thus, no improvement can be realized in the delivered rates at the clients by considering a subset of $R_1, R_2, \ldots R_k$. However, by placing a transcoder at R_a and placing a transcoder at one of the nodes $R_i, R_j, \ldots R_k$, rates better than r_{min} can be delivered to one or more clients served by the chosen node. Thus, it is *not sufficient* to consider only the nodes in $\mathcal{O}$ for placing transcoders when the number of available transcoders is less than $|\mathcal{O}|$. $\Box$

From Lemma 6.1, we can infer that the nodes which are considered redundant during the detrmination of $\mathcal{O}$ may become useful when a limited number of transcoders has to be placed such that the delivered rates to clients is maximized. Thus, we have answered the first question raised in the previous section: It is *not* sufficient to consider only the nodes in optimal set to find optimal placement for a limited number of transcoders.

In order to answer the next two questions, we introduce *super nodes* in the next section. We start with the properties of super nodes; this provides the insight that is useful to develop an algorithm to identify the super nodes.

6.3.2 Super nodes and their properties

As defined at the beginning of this chapter, any common ancestor (other than $\mathcal{S}$) to two or more nodes in the optimal set that does not have a transcoder is a super node. We will now take a closer look at super nodes and list their properties.

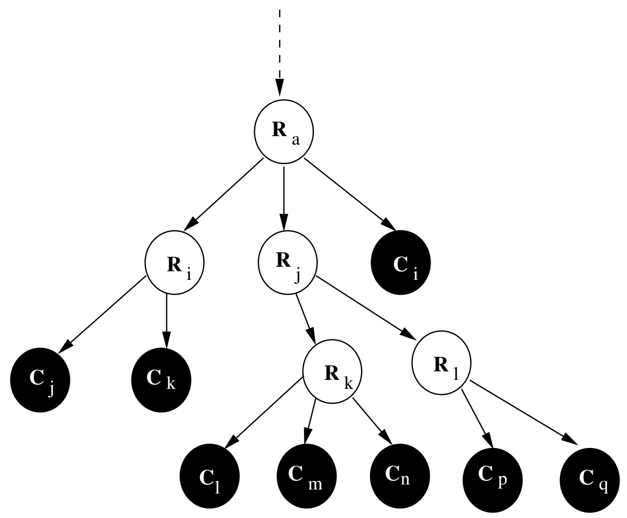


Figure 6.4: Identifying a super node

Consider Figure 6.4. Given this network and client requirements, we run *find_min_transcoders*. Nodes R_i, R_k, and $R_l \in \mathcal{O}$. Nodes R_a and R_j are the super nodes.

- By default, all nodes in the network have $\mathcal{S}$ as their common ancestor. However, $\mathcal{S}$ is not considered to be a super node.
- A super node is not in $\mathcal{O}$. In our example, Nodes R_a and R_j are not in $\mathcal{O}$.
- A super node can serve clients at various levels. In Figure 6.4, R_a serves client C_i (say, at level d), clients C_j and C_k (at level d+1) and clients C_l, C_m, C_n, C_p, and C_q (at level d+2).
- Enabling only a super node with transcoder – when there are zero or more transcoders at the subtree roots served by the super node – may not provide any improvement in the delivered rates of the clients. In the example in Figure 6.4, having a transcoder at R_a does not serve any purpose when R_i, R_k and R_l do not have transcoders. Thus, in this example if there is only one available transcoder, it does not provide any benefit to the CSP as no improvement in delivered rates at clients can be achieved by this transcoder.
- A super node gives rise to dependant set $\mathcal{D}$ when each subtree served by the super node has at least one client having a delivered rate $= min(r_i)$, where $r_i s$ are the delivered rates at the clients served by the super node. We define a dependant set as follows:
  $\mathcal{D}$ is a dependant set having nodes $R_i, R_k, \ldots\ R_l$ where $R_i, R_k, \ldots\ R_l \in \mathcal{O}$ such that no

improvement in the delivered rates at clients is achieved unless transcoders are placed at all $R_i s \in \mathcal{D}$.

Using the insights developed from the above properties of super nodes and the dependant set, we develop an algorithm to identify the super nodes. Then, we find the optimal placement for q transcoders by considering the nodes in $\mathcal{U}$ instead of considering all the nodes in $\mathcal{E}$. In the next section, we present an example to illustrate these concepts and motivate the reasoning underlying the algorithm for finding the super nodes.

6.3.3 Identifying super nodes: An illustrative example

We present two scenarios (same subtree with some link bandwidths changed) in Figure 6.5. The link numbers and actual stream rate that flows through each link are marked in the figure. Nodes 4, 5, and 6 have transcoders, i.e., 4,5,6 $\in \mathcal{O}$. Since each of these nodes serve at least one client having delivered rate 384 kbps and the stream flowing through link 1 is encoded at 384 kbps, there is no need for a transcoder at node 2 (as decided by *find_min_transcoders* which comes up with the optimal placement). Suppose we are given two transcoders; our objective is to place the transcoders at nodes to provide optimal rates at the clients. We need to find the actual stream rates that can flow through the links to provide loss-free delivered rates to clients.

1. Case (i): Considering the optimal set {4,5,6}, to place two transcoders, the following combinations are possible: {4,5},{4,6}, and {5,6}. Note that the chosen combination determines the maximum stream rate that can flow through the shared link 1 in order to serve all the clients without loss.

    - If {4,5} is chosen, the maximum stream rate that can flow through the shared link 1 is given by: min(delivered rates at clients 12, 13, 14 served by node 6 that does not have a transcoder) = 256 kbps.
      In this case, 256 kbps flows through links 1, 2, 3, and 4. With transcoder at node 4, client 8 gets 128 kbps; with transcoder at node 5, client 11 gets 192 kbps; all other clients get 256 kbps;
    - If {4,6} is chosen, the maximum stream rate that can flow through the shared link 1 is given by: min(delivered rates at clients 10, and 11 served by node 5 that does not have a transcoder) = 192 kbps.

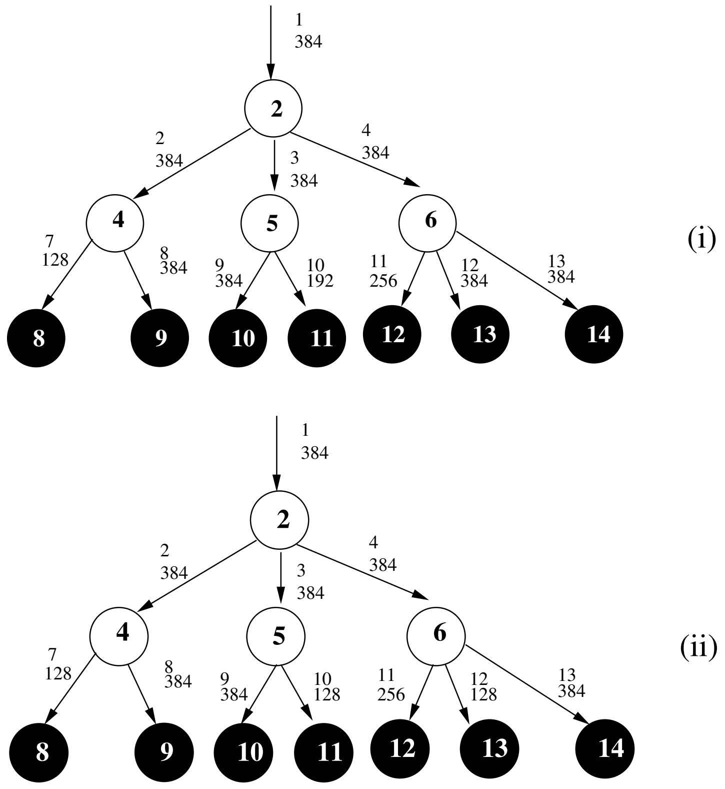


Figure 6.5: Example to illustrate the role of super nodes

In this case, 192 kbps flows through links 1, 2, 3, and 4. With transcoder at node 4, client 8 gets 128 kbps; transcoder at node 5 is not used as both clients 10 and 11 get 192 kbps; all other clients get 192 kbps;

- If {5,6} is chosen, the maximum stream rate that can flow through the shared link 1 is given by: min(delivered rates at clients 8, and 9 served by node 4 that does not have a transcoder) = 128 kbps.

  In this case, 128 kbps flows through links 1, 2, 3, and 4. In this case both the transcoders at 5 and 6 are not used as the delivered rates at clients is 128 kbps;

Thus, the best combination is {4,5} with the total delivered rates across clients being 1600 kbps. However, this is not the optimal combination that provides most improvement in delivered rates across all the clients, when all eligible nodes are considered for transcoder placement. Suppose node 2 is added to the optimal set; consider the combination {2,6}. In this case, 384 kbps flows through link 1 and 4. 128 kbps flows through link 2, 192 kbps through link 3. The delivered rates at clients 8 and 9 are 128 kbps; 10 and 11 are 192 kbps, client 12 gets 256 kbps and clients 13 and 14 get 384 kbps. Total delivered rates across clients is 1664 kbps.

Hence, by considering the parent node 2 along with $\mathcal{O}$, the optimal placement can be found.

2. Case (ii): Suppose the stream rates flowing through links 10 and 12 are 128 kbps. Since each of the nodes 4, 5, and 6 serve a client having a delivered rate of 128 kbps, choosing any two of these nodes does not improve the delivered rates for any client. Nodes $\{4,5,6\}$ form a dependant set. Here again, by including the parent node 2 in $\mathcal{U}$ and placing transcoders at nodes 2 and 6 optimal delivered rates at clients can be provided.

Node 2 is the **super node**. Note that when there are no super nodes in the network, $\mathcal{U} = \mathcal{O}$. In case (ii), $\mathcal{D} = \{4,5,6\}$; i.e., nodes 4, 5, and 6 belong to the dependant set, as no improvement in delivered rates is possible by placing transcoders in any subset of these nodes.

6.3.4 Algorithm to identify super nodes

Using the properties of super nodes, we develop the algorithm *find_super_nodes* as given in Figure 6.6 to identify the super nodes. Let $\mathcal{R}$ be the set of super nodes. These nodes are added to the optimal set. The resulting set of nodes is the **useful set**, $\mathcal{U}$.
$\mathcal{U} = \mathcal{O} \cup \mathcal{R}$.
Thus, we have answered the three questions raised at the beginning of this section.

- We have proved that it is not sufficient to consider only the nodes in $\mathcal{U}$ for optimal placement of a given number of transcoders.
- When there are instances of super nodes in the network, we need to include these nodes in the useful set to find optimal placement for q transcoders, where $q < |\mathcal{O}|$.
- We identify the super nodes using the algorithm *find_super_nodes* presented in Figure 6.6.

Based on the discussions thus far, we next develop algorithms that use the eligible set or the useful set, to find the placement for q transcoders. We analyze the savings in computational time in finding the optimal placement when we use the useful set instead of the eligible set and the property of $\mathcal{U}$ in the next section.

```
Input:
optim_nodes: Boolean vector that indicates nodes
that are in the optimal set
num_optim_nodes: number of nodes in the optimal set

% following global variable is used by the algorithm %
numnodes

Output:
super_node_list: Vector having node ids of the super nodes

Function find_super_nodes (num_optim_nodes, optim_nodes)
for n  =  1:num_optim_nodes % for every node in the optimal set %
    current_node  =  optim_node[n];
    [num_ancestors, ancestors]  =  find_ancestor(current_node);
     % find ancestors up to the source %
     for m  =  1: num_ancestors
         transcoder_present  =  is_transcoder(ancestors[m]);
          % check if ancestor has a transcoder %
         if transcoder_present  =  0
          % if ancestor does not have a transcoder,
            add it to the ancestor list %
            ancestor_list[n]  =  ancestors[m];
        end
    end
end
for each entry in ancestor_list
    if another entry for the same ancestor
       add the ancestor to super_node_list
    end
end
```

Figure 6.6: Algorithm: *find_super_nodes*

```
Input:
num_trans: constant indicating the number of transcoders available
num_unodes: constant indicating number of nodes in the useful set
u_nodes_list: vector of size numnodes with useful nodes indicated
num_enodes: constant indicating number of nodes in the eligible set
e_nodes_list: vector of size numnodes with eligible nodes indicated
option_considered: variable to indicate option to consider, E or U

% the following global variables are used by the algorithms %
numnodes, trans_info

Output:
trans_placement: vector indicating optimal placement for num_trans
transcoders
deli_rates: vector with delivered rates at the clients for the optimal
placement
n_combns: constant indicating number of combinations considered for the
option_considered
```

Figure 6.7: Variables used in *find_opt_placement*

6.4 Optimal algorithms

We have two options that find the optimal placement of q transcoders, $q < |\mathcal{O}|$. In the first one, given the eligible nodes in the network, all the combinations of q nodes from this eligible set are considered to find the combination that delivers the maximum rates at the clients. In the second option, only nodes from the useful set are considered. Algorithm *find_opt_placement* finds the optimal placement for q transcoders by using either the eligible set or the useful set based on the value of the input, *option_considered*.

We present the variables used in algorithm *find_opt_placement* in Figure 6.7. Outline of the algorithm *find_opt_placement* is presented in Figure 6.8. For the ease of understanding, we refer to the algorithm as *find_opt_placement_E* when the eligible set is considered and *find_opt_placement_U* when the useful set is considered. We present experiments to compare the performance of *find_opt_placement_E* and *find_opt_placement_U* in the next section.

```
With transcoder only at S, run find_opt_rates_I to
find the base_deli_rates at clients
if num_trans == 1
 % when only one transcoder is available %
    for each node in useful set
        place a transcoder and find the delivered rates at clients;
        find the node that maximizes the improvement in delivered rate
        compared to the base_deli_rates
    end
    return trans_placement, deli_rates
else
  % when multiple transcoders are available %
    if option_considered == 1
        max_num_combns = (factorial(E)/ factorial(n) factorial(E-n));
    else
        max_num_combns = (factorial(U)/ factorial(n) factorial(U-n));
    end
    for count = 1: max_num_combns
        for each combination of nodes
          find delivered rates at the clients
          for each client find the improvement compared to the
          base delivered rates
          choose combination that provides maximum improvement
          new_transinfo = chosen placement of transcoders
        end
        trans_placement = new_tarnsinfo;
        deli_rates = delivered rates with new_transinfo;
        num_combns = combinations considered;
    end
end
```

Figure 6.8: Algorithm *find_opt_placement*

6.4.1 Comparison of *find_opt_placement_E* and *find_opt_placement_U*

In this section, we present the experiments that compare the performance of the two optimal placement algorithms: *find_opt_placement_E* and *find_opt_placement_U*. Given that both algorithms find the optimal placement for a given number of transcoders, we use the *number of combinations considered* and *CPU time taken* as the performance parameters.

Set-up: In all the experiments presented in this chapter, the following parameters remain the same:
(i) Playout duration $\mathcal{T}$ is set to 1 hour and (ii) The minimum rate requirement of all clients is set to 128 kbps, and (iii) Without loss of generality, queuing delays and propagation delays are set to zero.

Experiments: We consider 40 random topologies having 15-40 nodes, with randomly chosen link bandwidths between 128–768 kbps and client δ requirements between 300-3600 seconds. Consider a specific value of transcoders to be placed q, where $q < |\mathcal{O}|$. We observed that *opt_placement_U* selects the same combination of nodes for transcoder placement as the optimal placement chosen by *opt_placement_E* for each run with the given q. We present a comparison of number of combinations considered by the two placement algorithms and CPU time for convergence of these algorithms.

Results: Figures 6.9 and 6.10 present the comparisons for topologies having 20-25 nodes and 35-40 nodes respectively. Topology index is plotted on the x-axis, number of combinations along the primary Y-axis, and CPU time along the secondary Y-axis. Comparison of the number of combinations considered is plotted as a bar chart while the CPU time taken for convergence of the algorithms is plotted as a line chart. As seen from these graphs, by using the useful set instead of the eligible set, the number of combinations considered, hence the CPU time taken by the algorithm, is significantly reduced.

6.5 Property of useful set $\mathcal{U}$

As demonstrated in the last chapter in Section 5.3.2, the average delivered rates across all clients is a monotonically increasing function of the number of deployed transcoders from $\mathcal{O}$, when the transcoders are enabled one by one top-down from the source. In this chapter, we refine this claim given the notion of super nodes.

Claim 6.1: It is sufficient to consider all possible combinations of nodes in the useful set $\mathcal{U}$, for

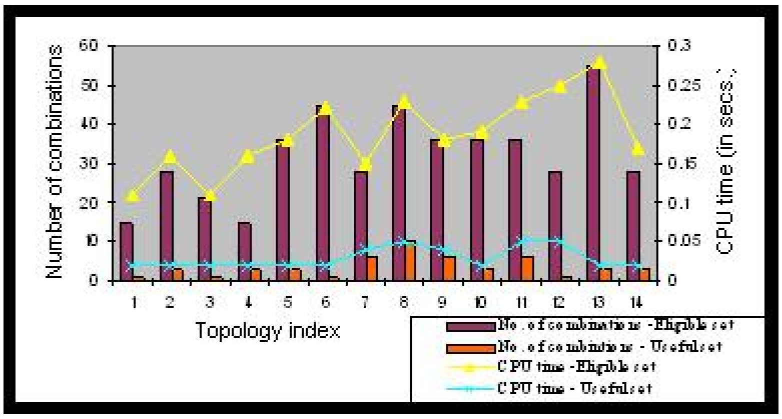


Figure 6.9: Comparison of optimal algorithms(20-25 nodes)

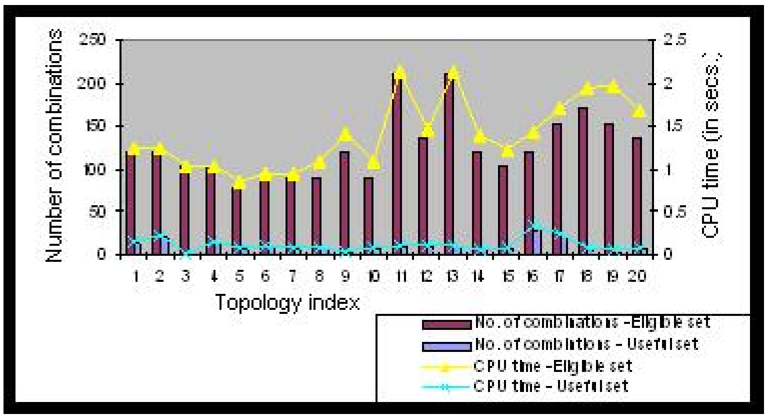


Figure 6.10: Comparison of optimal algorithms(35-40 nodes)

optimal placement of q transcoders, where $q < |\mathcal{O}|$.

To prove:

The union of super nodes and $\mathcal{O}$ contains all the nodes that can contribute to any improvement in the delivered rates of clients.

Proof:

Super nodes are a subset of nodes which are considered redundant by algorithm *find_min_transcoders* discussed in Chaper 5. We examine the redundancy rules (Refer to Section 5.2, in Chapter 5) to ensure that the super nodes list includes all nodes which may contribute to increase in the average delivered rates, when transcoders are incrementally placed.

- As proven in Lemma 5.1, adding any node from a string to the set of super nodes would not contribute to any improvement in the delivered rate of any client.

- When clients in a subtree are equi-deliverable rate clients, no improvement in delivered rates can be realized by enabling additional transcoders in that subtree, as proven by Lemma 5.3.

- Considering Corollary 5.2, the root node which is redundant when finding the optimal set of transcoders required to deliver the best delivered rates to clients becomes a candidate for placement of transcoder, when the transcoders are limited to $q < |\mathcal{O}|$.

Thus, the only nodes that need to be considered, in addition to the nodes in $\mathcal{O}$, are the super nodes, identified as redundant by Corollary 5.2. Hence, by considering all possible combinations from $\mathcal{U}$, optimal placement for q transcoders can be found. $\square$

As a corollary, it is clear that if there are no super nodes in the network, then it is sufficient to consider just the nodes in $\mathcal{O}$ for optimal placement of q transcoders, where $q < |\mathcal{O}|$.

Complexity of *opt_placement_U*

The number of unique combinations to be considered by *find_opt_placement_U* is given by:

$$\begin{pmatrix} U \\ q \end{pmatrix} = \frac{U!}{q!(U-q)!} \tag{6.2}$$

When the network is large and U, the number of nodes in $\mathcal{U}$ is large, since the complexity of algorithm *opt_placement_U* is exponential, it is necessary to develop other approaches of lower complexity but with effective placement decisions. We present two such algorithms in the next section.

6.6 Lower complexity alternatives: Greedy algorithms

The question addressed in this section can be stated as: Given a limited number of transcoders, where can each of these be inserted (among the useful nodes) in turn such that the gain – in terms of improved delivered rates across all clients – from each inserted transcoder is maximized? Algorithm *findplacement_Max_gain*, presented next, answers this question; Maximum gain refers to the modus operandi of the algorithm: it adds transcoders one at a time to the nodes in the useful set such that at each step, the improvement in delivered rates at clients is maximum.

The question above can also be posed as: Given transcoders at all the useful nodes, which of these transcoders can be removed (to use just the given number of transcoders) such that the loss – in terms of decreased delivered rates across all clients – from each removed transcoder is minimized? Algorithm *findplacement_Min_loss* answers this question; Minimum loss refers to removing transcoders one at a time, from the useful set that would have least impact on the delivered rates at the clients.

6.6.1 Max-gain algorithm

Let q be the number of transcoders to be placed. Given $\mathcal{U}$,

1. Start with no transcoders (other than $\mathcal{S}$) and find the delivered rates at clients.
2. Place first transcoder at each node in $\mathcal{U}$ one at a time and find the delivered rates at clients for each of these additions. Choose the option that maximizes the gain from the addition of a transcoder at the chosen location. Remove the node selected for placement from the useful node list. Reduce q by one.
3. While q is positive, with the chosen placement, repeat step 2 for the next transcoder placing it at each useful node. Note that for each round of selection, the delivered rates with the placement in the previous round would be used for comparison to find the additional gain.

Algorithm *findplacement_Max_gain* is given in Figure 6.11.

6.6.2 Min-loss algorithm

Steps in the algorithm are listed below:
Let q be the number of transcoders to be placed. Given $\mathcal{U}$,

1. Start with transcoders at all nodes in $\mathcal{U}$. Find the delivered rates at clients.
2. Remove transcoders one at a time and find the delivered rates at clients for each of these removals. Choose that transcoder that minimizes the loss from the removal of the transcoder at the chosen location. Remove the node from the useful node list. Reduce q by one.
3. While q is positive, with the chosen placement, repeat step 2 for the next transcoder, removing it from each remaining useful nodes. Note that for each round of selection, the delivered rates with the placement in the previous round would be used to find the cumulative minimal loss.

Algorithm *findplacement_Min_loss* is given in Figure 6.12.

6.6.3 Performance of greedy algorithms

Having established that *opt_placement_U* is equivalent to *opt_placement_E* with improved efficiency, we set *opt_placement_U* as our base optimal algorithm. Our objective is to understand the following:

- for a given network topology and client requirements, how do the greedy algorithms perform in comparison with the optimal algorithm?
- to place q transcoders where $q < |\mathcal{O}|$, how can the CSP choose the appropriate algorithm?

Comparison with *opt_placement_U*

We now study the performance of the two greedy algorithms as compared with the optimal algorithm.

```
Input:
num_trans: number of transcoders that need to be placed
useful_nodes: Boolean vector indicating nodes that are in the U-set
% the following global variables are used by the algorithm %
linkinfo, pathinfo, clients, numclients, numnodes
Output:
transinfo: Boolean vector that indicates nodes having transcoders

function findplacement_Max_gain (num_trans, useful_nodes);
Initialize transinfo with only one transcoder at source;
[base_act_rates, base_deli_rates]  =  find_opt_rates_I (transinfo)
 % find optimal delivered rates at clients with ST option %
maxgain = 0;
for n = 1 : num_trans
    for m = 2 : numnodes
        current_transinfo = transinfo;
        if node[m] is in useful_nodes
            enable transcoder at node[m] in current_transinfo;
            [act_rates, deli_rates] =
                 find_opt_rates_I (current_transinfo);
            gain = 0;
            for k = 1 : numclients
               gain = gain +
              (deli_rates at clients[k]- base_deli_rates at clients [k]);
            end %find improvement in delivered rates across all clients%
            if gain is greater than maxgain
               maxgain = gain;
               chosen_node = node;
            end
         end
     end
     remove chosen_node from useful_nodes;
     enable transcoder at the chosen_node in transinfo;
     base_deli_rates  =  deli_rates;
end
```

Figure 6.11: Algorithm: *findplacement_Max_gain*

```
Input:
num_trans: number of transcoders that need to be placed
useful_nodes: Boolean vector that indicates nodes that are in the U-set
% the following global variables are used by the algorithm %
linkinfo,  pathinfo, clients, numclients, numnodes
Output:
transinfo: Boolean vector that indicates nodes having transcoders

function  findplacement_Min_loss (num_trans, useful_nodes)
Initialize transinfo with transcoders at all nodes in useful_nodes;
[base_act_rates, base_deli_rates]  =  find_opt_rates_I (transinfo)
 % find optimal delivered rates at clients with AT placement option %
minloss = 0;
for n = 1 : num_trans
    for m = 2 : numnodes
        current_transinfo = transinfo;
        if node [m] is in useful_nodes
            disable transcoder at node[m] in current_transinfo;
            [act_rates, deli_rates] =
                 find_opt_rates_I (current_transinfo);
            loss = 0;
            for k = 1 : numclients
                loss = loss +
                 (base_deli_rates at client[k]- deli_rates at client[k]);
            end % find decrease in delivered rates across all clients %
            if loss is less than minloss
               minloss = loss;
               chosen_node = node;
            end
        end
    end
    remove chosen node from useful_nodes;
    disable transcoder at the chosen_node in transinfo;
    base_deli_rates = deli_rates;
end
```

Figure 6.12: Algorithm: *findplacement_Min_loss*

Set-up: In these experiments, we use the following metrics to compare their performances: (i) average delivered rates at the clients (ii) number of clients affected by the chosen placement: these are the clients, each of which has suffered a reduction in its delivered rate, and (iii) CPU time for the algorithm to converge.

Experiments: As discussed in Section 6.4, since the complexity of *opt_placement_U* is exponential, when the number of useful nodes increases, the number of combinations to be considered for optimal placement of q transcoders explodes. Hence we consider small networks having 40–100 nodes. We find the optimal set and any super nodes, to find the useful set. For all values of q where $q > 1$ and $q <= |O|$, we run *opt_placement_U* and the two greedy algorithms. As an example, we present the result from a topology having 100 nodes with 12 nodes in optimal set and two super nodes.

Results:

- Starting with $q = 2$, we run *opt_placement_U* and the two greedy algorithms for each q value up to $|O|$. The average delivered rates at the clients for each algorithm is found. The first graph in Figure 6.13 plots the average delivered rates for each algorithm. As seen from this figure, both the greedy algorithms perform close to the optimal algorithm.

- When the greedy algorithms are used, the second graph in Figure 6.13 depicts the number of clients for which the delivered rates is less than the delivered rates if the optimal placement were chosen. As seen in this graph, the average delivered rates for the two greedy algorithms are very close, even though different nodes are chosen for placement of transcoders. However, based on the nodes chosen, delivered rates at specific clients can be affected and the number of affected client can differ for these two algorithms. This is an important observation, as a CSP can choose an algorithm based on the service guarantees it has agreed to with specific clients.

- The next parameter we are concerned about is the time taken by the algorithms to find the placement. As shown in the third graph in Figure 6.13, the time of convergence for *opt_placement_U* is bell-shaped. Considering the difference in the scale of time of convergence for *opt_placement_U* as compared with the greedy algorithms, we have used two Y axes: the primary Y-axis (to the left) is calibrated according to the time scale of *opt_placement_U* and the secondary Y-axis (to the right) is calibrated according to the time scale of the greedy algorithms.

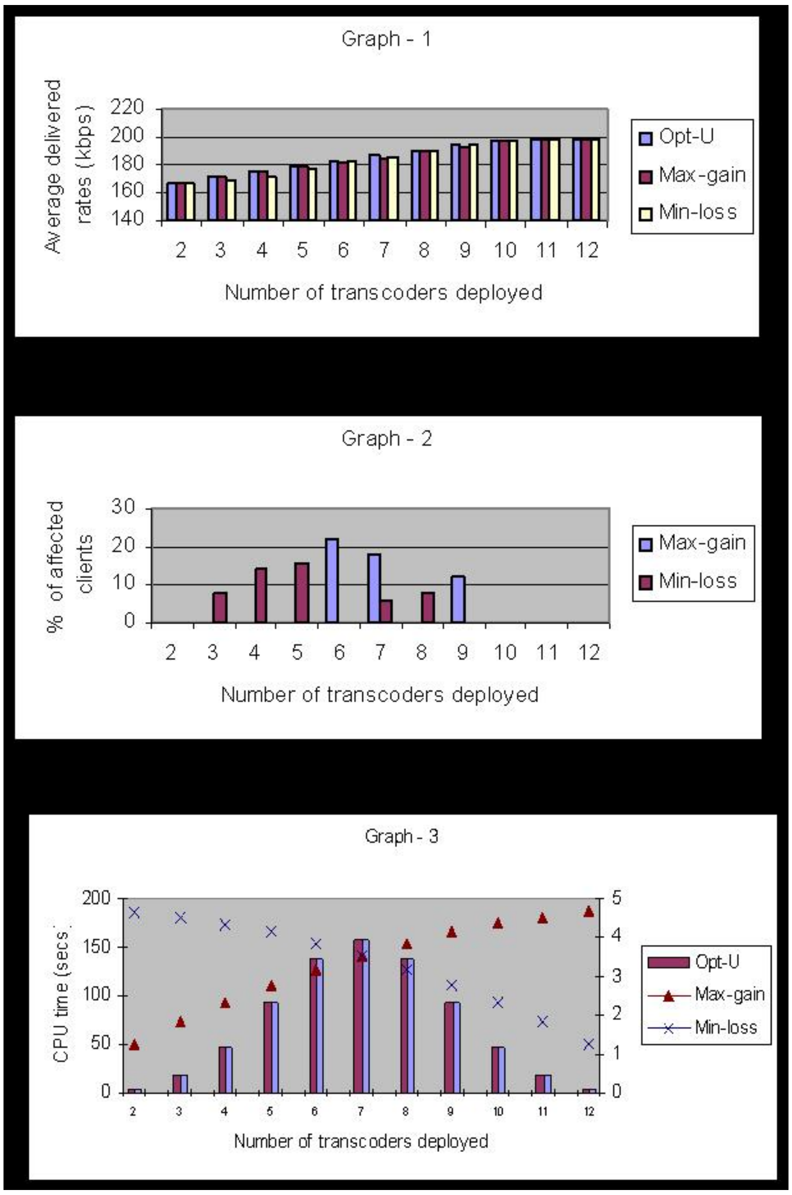


Figure 6.13: Performance of greedy algorithms

– *opt_placement_U* can be used when the number of transcoders to be placed falls at either end of q. For very small values of q or very large values, the number of combinations considered is not too large. In such cases, optimal placement can be found using *opt_placement_U*.

– When the number of combinations to be considered is beyond a threshold (can be set by the CSP), one of the greedy algorithms is used.

– As seen from the third graph in Figure 6.13, when $q < |\mathcal{O}|\ /2,$ *findplacement_Max_gain* is preferred as it takes less time than *findplacement_Min_loss*.

– When $q > |\mathcal{O}|\ /2,$ *findplacement_Min_loss* converges faster as it starts with transcoders at all the nodes in $\mathcal{U}$ and needs to remove fewer transcoders to find the placement.

Next, we consider 20 topologies having 150-300 nodes with link bandwidths randomly chosen from 192-1024 kbps. All clients' minimum rate requirements are set to 128 kbps. Their delay tolerance values are randomly assigned from 30-3600 seconds. For each topology first we set the number of transcoders to be placed $= |\mathcal{O}|\ /4$. The top two graphs in Figure 6.14 depict the average delivered rates at the clients and the CPU time taken by the two greedy algorithms.

The bottom two graphs in Figure 6.14 depict the average delivered rates at the clients and the CPU time taken by the two greedy algorithms, when the number of transcoders to be placed $= |\mathcal{O}| \times 3/4$.

As also seen from these graphs, the *findplacement_Max_gain* works efficiently when few transcoders are to be placed and *findplacement_Min_loss* works efficiently when $q > |\mathcal{O}|\ /2$ transcoders are to be placed.

6.7 Conclusions

In the previous chapter, we explained how the CSP servicing clients as in a distance education application, can find the minimum number of transcoders required to provide the best delivered rates to clients. This provides the upper bound on the investment required for the transcoders. As discussed in the conclusions section of Chapter 4, by providing various classes of service for different premiums, a CSP catering to a set of subscribed clients can enhance its revenue.

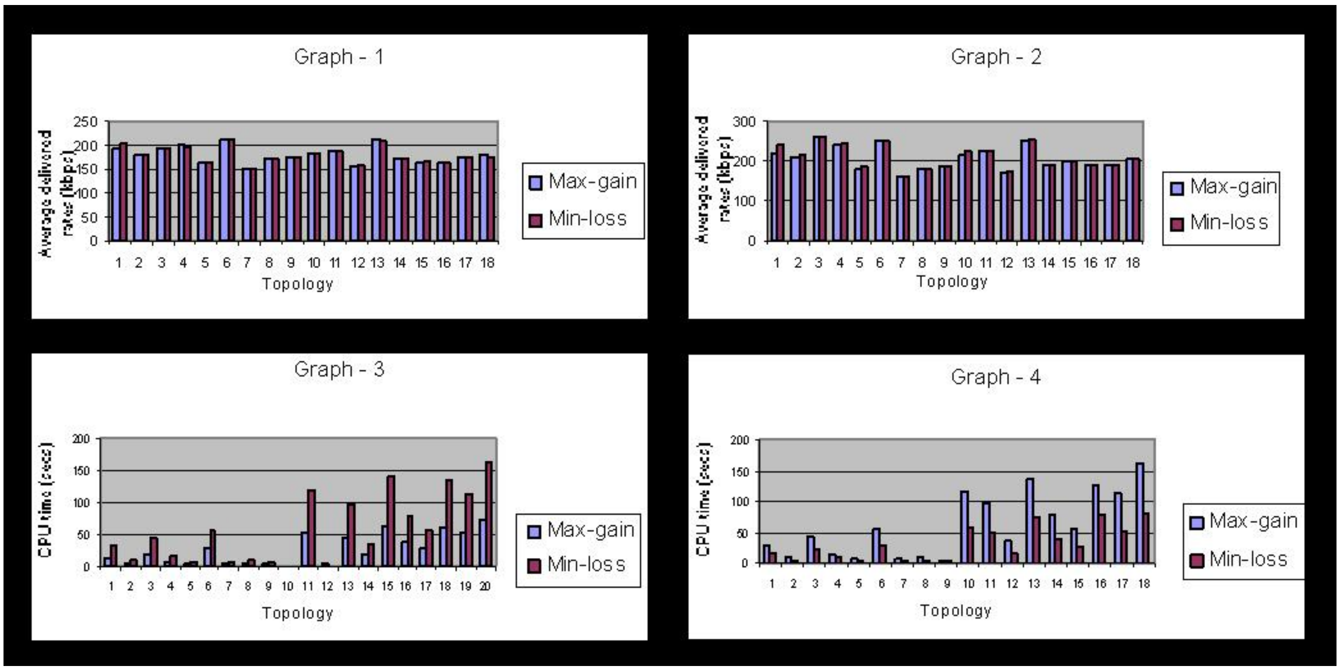


Figure 6.14: Efficiency of greedy algorithms

When the available resources are constrained, a CSP needs to answer the following question: If the number of transcoders q is constrained such that $q < |\mathcal{O}|$, where should the transcoders be placed such that the clients can be serviced with the best rates? Note that this is an important question to answer as it helps the CSP in: (i) making deployment choices for a specific number of transcoders and (ii) using the transcoders efficiently to improve the delivered rates at clients maximally. We have discussed this problem in this section, providing optimal solutions as well as greedy alternatives with lower complexity. This analysis is also useful for a CSP to ensure various classes of service (say, Platinum, Gold, and Silver memberships) guaranteed to the clients, even when the number of resources are constrained. While such an analysis based on the business model of the CSP is beyond the scope of this thesis, the analysis and algorithms provided in this chapter can be used in conjunction with the CSP's cost-revenue model, to make informed choices regarding investment and placement of resources.

Chapter 7

Solutions for scheduled streaming when link bandwidths vary

7.1 Introduction

In the previous chapters we presented solutions for finding (i) the optimal delivered rates at clients, (ii) the optimal placement for transcoders to achieve optimal delivered rates at clients, and (iii) the optimal placement of transcoders, when there is a constraint on the number of transcoders – for the scheduled streaming case when the bandwidths are assumed to be static for the duration of the session. The assumption that bandwidth remains constant over an interval is likely to hold better for shorter length time intervals. This motivates the work described in this chapter.

With respect to the solution context presented in Figure 1.2 in Chapter 1, typically the CSP's distribution network is provisioned; however, the provisioned bandwidth on links in the distribution network may vary with time. Even though links in the access network are typically un-provisioned, a CSP can get information on the type of connectivity a client is subscribed to and may be able to estimate the available bandwidths on the links in the access network. Hence, in a subscription based network, it is possible for the CSP to estimate the available bandwidth over time intervals that make up the session duration. We term the intervals over which bandwidths are expected to be stable as *prediction intervals*.

In this chapter, we analyze the scheduled streaming scenario when the available bandwidths on the links from the source to the clients remain constant over prediction intervals spanning the session duration. Preliminary work on this topic is presented in [27]. We have borrowed the notion of *break point* while finding the stream rates through the links from this work. The following model is assumed for the analysis presented:

1. Predicted bandwidths on the network links are available at $\mathcal{S}$ for prediction intervals spanning the session duration. These values are constant over the duration of the prediction interval. The session duration is divided into prediction intervals of equal duration.

2. The delay tolerance values specified by the clients are multiples of the length of the prediction interval. This assumption simplifies the exposition of the algorithms; otherwise this assumption is not essential.

3. An adaptive layering encoder is available at $\mathcal{S}$ that encodes the stream into appropriate number of layers to best serve the clients.

Our objective is to efficiently utilize the delay tolerance of clients and the bandwidth on the links in their paths, to provide loss-free delivered rates at the clients over each prediction interval. The following questions define the scope of the problem:

1. Given the client requirements and link bandwidths predicted over the session duration, how can we find the delivered rates at clients over each prediction interval leveraging their delay tolerance?

2. How can the available bandwidth on prediction intervals spanning the session duration be effectively utilized to improve the delivered rates at clients?

3. If the network provides more precise bandwidth estimates for the links at the beginning of every prediction interval, how can these estimates be used to revise the delivered rates at the clients?

Before we proceed to answer these questions, we precisely define some of the terms used in this chapter in addition to the terms defined in Chapter 2. We then discuss the flow of data through the links and list some assumptions made in the analysis that follows.

7.1.1 Additional definitions and assumptions

As defined earlier, $\mathcal{T}$ is the playout duration of the content. Let $C_1, C_2, \ldots C_m$ be the clients having delay tolerance values $\delta_1, \delta_2, \ldots \delta_m$. Session duration over which predicted bandwidths are assumed to be known is: $(\mathcal{T} + \max(\delta_i))$.

The following additional terms and assumptions are used in this chapter.

Definitions

Prediction Interval (PI): Time intervals over which bandwidths are predicted on a given link. Prediction interval is denoted by P_i^j, where i denotes the link index and j denotes the interval index.

Prediction Interval Duration: The duration of a prediction interval is defined as the prediction interval duration, denoted by $\mathcal{P}$.

Number of prediction intervals: Number of prediction intervals depends upon the time duration considered and the prediction interval duration. For example, the term *(Session duration/Prediction interval duration)* determines the number of prediction intervals in a session. *(Playout duration/Prediction interval duration)* determines the number of prediction intervals in the playout duration. This is denoted by P.

Predicted bandwidths: Bandwidths are predicted on each link l_i in the multicast tree for prediction intervals spanning the session duration. Predicted bandwidth of P_i^j is denoted by b_i^j.

Available data capacity: Given the predicted bandwidths on a link l_i over the session duration, available data capacity is the total amount of data that *can* flow through l_i (without loss) from the start time of the session t_0 up to the end of a given prediction interval. Available data capacity on l_i up to a prediction interval j is denoted by: A_i^j.

Required data capacity: Given the encoded rate of a stream that flows through a link l_i over the prediction intervals spanning the playout duration, required data capacity is the maximum amount of data that *will* flow through l_i from the start time of the session t_0 up to the end of a given prediction interval, in order to provide loss-free playout at the client. Required data capacity on l_i up to a prediction interval j is denoted by: D_i^j.

Break point: For a given link l_i, in order for the data to be flowing without any loss, for every prediction interval j, $D_i^j <= A_i^j$. Else, a break-point is said to occur at the beginning of the prediction interval $\mathcal{P}_i^j$. This break point indicates that the encoded rate of the stream up to that prediction interval can not be supported without loss.

Sent data capacity: Given the encoded rate of a stream that flows through a link l_i over the prediction intervals spanning the playout duration, sent data capacity is the amount of data that *actually* flows through l_i from the start time of the session t_0 up to the end of a given prediction interval. Sent data capacity on l_i up to a prediction interval j is denoted by: S_i^j.

Active period of a link: According to Theorem 4.1, when a link is shared, the minimum of delay tolerance values of the clients sharing that link determines the upper bound for the stream rate that can flow through that link, in order to serve all the clients without loss. The time duration over which data flows through a link such that the delay tolerance constraints of the clients having the link in their paths are honored is termed the *active period* of that link. The active period of a link is given by: $(\mathcal{T} + min\ (\delta_k))$, where δ_ks are the delay tolerance values of the clients having the link in their paths.

Under-utilization of a link: Consider a link l_i. Let A_i^j be the available data capacity and S_i^j be the sent data capacity on l_i up to P_i^j. Consider the case when A_i^j is greater than S_i^j; We refer to such a situation as under-utilization of l_i or simply say that the *link is under-utilized.*

Before we present the assumptions, we illustrate some of the definitions discussed above and discuss the flow of data through the links.

Illustration of flow of data through the links

We illustrate the flow of data in Figure 7.1 taking a simple topology and considering the path from source $\mathcal{S}$ to client C_1.

With reference to Figure 7.1, let $\mathcal{T} = 5$ prediction intervals and (max(δ_k)) where δ_ks are the delay tolerance values of the clients, be 2 prediction intervals; Hence, the session duration spans 7 prediction intervals. Let the prediction interval duration be $\mathcal{P}$. Consider client C_1. Links l_1 and l_2 are in the path of C_1 from $\mathcal{S}$. Since l_1 is a shared link, its active period is given by: $(5 + min(2, 1)\ ) = 6$ prediction intervals. Active period of l_2 is 7 prediction intervals. We trace the flow of data to C_a when $r_1^1, r_1^2, \ldots\ r_1^5$ are the stream rates generated by $\mathcal{S}$ over $\mathcal{T}$.

- Data sent from $\mathcal{S}$ over the first prediction interval is: $S_1^1 = r_1^1 * \mathcal{P}$.
  Data sent from $\mathcal{S}$ over the first 2 prediction intervals is: $S_1^2 = (r_1^1 + r_1^2) * \mathcal{P}$.
  Hence, total data sent from $\mathcal{S}$ over $\mathcal{T}$ = Sent data capacity = S_1^5 = $(r_1^1 + r_1^2 \ldots\ + r_1^5) * \mathcal{P}$.

- Delay tolerance of C_1 is 2 prediction intervals. Hence, Playout at C_1 must start at the beginning of third prediction interval and end at the seventh interval. Note that the total amount of data played out at C_1 over $\mathcal{T}$ is S_1^5. This amount of data needs to flow through l_1 over 6 prediction intervals and l_2 over 7 prediction intervals.

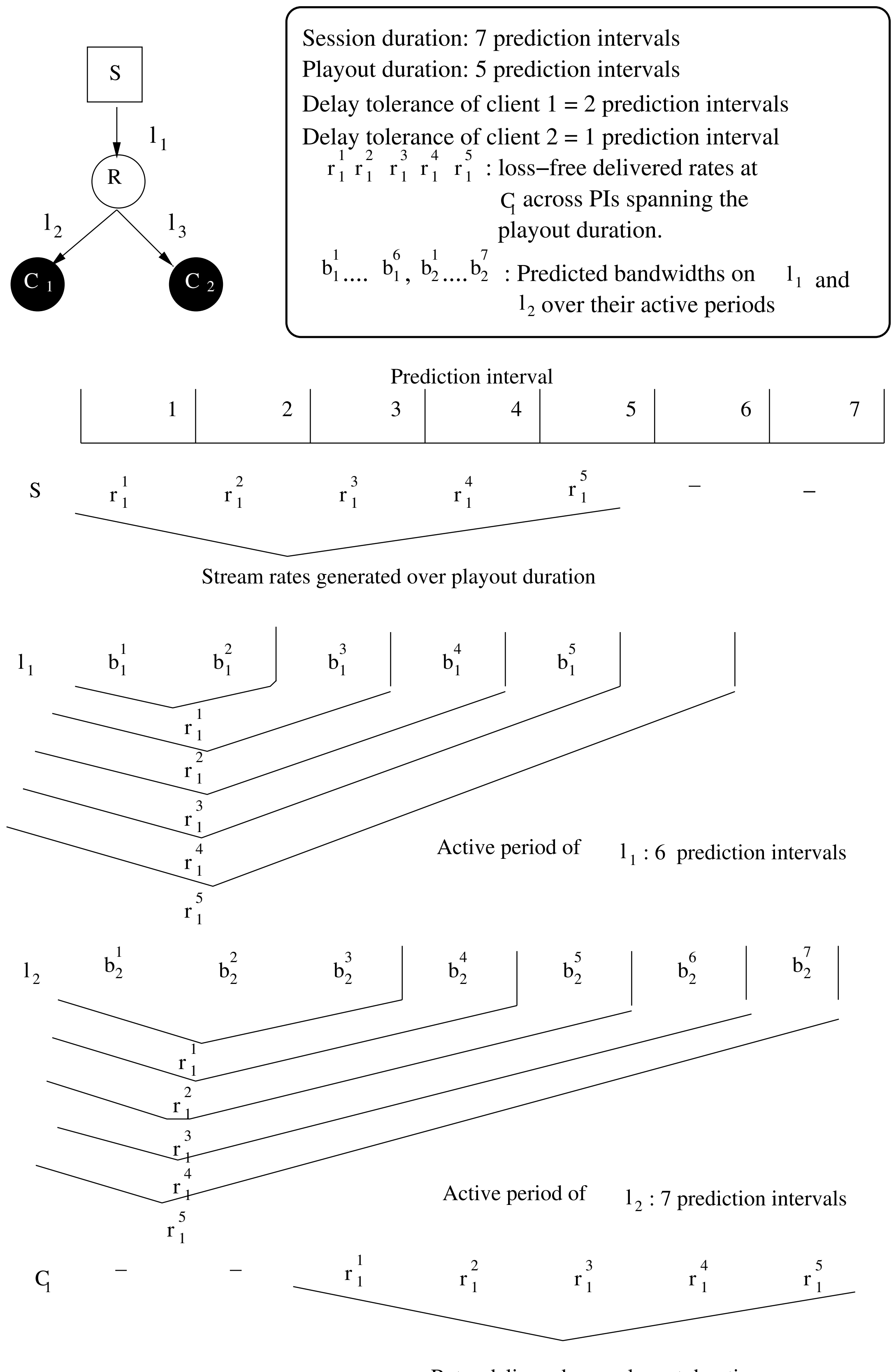

S
R
l_1
l_2
l_3
C_1
C_2
Session duration: 7 prediction intervals
Playout duration: 5 prediction intervals
Delay tolerance of client 1 = 2 prediction intervals
Delay tolerance of client 2 = 1 prediction interval
$r_1^1\ r_1^2\ r_1^3\ r_1^4\ r_1^5$: loss−free delivered rates at C_1 across PIs spanning the playout duration.
$b_1^1 \ldots b_1^6,\ b_2^1 \ldots b_2^7$: Predicted bandwidths on l_1 and l_2 over their active periods
Prediction interval
1 2 3 4 5 6 7
S r_1^1 r_1^2 r_1^3 r_1^4 r_1^5 – –
Stream rates generated over playout duration
l_1 b_1^1 b_1^2 b_1^3 b_1^4 b_1^5
r_1^1 r_1^2 r_1^3 r_1^4 r_1^5
Active period of l_1 : 6 prediction intervals
l_2 b_2^1 b_2^2 b_2^3 b_2^4 b_2^5 b_2^6 b_2^7
r_1^1 r_1^2 r_1^3 r_1^4 r_1^5
Active period of l_2 : 7 prediction intervals
C_1 – – r_1^1 r_1^2 r_1^3 r_1^4 r_1^5
Rates delivered over playout duration

Figure 7.1: Flow of data over prediction intervals spanning session duration

- Available data on l_1 over the first prediction interval is: $A_1^1 = b_1^1 * \mathcal{P}$.

  Available data on l_1 over the first 2 prediction intervals is: $A_1^2 = (b_1^1 + b_1^2) * \mathcal{P}$.

  Hence, total data available over the active period of l_1 = Available data capacity = $A_1^6 = (b_1^1 + b_1^2 \ldots + b_1^6) * \mathcal{P}$

- Required data on l_1 up to the second prediction interval is: $D_1^2 = r_1^1 * \mathcal{P}$.

  Required data on l_1 up to the third prediction interval is: $D_1^3 = (r_1^1 + r_1^2) * \mathcal{P}$.

  Hence, total data required over the active period of l_1 = Required data capacity = $D_1^6 = S_1^5 = (r_1^1 + r_1^2 \ldots + r_1^5) * \mathcal{P}$.

  Similarly, considering that the active period of l_2 is 7 prediction intervals, l_2 must support total required data S_1^5 over 7 prediction intervals such that the playout at C_1 is loss-free.

Flow of data through the links

Generalizing the above illustration, the steps that trace the flow of data from the source $\mathcal{S}$ to a client C_a over the session duration are discussed below.

- The content stream generated at $\mathcal{S}$ spans the playout duration $\mathcal{T}$. Suppose there are P prediction intervals in $\mathcal{T}$. Note that the data is *sent* from $\mathcal{S}$ over P intervals at rates r_a^j, where $j = 1, 2, \ldots P$. Total data sent through a link l_i in C_as path up to j intervals is given by:

$$\forall j, j = 1, 2, \ldots, P, \;\; S_i^j = \sum_{k=1}^{j} r_a^k * \mathcal{P}$$

- The duration over which data can flow through each link is equal to its active period. Consider a link l_i. Suppose the active period of l_i is $(P + d)$. Let b_i^js be the predicted bandwidths on link l_i where $j = 1, 2, \ldots (P + d)$.

  The total amount of available data, that can flow through l_i without loss is given by:

$$\forall j, j = 1, 2, \ldots, (P + d), \;\; A_i^j = \sum_{k=1}^{j} b_i^k * \mathcal{P}$$

- The total amount of data that is required to flow through l_i over its active period is given by $S_i^P = (r_a^1 + r_a^2 + \ldots + r_a^P) * \mathcal{P}$. Suppose the number of prediction intervals in the δ value considered to compute the active period is d, playout at C_a starts in the $(d + 1)^{th}$

prediction interval and ends at $(d + P)$ intervals. In other words, data at rate r_a^1 can reach C_a by the end of d^{th} interval. Thus, data required to flow through l_i over j^{th} interval is given by:

$$\begin{aligned} \forall j, j = 1, 2, \ldots, (P+d),\ \ D_i^j &= 0 && if\ j <= d \\ &= \sum_{k=1}^{j} r_a^k * \mathcal{P} && otherwise; \end{aligned}$$

For the stream to flow through l_i loss-free, the following condition must hold:

$$\forall j, j = 1, 2, \ldots, (P+d),\ \ A_i^j >= D_i^j.$$

- Consider client C_a. Playout at the client starts at $(t_0 + \delta_a)$. The delivered rate at C_a over each prediction interval spanning $\mathcal{T}$ is atmost r_a^j.

Note that in finding the available data capacity, required data capacity and sent data capacity we multiply the rate with the duration of the prediction interval which cancels out in the final rate calculation. Henceforth, we do not explicitly use the prediction interval duration in our calculations and illustrations.

Assumptions

We make the following assumptions in the analysis presented in this chapter:

- *Prediction intervals are of constant and equal duration.*
  Length of prediction intervals spanning the session duration is chosen to be a fraction of $\mathcal{T}$.

- *Predicted bandwidth over a prediction interval remain constant over that prediction interval.*

- *A prediction module that accurately predicts link bandwidths over prediction intervals is available.*
  We assume a feedback based bandwidth prediction mechanism such as the one used in the loss-based rate adaptation scheme described in [43].

- *A source-driven layer encoder adapts the layers for each prediction interval to match the delivered rates at the clients.*

- *Bandwidth estimates received at the beginning of each prediction interval are accurate.*

We briefly outline the solution strategies used to find the loss-free delivered rates at the clients in the next section.

7.1.2 Overview of solution approaches

To find the loss-free delivered rates at the clients over each prediction interval, firstly, we develop a solution by formulating it as an optimization problem. Given the complexity of this approach, we then explore practical alternatives.
Algorithms required to find the delivered rates at clients over each prediction interval depend on the nature of bandwidth variation.

- When the variations are very small, where we can assume available bandwidths to be static over the session duration, we use the following simple solution: for each link, find the minimum of the predicted bandwidths over prediction intervals spanning its active period. Assume this to be the available bandwidth on the link for all the prediction intervals in its active period. Now we can apply the algorithm *find_opt_rates_I* developed in Chapter 4 for the static case to find the delivered rates at the clients. Note that when fluctuations in bandwidths across prediction intervals are high, such a solution would grossly underutilize the links.

- For a given link, when the average of the predicted bandwidths over the session duration captures the variations in the available bandwidth, we can use the average bandwidth as the available bandwidth on the link for all the prediction intervals spanning the session duration. In this case, algorithm *find_opt_rates_I* developed for the static case can be used to find the delivered rates at the clients. However, when fluctuations occur such that the average underutilizes the link or does not sustain the predicted bandwidth, this method would result in lossy transmission.

- Given that the bandwidth remains constant over each prediction interval, another solution would be to consider each prediction interval in isolation. The multicast tree is considered with the predicted bandwidths for each prediction interval and algorithm *find_opt_rates_I* is applied to find the delivered rates at clients over that prediction interval. In this *interval-by-interval algorithm*, we divide the δ value of a client equally across the prediction intervals spanning the playout duration and find the bandwidth of the weakest link in the client's path for each prediction interval.

When the same link in a client's path remains the weakest link across all prediction intervals, and the bandwidth on that link varies very little, this solution would work. When the weakest link in the path of a given client in one prediction interval has high bandwidth in another interval, the link can support a higher encoding rate. This possibility is not exploited by this solution. Even when the same link remains the weakest link in a client's path across all prediction intervals, this solution does not guarantee loss-free playout when the bandwidth of the weakest link fluctuates randomly.

- To ensure efficient utilization of the available bandwidth on each link when the fluctuations cannot be characterized into any of the above three special cases, we need a solution which accounts for the changes in the available bandwidth from one prediction interval to another over the session duration. We propose a *link-by-link algorithm* where each link is considered in isolation to find the maximum loss-free rate that can flow through the link for each prediction interval spanning the session duration. The shared link constraint is then applied to each link to find the actual rates that can flow through it.

  In this solution, delivered rates at the clients are determined considering the stream rates flowing through the links in their paths over each prediction interval. This solution utilizes the fluctuating bandwidths on every link in the network to maximize the delivered rates at the clients.

In all the solutions discussed thus far, we assumed that the predicted bandwidths for each link over the session duration are available up front, at the beginning of the session.

As the last step, we relax this assumption.

- We consider the case where re-estimated bandwidth is available at the beginning of each prediction interval for each link.

- Based on whether the new estimate is lower or higher than the estimate available at the beginning of the session, we adjust the stream rates flowing through a given link and re-compute the delivered rates at the clients.

Approach	Method	Algorithm/s used	Suitable scenarios	Unsuitable scenarios
Converting to static case	link bandwidth = min(predicted b/ws)	find_opt_rates_I	bandwidth variations are small and infrequent	Differnce between lowest and highest values of predicted bandwidths is high
	link b/w = avg(predicted bandwidths)	find_opt_rates_I	Average value captures bandwidth variation	* High bandwidths occur in the beginning * high and low bandwidths are clustered
Interval–by–interval approach	for each PI link bandwidth = predicted bandwidth	find_opt_rates_I for each PI	When the same link remains the weakest link	* bandwidths vary randomly over PIs * any link can be weakest link in a PI
Link–by–link approach	for each link find sustainable stream rate over session duration	find_base_stream_rate find_link_stream_rates find_delivered_rates	when predicted b/ws vary randomly	––
Adjusted Link–by–link approach	for each link find sustainable stream rate considering revised estimate at the beginning of PI, over session duration	find_adjusted_link_stream_rates find_delivered_rates	when predicted b/ws vary randomly	––

Figure 7.2: Overview of the solution approaches

- Assuming a robust prediction module that estimates available bandwidths that are very close to the actual available bandwidths at the beginning of each prediction interval, this algorithm can be expected to adjust the delivered rates at the clients over each prediction interval such that the available bandwidth is maximally utilized.

An overview of the solution approaches is presented in Figure 7.2. When the predicted bandwidths vary, the rate at which the contents are encoded needs to be adapted. Note that based on the fluctuations in the available bandwidth on a given link, the stream rate supported by the link can go up or down. Layer encoders can be used for adapting the rate at which the contents are served. Layering is a well researched mechanism with the adaptation initiated either from the receiver or the source [25][47]. In our solutions we assume a source-driven layer encoder as discussed in [2], that adapts the layers for each prediction interval to match the delivered rates at the clients.

Before we discuss the solution approaches in detail, we present a running example that is used to illustrate the various solution approaches.

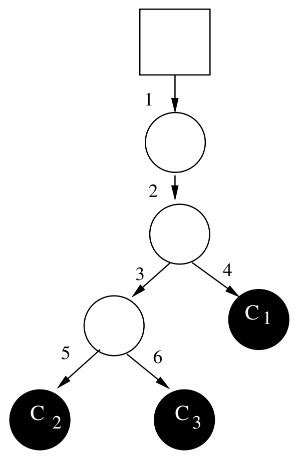


Figure 7.3: Example to illustrate solution approaches

Number of nodes $N = 7$;

Number of links $L = 6$;

Number of clients $C = 3$;

Playout duration $\mathcal{T}$ = 100 seconds;

Prediction interval duration $\mathcal{P}$ = 20 seconds;

Delay tolerance δ_1 of C1 = 20 seconds;

Delay tolerance δ_2 of C2 = 40 seconds;

Delay tolerance δ_3 of C3 = 20 seconds;

Number of prediction intervals $\mathcal{N}$ =

$(\mathcal{T} +$ max $(\delta_i))$ $/\mathcal{P}$ = $= 140/20 = 7$;

Figure 7.4: Example: Network and application parameters

Link	Bandwidths over prediction intervals						
	1	2	3	4	5	6	7
Link1	384	384	384	256	448	512	512
Link2	128	256	448	512	256	128	384
Link3	256	128	128	192	128	256	128
Link4	384	384	320	192	320	128	128
Link5	128	192	128	384	128	128	192
Link6	384	256	384	128	192	256	192

Table 7.1: Example: Predicted bandwidths over the session duration

7.1.3 Example used to illustrate solution approaches

Consider a network as shown in Figure 7.3 with parameters as given in Figure 7.4. Predicted bandwidths over the seven prediction intervals spanning the session duration are given in Table 7.1. We will be referring to this example while discussing the various solutions in the sections that follow.

7.2 Optimization-based approach

In this section, we formulate the problem of finding the loss-free delivered rates at the clients as an optimization problem. Recall that the playout duration and clients' delay tolerance values are assumed to be multiples of the prediction interval duration.

Design variables: Stream Rates r_n flowing through links l_n for the playout duration $\mathcal{T}$.

Objective function: Maximize delivered rates at the clients over each prediction interval spanning the playout duration. Delivered rate at a client depends on the encoded rate of the stream flowing through the last link in its path; i.e., for each prediction interval, for the last link in each client's path, the rate that flows through the link should be as close to the base encoding rate (maximum stream rate) as possible. This is written as:

Minimize $\forall k, k = 1, 2, \dots m$, $\sum_P (\Gamma - r_l^k)^2$, where Γ is the base encoding rate and r_l^k is the rate flowing through the last link l_l for each client k over each prediction interval in P.

Constraints:

- **Rate constraint**: This constraint ensures that the stream rate flowing across any link l_i in the network at any prediction interval $j \in P$ is bounded by Γwhich is the highest possible stream rate. We represent this constraint as:
  $r_i^j <= \Gamma, \forall i, i = 1, 2, \dots n, \forall j, j = 1, 2, \dots P$.

- **Layer constraint**: This constraint captures the property of layering, i.e., the stream rates flowing through two consecutive links must be such that the stream rate through the first link is greater than or equal to the rate flowing through the next one. Suppose r_{i-1}^j and r_i^j are the stream rates flowing through two consecutive links l_{i-1} and l_i for a given prediction interval j where $j \in \{1, 2, \dots P\}$. This constraint is denoted as:
  $r_{i-1}^j >= r_i^j, \forall i, i = 1, 2, \dots n, \forall j, j = 1, 2, \dots P$.

  We summarize the optimization function and the above two constraints in Figure 7.5.

Objective function: Minimize $\forall k, k = 1, 2, \ldots m, \sum_P (\Gamma - r_l^k)^2$,

where Γ is the base encoding rate and r_l^{jk} is the rate flowing through the last link l_l

for each client k over each prediction interval j in P.

Constraints:

Rate constraint: $r_i^j <= \Gamma, \forall i, i = 1, 2, \ldots\ n, \forall j, j = 1, 2, \ldots\ P$,

where n is the number of links in the multicast tree and P is the number of prediction intervals

in the session duration.

Transcoder constraint:$r_{i-1}^j >= r_i^j, \forall i, i = 1, 2, \ldots\ n, \forall j, j = 1, 2, \ldots\ P$.

Delay tolerance constraint: The two parts of this constraint are presented in

Figure 7.6 and Figure 7.7.

Figure 7.5: Optimization formulation

- **Delay tolerance constraint**: There are two parts to this constraint:

  1. *Constraints for loss-free transmission over a link*: Considering each link over prediction intervals spanning its active period, the following two conditions must hold:

     (a) For a given link at the end of its active period, buffer (at the node sending the stream through the link) should be empty. This constraint function is presented in Figure 7.6.

     (b) At any prediction interval over the active period of a link, the available data capacity should be less than or equal to the required data capacity. This constraint function is presented in Figure 7.7.

  2. *Constraint for loss-free transmission to a client*: Considering each client over each prediction interval spanning the playout duration, the following condition must hold: For a given client, for each prediction interval over the playout duration, for each link in its path, the available data capacity should be less than or equal to the required data capacity. This constraint function is also presented in Figure 7.7.

We implemented the optimization function in **Matlab** and ran it for the example in Figure 7.3. The resulting stream rates through the links are given in Table 7.2 and the delivered rate at the clients are presented in Table 7.3.

Constraint function used in the optimization approach :

```
Input: Stream rates x, through links over playout duration
Global variables used:
TRANSPREDINT: Number of prediction intervals in Playout duration
PRED_INT_DURATION: Duration of each prediction interval
Vector_map: vector that provides the starting values for rates that
correspond to each link
numclients, pathinfo
Output:
Equality and inequality constraints that ensure loss-free playout

/* ceq constraints are equality constraints which enforce
       the condition that at the end of the active period, buffers
       must be empty */
for each link L
    find active period;
    buffer = 0;
    curr_row = vector_map(L,1);
    for each prediction interval PI spanning its active period
        pred_bw = pred_table(L, P);
        if PI is less than or equal to TRANSPREDINT
            buffer = buffer + stream rate x;
        end
        if buffer is greater than pred_bw
            buffer = buffer  pred_bw;
        else
            buffer=0;
        end
    end
    ceq(L,1) = buffer ;
end
```

Figure 7.6: Part 1: Constraints arising from delay-tolerance requirements

```
for each link L
    avail_data_capacity = 0;
    req_data_capacity = 0;
    additional_intervals = active period - playout duration;
    for each prediction interval PI spanning its active period
       avail_data_capacity = avail_data_capacity + pred_table(L, P);
       if PI is greater than additional_intervals
           req_data_capacity = req_data_capacity + stream rate x;
       end
    end
    c(L,1) = req_data_capacity - avail_data_capacity ;
    /* These inequality constraints enforce the condition that
       req_data_capacity is less than or equal to avail_data_capacity
       for prediction intervals spanning the active period of a link*/
end

start_point = numlinks;
for each client C
    for each link L in Cs path
        avail_data_capacity = 0;
        req_data_capacity = 0;
        for each prediction interval PI spanning its active period
            avail_data_capacity = avail_data_capacity + pred_table(L, P);
            if PI is greater than additional intervals in active period
               req_data_capacity = req_data_capacity + stream rate x;
            end
        end
        c((start_point + L),1)
           = req_data_capacity - avail_data_capacity;
  /* These inequality constraints enforce the condition that
       req_data_capacity is less than or equal to avail_data_capacity
       for prediction intervals spanning the active period for links
       in the path of a client*/
    end
end
```

Figure 7.7: Part 2: Constraints arising from delay-tolerance requirements

Link	Stream rates for each prediction interval				
	1	2	3	4	5
Link1	493.71	300.43	449.40	271.95	350.29
Link2	491.52	276.41	387.51	269.35	302.6
Link3	255.9	219.91	208.38	207.11	196.6
Link4	491.52	276.41	387.51	269.35	302.6
Link5	212.99	219.91	208.38	207.11	196.6
Link6	214.01	219.91	208.38	207.11	196.6

Table 7.2: Stream rates through links over session duration (for topology of Figure 7.3)

PI	Delivered rates		
	C1	C2	C3
1	491.52	212.99	214.01
2	276.41	219.91.12	219.91
3	387.51	208.38	208.38
4	269.35	207.11	207.11
5	302.6	196.6	196.6
Average	345.58	208.99	209.2

Table 7.3: Using optimization function: delivered rates for clients in Figure 7.3

Complexity of the optimization formulation

The design variables are the stream rates flowing through the links over the playout duration. Given the upper and lower bounds for the stream rates that can flow through the links, suppose there are v possible values, the optimizer needs to consider at least $(v^{n \times P})$ combinations to find the loss-free rates that can flow through the links. As discussed, the stream can flow through each link over its active period. Note that the active period of any link has at least P intervals. Let p_a be the average number of active periods for the links in the network. Thus, the constraints applied to the rates flowing through the links are in the order of $(n \times p_a)$ and constraints applied to the rates delivered at the clients are in the order of $(m \times n_a)$, where n_a is the average number of links in a client's path. The exponential search space of this approach renders it impractical to apply it to a network with hundreds of links each of which can take values from a wide range of bandwidth values over each prediction interval.

In the following sections, we present several algorithms starting with some simple ones. Some of the insights gained from the simple solutions are used to devise solutions that use the link bandwidths efficiently to provide enhanced rates to the clients. We use a running example to illustrate each of the solutions presented.

7.3 Methods that use the scheduled streaming, static bandwidth formulations

Given that for every link, predicted bandwidths over the session duration are known, we consider the following algorithms:

7.3.1 Using the minimum predicted bandwidth

In this solution, for each link we choose the lowest value of the predicted bandwidth across all prediction intervals and assume this to be the available bandwidth on the link for the session duration. We outline the algorithm below:

```
-----------------------------------------------------------------
For each link in the network
    Find the minimum bandwidth across all prediction intervals
    spanning the session duration.
     % This bandwidth is guaranteed to be available on the link
```

```
       throughout the session duration. Thus, we have a static
       topology with known link bandwidths. %
end
Run algorithm find_opt_rates_I to find the delivered rates
at the clients with AT option for transcoder placement.
-----------------------------------------------------------------
```

When the link bandwidths fluctuate very little, this easy to implement algorithm would work well. We now quantify *very little fluctuations*, when this naive algorithm can be used:

Consider a link l_i having a bandwidth b_i. Suppose r_b is the rate at which the base layer of the stream is encoded. Suppose a minimum of g additional bits are required if another enhancement layer is to be generated. Note that when b_i fluctuates between r_b and $(r_b + g/100)$ kbps, these fluctuations are not *significant* as **no additional layer can be generated**. We term this a *near-static case*. In such a case, even though b_i is varying, we can use $min(b_i)$ as the available bandwidth on l_i across all prediction intervals and apply the simple algorithm.

When the difference between the highest and lowest values of bandwidth over the prediction intervals spanning the session duration is high, this method grossly underutilizes the links, delivering the stream encoded at low rates, compromising quality, as shown in the illustrative example discussed below.

Illustration: using minimum bandwidth

Considering the predicted bandwidths as given in Table 7.1, bandwidths available on the links are:
Link 1: 256 kbps;
Link 2 - Link 6: 128 kbps.
Using algorithm *find_opt_rates_I* we find the delivered rates at the clients: C_1 : 153.6 kbps; C_2 : 153.6 kbps; C_3 : 153.6 kbps; Note that the links are under-utilized over most prediction intervals.

7.3.2 Using the average predicted bandwidth

Another simple solution is to find the delivered rates at clients using the average of the available bandwidth on each link and assume this average value to be the link bandwidth for the session

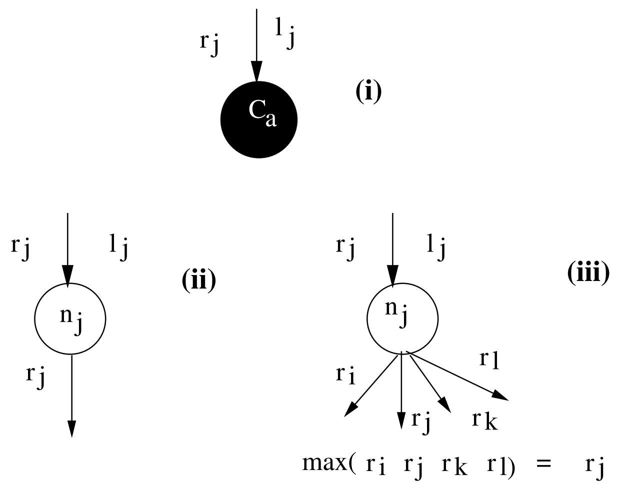


Figure 7.8: Instances of links

duration. By assigning the average rate to each link for the session duration, we can use the algorithms developed for the static case to find the delivered rates at the clients.

The first step in computing the average predicted bandwidth on a link is to find the appropriate value of δ_i to be used for determining the number of prediction intervals considered in the computation.

Choosing appropriate value of δ_i

Consider a link l_j. l_j can be a link directly connected to a client or it can be in the path of one or more clients as shown in Figure 7.8. In either case, the upper bound on the stream rate through l_j is calculated by Theorem 4.1 as: $b_i * (1 + (\delta_p/\mathcal{T})$, where $\delta_p = \min(\delta_k)$, δ_ks are the delay tolerance values of C_k, $k = 1, 2, \ldots m$, having l_j in their path. When bandwidths are predicted, in order to provide loss-free transmission to clients over all the prediction intervals spanning the playout duration, it is necessary that the lowest delay tolerance value of the clients sharing a link is considered for calculating the sent data capacity over the active period of the link.

Thus, for each link in the multicast tree, we choose the minimum of the delay tolerance values of the clients having the link in their paths and use this extra time to accumulate data.

Illustration: using average predicted bandwidth

We use the same example as shown in Figure 7.3 to illustrate this solution.
The average bandwidths available on the links are given below:
Link 1: 394.7 kbps;

Link 2: 288 kbps;

Link 3: 181.3 kbps;

Link 4: 288 kbps;

Link 5: 182.85 kbps;

Link 6: 266.7 kbps;

Using these as the static bandwidths available for the session duration, we run algorithm *find_opt_rates_I* to determine the optimal delivered rates at the clients: C_1 gets 345.6 kbps while C_2 and C_3 get 217.56 kbps.

When the average bandwidth captures the variation in the predicted bandwidths on a link over its active period, it can be used as the static available bandwidth across all prediction intervals. However, the following two situations can arise:

1. the average may fail to capture under-utilization of the link bandwidths in some intervals; this will lead to lossy transmission as the average rate cannot be supported during other intervals having less bandwidth. Such a situation arises when bandwidths in consecutive intervals are much higher than the average rate.

2. the average may fail to capture the shortage of bandwidth at a prediction interval to support the average rate. Such a situation arises when bandwidths in consecutive intervals are much less than the average.

We illustrate the above two cases as well as a case when the average bandwidth works, taking three instances of predicted bandwidths on a given link, say link 2. We will consider 6 prediction intervals to calculate the average for this link as this link is shared between all the three clients.
Predicted bandwidths on link 2:

- Case 1: $128, 256, 448, 512, 256, 128$ over prediction intervals 1, 2, 3, 4, 5, and 6 respectively as given in Table 7.1.

- Case 2: $512, 128, 256, 448, 256, 128$ over prediction intervals 1, 2, 3, 4, 5, and 6 respectively.

- Case 3: $128, 256, 128, 256, 448, 512$ over prediction intervals 1, 2, 3, 4, 5, and 6 respectively.

PI	Stream rate	Predicted b/w	Data sent	Data buffered	Available data capacity(A_i^j)	Required data capacity(D_i^j)
1	345.6	128	128	217.6	128	0
2	345.6	256	(217.6 + 38.4)	307.2	384	345.6
3	345.6	448	(307.2 + 140.8)	204.8	832	691.2
4	345.6	512	(204.8 + 307.2)	38.4	1344	1036.8
5	345.6	256	(38.4 + 217.6)	128	1600	1382.4
6	0	128	128	0	1728	1728

Table 7.4: Case:1

PI	Stream rate	Predicted b/w	Data sent	Data buffered	A_i^j	D_i^j
1	345.6	512	345.6	0	345.6(UU)	0
2	345.6	128	128	217.6	473.6	345.6
3	345.6	256	(217.6 + 38.4)	307.2	729.6	691.2
4	345.6	448	(307.2 + 140.8)	204.8	1177.6	1036.8
5	345.6	256	(204.8 + 51.2)	294.4	1433.6	1382.4
6	0	128	128	166.4	1561.6	1728

Table 7.5: Case:2 – UU indicates Under-Utilization of the link

The average rate is the same in all three cases, 288 kbps. This rate is assumed to be available on the link for the session duration. Thus, using Theorem 4.2, the stream rate that flows through the link is computed as: $(288 + (288 * 20/100)) = 345.6$ kbps. A stream encoded at 345.6 kbps flowing over 5 prediction intervals needs to be supported without loss by this link over 6 prediction intervals. We present the flow of the stream for the three cases in Tables 7.4, 7.5, and 7.6.

Note that the maximum amount of data that can flow through an outgoing link from a given node n_j at each prediction interval is bounded by the predicted bandwidth on the link for that prediction interval. Any additional data is buffered at n_j. The stream is sent encoded at the average rate over prediction intervals spanning the playout duration. At every prediction interval, any data in the buffer is sent first before considering data from the stream for that prediction interval. (We indicate this in the *data sent* column in the Tables 7.4, 7.5, and 7.6).

- In the first case, 345.6 kbps is supported without any loss by the links as $A_i^j >= D_i^j$,

PI	Stream rate	Predicted b/w	Data sent	Data buffered	A_i^j	D_i^j
1	345.6	128	128	217.6	128	0
2	345.6	256	(217.6 + 38.4)	307.2	384	345.6
3	345.6	128	128	(179.2 + 345.6) = 524.6	512 (BP)	691.2
4	345.6	256	256	(268.8 + 345.6) = 614.4	768	1036.8
5	345.6	448	448	(166.4 + 345.6) = 512	1216	1382.4
6	0	512	512	0	1728	1728

Table 7.6: Case:3 – BP indicates a Break Point

at every prediction interval j and the available bandwidths on the link are utilized properly. Note that the algorithm is simple to implement and uses the available bandwidth efficiently in this case.

- In the second case, the links are under-utilized in the first prediction interval. This leads to lossy transmission at the end of the playout as the available bandwidth on the link is not adequate to send all the buffered data. In our example at the end of the playout, 166.4 kbps of data remains in the buffer. When a link is under-utilized at a prediction interval, the available bandwidth at a later prediction interval falls short of what is needed to support the buffered data. In this case, the average rate fails to capture the high availability of bandwidth in the initial prediction interval.

- In the third case, a break-point is detected at the beginning of the third prediction interval where the required data capacity 691.2 kbps exceeds available data capacity 512 kbps. From the beginning of the second prediction interval (when the playout starts at a client having this link in its path) the link needs to support the stream at 345.6 kbps in order to provide loss-free playout at the client. In this case, the available bandwidth is not enough to support this average rate in the third prediction interval.

Thus, when the nature of bandwidth variation over the prediction intervals is random, this method does not guarantee loss-free transmission, as the average rate does not capture the fluctuations in the bandwidth. This solution may result in lossy transmission, due to under-utilization of links or inadequate bandwidths on links to support the stream encoded at the average rate. While the above results are obvious, we have presented the example for completeness. Also, these insights are applied to (in the second step of) the link-by-link algorithm to find the base stream rate, which is discussed in Section 7.5.1.

7.4 An interval-by-interval solution

As we have established in our analysis in the previous chapters, the extra time available due to the client specified delay tolerance can be leveraged to enhance the delivered rates at the clients. In the two solutions discussed thus far, the extra time available for collecting data is spread across all the prediction intervals. However, we did not pay attention to the bandwidth available on the weakest link in each prediction interval. We propose a solution that considers each prediction interval as an instance of the static bandwidth case; the weakest link bandwidth for each prediction interval is considered to find the delivered rates at the clients for that prediction interval.

We start with Theorem 4.1 discussed in Chapter 4, to understand the implication of prediction intervals while using the algorithm developed for the static bandwidth case. We present the theorem again for ease of understanding:

Theorem 4.1: l_i having bandwidth b_i is a link in the path of one or more clients $C_1, C_2, \dots C_m$. r_i, the maximum stream rate that can flow through l_i, is given by $b_i * (1 + (\delta_p/\mathcal{T}))$, where $\delta_p = min(\delta_k)$, δ_ks are the delay tolerance values of C_ks, $k = 1, 2, \dots m$. i.e.,

$$r_i^{max} = b_i * (1 + (\delta_p/\mathcal{T})) \tag{7.1}$$

The above expression is the same as Equation 4.4, which is derived for the case when the link bandwidths remain constant over the session duration; this is equivalent to having only one prediction interval spanning the playout duration $\mathcal{T}$.

In the interval-by-interval solution, we consider each prediction interval spanning the playout duration as a static instance of the multicast tree. As we have seen, to find the delivered rates at the clients when link bandwidths are static, we use algorithm *find_opt_rates_I*. The basis of this algorithm is to find the maximum stream rate that can flow through each link in the multicast tree. To find the maximum stream rate through a given link, we use Theorem 4.1, which in turn uses the bandwidth of the link considered and the clients- delay tolerance values that have the link in their path. Hence, to use algorithm *find_opt_rates_I* over each prediction interval spanning the playout duration, we need to determine the following:

- Delay tolerance of each client for each prediction interval spanning the playout duration. We divide the delay tolerance of each client equally across the prediction intervals spanning the playout duration; $\forall k, k = 1, 2, \dots m$, $\delta_k^j = \delta_k / P, \forall j, j = 1, 2, \dots P$.
- Bandwidth available on the link considered and the time duration over which the bandwidth remains constant; We set the period over which the bandwidth remains constant to the prediction interval duration, $\mathcal{P}$.

Since the predicted bandwidth remains constant over each prediction interval, equation 4.4 is applied to link l_i having bandwidth b_i over a given prediction interval j as:

$$r_i^{j(max)} = b_i^j * (1 + ((\delta_p / P)/\mathcal{P})) \tag{7.2}$$

Similarly, in the equation for maximum deliverable rate at a client (Theorem 4.2, Chapter 4), values of delta and prediction interval duration as derived above, are used. Using the modified equations in *find_opt_rates_I* along with the AT option for transcoder placement, the delivered rates at the clients for each prediction interval are found. Note that all the data structures and global variables used in algorithm *find_opt_rates_I* are used by the interval-by-interval algorithm. These are presented in Figures 4.10 and 4.8 in Chapter 4.

The interval-by-interval algorithm referred to as *I-by-I algorithm* is outlined in Figure 7.9.

Illustration of I-by-I algorithm

With reference to the example in Figure 7.3 and the predicted bandwidths given in Table 7.1, we calculate the delivered rates at the clients treating each prediction interval as an instance of static case using algorithm *find_opt_rates_I*. Prediction interval duration used is 20 seconds. The δ values of the clients are applied equally across prediction intervals spanning the playout duration:

delta_applied for C_1 = $(20/5) = 4$ seconds.

delta_applied for C_2 = $(40/5) = 8$ seconds.

delta_applied for C_3 = $(20/5) = 4$ seconds.

```
Output:
delivered_rates [i,j]: delivered rate of a client
i over prediction interval j.

Use AT option for transcoder placement;
for each PI
    read the predicted bandwidths into matrix M,
    representing the topology;
    for each client C_i
       delta_applied =
          delta of C_i/number of PIs in playout duration;
    end
    duration_for_delta = prediction interval duration;
    [stream_rates, opt_deli_rates] = find_opt_rates_I(transinfo);
    delivered_rates(:, PI) = opt_deli_rates;
end
```

Figure 7.9: I-by-I algorithm

The delivered rates calculated for each prediction interval using algorithm *find_opt_rates_I* is given in Table 7.7.

A critique of the interval-by-interval solution

Advantages: This solution where each prediction interval is considered in isolation to find the delivered rates at the clients may work well in certain practical scenarios. For example, in a distance education application where a CSP has a provisioned distribution network, bottleneck links occur in the access network; typically such bottlenecks occur at the last link to the client. In such cases, it can be assumed that the same link remains the weakest link in a client's path; in addition, when the link bandwidths vary very little over the prediction intervals spanning the session duration, the interval-by-interval algorithm can be used to find the delivered rates at the clients.

Disadvantages: However, there are two major shortcomings in this solution which limit application of this algorithm to scenarios where link bandwidths vary randomly: (i) consideration of each prediction interval in isolation and (ii) lack of consideration of the predicted bandwidths

PI	Delivered rates		
	C1	C2	C3
1	133.12	133.12	133.12
2	266.14	133.12	133.12
3	322.8	133.12	133.12
4	199.68	199.68	133.12
5	266.24	133.12	133.12
Average	237.6	146.4	133.1

Table 7.7: Interval-by-interval solution: delivered rates

over all intervals spanning the active period of a link.

Given these shortcomings of the I-by-I algorithm, we need a solution that exploits both the delay tolerance and the available bandwidth across prediction intervals. We propose such a solution in the next section.

7.5 A link-by-link solution

This solution finds the maximum loss-free rate supported by each link for each prediction interval by considering the bandwidths available on each link over its active period; thus it implicitly considers the fact that the weakest link in a client's path in one prediction interval may not be the weakest link in another prediction interval. By using the buffering capability at the nodes, this algorithm finds the stream rate that utilizes the available bandwidth on each link effectively.

With reference to Figure 7.1, consider a link l_i. Suppose the active period of l_i is $(P + 1)$. Let the predicted bandwidths on l_i be b_i^j, where $j = 1, 2, \dots\ (P + 1)$. The maximum data rate that can flow through l_i is given by:

$$d_i^{max} = \sum_{j=1}^{(P+1)} b_i^j \times \mathcal{P} \tag{7.3}$$

This data rate is delivered across l_i over P prediction intervals. Thus, the maximum stream rate that can be supported by l_i without loss is given by:

$$r_i^{max} = d_i^{max}/(P * \mathcal{P}) \tag{7.4}$$

The overall link-by-link algorithm referred to as *L-by-L algorithm* is presented in the next section.

7.5.1 Steps in the L-by-L algorithm

The first step is to find the stream rate that can be supported without loss by each link over its active period. For a given link, we start with the maximum stream rate calculated using Equation 7.4. As discussed in Section 7.1, the following *constraints for loss-free transmission over a link* must hold:

1. *For a given link at the end of its active period, buffer should be empty.* This condition ensures that there is no loss of data due to underutilization of the available link bandwidth as illustrated in Section 7.3.2 in the example presented in Table 7.5. Algorithm *find_base_stream_rate* presented in Figure 7.14, finds the stream rate that ensures that the link is not under-utilized over its active period.

2. *At any prediction interval P_i^j over the active period of a link l_i, the available data capacity should be less than or equal to the sent data capacity*, i.e., $A_i^j <= S_i^j$.

   Using the base stream rate as the start rate, the loss-free stream rate that flows through the link for each prediction interval over its active period is found using Algorithm *find_link_stream_rates* presented in Figure 7.16. This step uses the insight from the analysis in Section 7.3.2 as shown in the example presented in Table 7.6.

Having found the loss-free stream rates, r_i^P that can flow through a link l_i over each prediction interval P spanning the playout duration, the next step is to find the delivered rates at the clients considering the links in the path of each client.
The stream rate that can be delivered at any client C_k for each prediction interval P, is given by:

$$r_{c_k}^{max} = min(r_i^P), \forall l_i \in p(C_k) \tag{7.5}$$

Algorithm *find_delivered_rate* presented in Figure 7.17 considers the links in a given client's path and finds the loss free rate delivered at the client over prediction intervals spanning the playout duration.

The overall L-by-L algorithm is presented in Figure 7.10. Before we discuss the three labeled components of the overall algorithm in detail in Sections 8.1, 8.2, and 8.3, we present a comparative study of the I-by-I and L-by-L algorithms.

```
for each link L
      find active period of L;
      7.6.1. find_base_stream_rate;
      % this function finds the stream rate ensuring that the link
       is not under-utilized at any prediction interval. Using
       function find_rate it computes the maximum stream rate
       and uses it as the initial value. %

      7.6.2. find_link_stream_rate;
      % this function returns the stream rates that can flow
       through a given link ensuring that the stream rates are
       sustained without loss over the session duration. %
end
for each client C
    7.6.3. find_delivered_rate;
     % this function finds the delivered rate at a client
        given the stream rates that can flow through the
        links in the client's path %
end
```

Figure 7.10: L-by-L algorithm

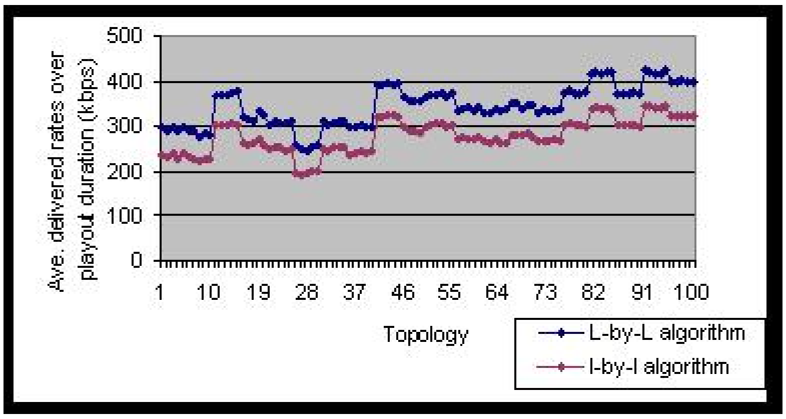


Figure 7.11: Comparison of L-by-L and I-by-I algorithms

7.5.2 Comparison of L-by-L and I-by-I algorithms

To understand the disadvantages of the I-by-I algorithm and to motivate the need for the L-by-L algorithm, we present comparison of these two algorithms, considering two cases: (i) when the weakest link in a client's path changes every prediction interval and (ii) when the same link remains the weakest link in a client's path for the session duration.

When the weakest link changes from one prediction interval to another

In the I-by-I algorithm, we consider each prediction interval in isolation. In the L-by-L algorithm which is discussed in the next section, we consider each link across all prediction intervals over its active period to find the stream rates that the link can support over the playout duration. We present the delivered rates as computed by the two algorithms in Figure 7.11. When the link bandwidths are randomly varying, we find that L-by-L algorithm always outperforms the I-by-I algorithm. This is because the I-by-I algorithm does not take into account the fact that a link which is the weakest in a client's path in one interval may have high bandwidth in a subsequent interval and hence can support a higher rate. Thus, this algorithm makes poor use of the available bandwidth.

When the weakest link remains constant over the prediction intervals

From our argument above it seems that when the weakest link remains the same, the I-by-I algorithm must work fine. However, this need not be true. Note that the algorithm considers the predicted bandwidths over prediction intervals spanning the playout duration, while the stream actually flows through the link over its active period. The loss-free stream rate through a link

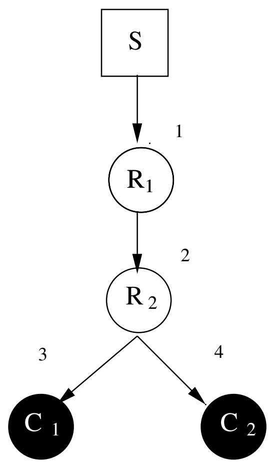


Figure 7.12: Example: Shortcoming of I-by-I

depends on the available bandwidths on the link over its active period. Since, the algorithm does not consider this fact, the following two situations may arise:

1. The algorithm may compute delivered rates which can not be delivered without loss.
2. The algorithm may underestimate the loss-free delivered rates at the clients.

We illustrate these two situations with a simple example. Consider the topology as shown in Figure 7.12. Link 2 is the weakest link across all prediction intervals spanning the session duration. There are 14 prediction intervals in the session duration and 10 in the playout duration. Active period of Link 2 spans 12 prediction intervals. We consider two estimates for the predicted bandwidths on link 2. The estimates differ only in prediction intervals 11 1nd 12 as shown below:

Estimate 1: 272, 240, 208, 176, 208, 176, 272, 240, 272, 240, 208, 176;
Estimate 2: 272, 240, 208, 176, 208, 176, 272, 240, 272, 240, 336, 304;
Average delivered rates:
I-by-I algorithm:
Estimate 1: 276.48; Estimate 2: 276.48.
L-by-L algorithm:
Estimate 1: 268.44; Estimate 2: 294.4.
For both estimates the average delivered rates computed by the I-by-I algorithm is 276.8 kbps. This is because the algorithm considers prediction intervals spanning the playout duration (10 intervals), ignoring the bandwidth in the additional two intervals which are available for sending

the data. When the bandwidth in these prediction intervals is low, the algorithm overestimates the delivered rates, which leads to lossy transmission; when the available bandwidth in the additional prediction intervals (in the active period of the link) is high, the algorithm underestimates the delivered rates at the clients. Note that, in contrast the results from the L-by-L algorithm, discussed in detail in the next section, for the two cases are: 268.44 kbps and 294.4 kbps respectively.

7.6 Details of the L-by-L algorithm

In this section, we consider each step of the overall L-by-L algorithm presented in Figure 7.10. We discuss the algorithm for each step and illustrate the step using the topology presented in Figure 7.3.

7.6.1 Determining base stream rate for a link

In this step, our objective is to find a base stream rate which does not lead to loss of data due to under-utilization of link bandwidths, as explained in the example presented in Table 7.5. Note that for a given link, data is collected over its active period. However, the data is always consumed over the playout duration, which is constant for all clients. Thus, by dividing the total rate over the active period by the number of prediction intervals in the playout duration, we find the maximum stream rate that can be supported by the link, as given by Equation 7.4. This is used as an initial estimate of the base stream rate, *start rate*.

Consider a link l_i having $(P + d)$ prediction intervals over its active period. To ensure that the start rate can be sustained without loss across P, we use the steps presented below. Note that by reducing the predicted rate at each iteration, we find the start rate that even though underutilizes the link, ensures that the data can be sent without loss across all prediction intervals spanning the active period of the link.

```
compute start rate for link $l_i$;
For every prediction interval $j \in$ active period of $l_i$
find sent data capacity $S_i^j$, using start rate as the stream rate
find available data capacity $A_i^j$, using the predicted bandwidths
if $A_i^j > S_i^j$
```

```
Input:
linkid: index of link considered
start_point: Prediction interval from where the average needs to
be calculated
num_PI: number of prediction intervals considered
delta: number of prediction intervals for collecting additional data

% the following global variables are used by the algorithms %
pred_table: matrix with predicted bandwidths for
            the session duration
```

Figure 7.13: Input and global variables used in L-by-L algorithm

```
bandwidth predicted for interval $j$ = start rate;
recompute start rate;
$j = 1$;
end
end
base stream rate = start rate;
```
——————————————————————————————————-

The input and global variables that are common to the functions discussed in the next three sections are given in Figure 7.13.

Function *find_base_stream_rate* uses function *find_rate* that returns a scalar value of the start rate for a given link, calculated from the start point up to the number of prediction intervals given as input. Function *find_rate* takes the same input and global variables as function *find_base_stream_rate*. These functions are presented in Figures 7.14 and 7.15 respectively.

Illustration of algorithm ***find_base_stream_rate***

With reference to the predicted bandwidths in Table 7.1, let us consider Link 4 to illustrate this step. The δ value used for this link is 20 seconds, i.e. one prediction interval. Hence data that is

```
Additional global variable used:
TRANS_INT: Number of prediction intervals in the transmission
           duration.
Output:
stream_rate: base stream rate for a given link
the start_point upto the number of prediction intervals.

function find_base_stream_rate
             (link_id, start_point, num_pred_intervals, delta);
under_utilized == 1;
while under_utilized == 1
    available_data = 0; data_to_be_sent = 0; counter = 0;
    start_rate = find_rate
             (link_id, start_point, num_pred_intervals, delta);
    stream_rate = start_rate;
    for PI = 1: TRANS_INT
        counter = counter + 1;
        curr_pred_rate = pred_table(link_id, PI);
        available_data = available_data + curr_pred_rate;
        data_to_be_sent = data_to_be_sent + stream_rate;
        if available_data is less than or equal to data_to_be_sent
            if counter == TRANS_INT  when no under-utilized link
                under_utilized = 0;
            end
        else  when under-utilization of a link is detected
            pred_table(link_id, PI) = start_rate;
            break;
        end
    end
end
```

Figure 7.14: Algorithm: *find_base_stream_rate*

```
output: start_rate: scalar value of stream rate given the number of
        prediction intervals over which data is collected and
        consumed
find_rate (link_id, start_point, num_pred_intervals, delta)
counter  =  0;
cum_data  =  0;
for i  =  start_point: (start_point + num_pred_intervals  1)
    counter  =  counter + 1;
    curr_bw  =  pred_table(link_id, i);
    cum_data  =  cum_data + curr_bw;
end
num_intervals_considered  =  counter - delta;
start_rate  =  cum_data/num_intervals_considered;
```

Figure 7.15: Algorithm: *find_rate*

available over 6 prediction intervals are consumed over 5 intervals; the start rate computed is: 345.6 kbps.

For the first interval, $A_i^j = 384 * 20$ and $S_i^j = 345.6 * 20$. Link 4 would be under-utilized in the first interval itself. Hence, the predicted bandwidth of Link 4 is reduced to 345.6 kbps in the first prediction interval. The new start rate is computed as: 337.92 kbps. The process is repeated till predicted bandwidth of Link 4 is such that the buffer is empty at the end of the sixth prediction interval.

This process is repeated for all the prediction intervals spanning the session duration over which the link is active. For Link 4, when a start rate of 320 kbps is used (note that the predicted bandwidth on the link for the first and second intervals are reset to 320 kbps during computation), we find the condition for the base stream rate is satisfied. Thus, the base stream rate for Link 4 is 320 kbps.

Base stream rates computed for all the links for the example presented in Figure 7.3 over the session duration are: Link1: 473.6 kbps; Link2: 345.6 kbps; Link3: 208 kbps; Link4: 320 kbps; Link5: 256 kbps; Link6: 284 kbps.

7.6.2 Determining stream rates through links

For each link l_i, given a base stream rate, we need to verify that the base stream rate can be sustained by l_i across all prediction intervals over its active period. In this section, we discuss the algorithm *find_link_stream_rates* that finds the stream rates delivered across l_i without loss over the prediction intervals spanning its active period.

1. For each link l_i, we find the base stream rate using function *find_base_stream_rate* that ensures that the link is not under-utilized over any prediction interval. The current rate is set to the base stream rate. The variables *avail_data_vol*, *req_data_vol* and *break_point* are initialized to zero.

2. We need to check whether the current rate can be sustained by the link over all the prediction intervals spanning its active period without loss. When $A_i^j < D_i^j$ at a prediction interval j, the current rate can not be supported without loss. As discussed, we term the beginning of a prediction interval for which such a situation occurs as the *break_point*. When a break point is detected, the following steps are taken:

   (a) The stream rate is calculated up to the break point using function *find_rate*. This is the actual stream rate flowing through the link up to the break point.

   (b) Current rate is recalculated considering the available bandwidth over the remaining prediction intervals.

   (c) *start_point* is set to the next prediction interval; δ is set to zero; *pred_int_considered* is set to the appropriate value; *break_point* is set to one. The algorithm starts another iteration.

3. When the condition that $A_i^j >= D_i^j$ is satisfied up to the last prediction interval in the active period of the link, the current rate is assigned to stream rates of all the remaining prediction intervals and the loop terminates.

The algorithm is presented in Figure 7.16.

Illustration of determining stream rate through a link

We take link 1 in the example presented in Figure 7.3 to illustrate this step. Note that the constant multiplication factor of 20 seconds, the prediction interval duration, is not explicitly shown

```
Additional global variable used:
base_st_rates: vector of base stream rates as calculated by
the function find_base_stream_rates
Output:
st_rates: Vector of stream rates flowing through the link
function find_link_stream_rates(linkid, start_point, num_PI, delta);
last_PI  =  0; st_rates = base_st_rates(linkid, 1);
while last_PI  ==  0
    avail_data_vol = 0; req_data_vol = 0; counter = 0; break_point = 0;
    while break_point  ==  0
        For PI = 1: num_PI
            counter  =  counter + 1;
            link_bandwidth = pred_table[counter];
            avail_data_vol = avail_data_vol + link_bandwidth;
            if counter is greater than delta
               % after collecting data for delta %
                req_data_vol = req_data_vol + current_rate
                if req_data_vol is greater than avail_data_vol
                    % when the rate can not be sustained %
                   current_rate = find_rate(....);
                    for interval = (start_point + delta) to counter
                         st_rates[interval]  =  current_rate
                    end % find the rate for the intervals up to break-point %
                    if the last PI is reached
                       break_point = 1; last_PI = 1;
                    else start from the next PI with,
                       num_delta = 0; curr_rate = find_rate(..); break_point = 1;
                    end
                else
                    if last PI is reached
                        assign curr_rate to stream_rates for all intervals;
                    end
                    break_point = 1; last_PI = 1;
                end
            end
         end
     end
 end
```

Figure 7.16: Algorithm: *find_link_stream_rates*

PI	Predicted bandwidth	Available data capacity	Required data capacity	effective stream rate
1	384	384	0	0
2	384	768	473.6	469.3
3	384	1152	947.2	469.3
4	256	1408	1420.8	469.3
5	448	1856(448)	(480)	448
6	512	2368 (512)	(512)	512

Table 7.8: Stream rates through link 1

in Available data capacity and Required data capacity, as this factor cancels out in the calculation of the rate. Since link 1 is shared between all the three clients, minimum of their delay tolerance values (20 seconds) is the additional time available for sending data; i.e., the active period of link 1 has 6 prediction intervals. Available data capacity for 6 prediction intervals is calculated using predicted bandwidths from Table 7.8. This provides the maximum amount of data that can be delivered without loss over 5 prediction intervals spanning the playout duration $\mathcal{T}$. Hence the maximum stream rate supported by Link 1 is 473.6 kbps over 5 prediction intervals. Start rate is initialized to this value and algorithm *find_base_stream_rate* is invoked. Since there is no under-utilization of the link, 473.6 kbps is returned as the base stream rate by algorithm *find_base_stream_rate*. Steps in algorithm *find_link_stream_rates* are presented below.

```
Interpretation of values in Table 7.8 showing stream rates through link 1
---------------------------------------------------------------------
Base stream rate as determined by find_base_stream_rate is
473.6 kbps.

At PI 4, break-point detected as

required data capacity (1420.8) > available data capacity (1408);
Stream rate recomputed for first 3 PIs:
           = (1408/3)
           = 469.3 kbps
Note that all the accumulated data is consumed in this period;

The above steps are repeated with PI 5 as the starting point.
```

Link	Stream rates over prediction interval				
	1	2	3	4	5
Link1	469.3	469.3	469.3	448	512
Link2	345.6	345.6	345.6	345.6	345.6
Link3	208	208	208	208	208
Link4	320	320	320	320	320
Link5	256	256	256	256	256
Link6	284	284	284	284	284

Table 7.9: Stream rates through links over session duration

```
Base stream rate across remaining PIs spanning
the active period of the link
           = Total available data capacity / number of remaining PIs
           = (448+512)/2
           = 480 kbps

At PI 5, break-point detected as
required data capacity (480) > available data capacity (448);
Stream rate recomputed over only one prediction interval,
viz., PI 5, is 448 kbps.

Iterating one more time,
Base stream rate across remaining PIs spanning playout duration


           =  Total available data capacity / number of remaining PIs
           =  (512/1)
           =  512  kbps
Thus 512 kbps is chosen as the stream rate for the last PI
in the active period of Link 1.
------------------------------------------------------------------------
```

Similarly the stream rates are calculated for each link in the network. Note that for Link 5 which is only in the path of C_2, 40 seconds – two prediction intervals – is used in the computation of the stream rate.

Table 7.9 lists the stream rates over $\mathcal{T}$ computed for each link in the network.

```
% the following global variables are used by the algorithm %
linkinfo, pathinfo, clients, numlinks, numclients
Input:
stream_rates: matrix with stream rates flowing through each link over
each prediction interval spanning the playout duration.
Output:
delivered_rates: vector of stream rate delivered at the clients

for each client C
    for each prediction interval P
        % traverse path from source to client %
        for each link L in its path
            if first link in client's path
                min_rate =  stream_rates(C,P);
            else
                min_rate =  min(min_rate, stream_rates(C,P));
            end
        end
        delivered_rates [C, P] =  min_rate
    end
end
```

Figure 7.17: Algorithm: *find_delivered_rates*

7.6.3 Determining the loss-free delivered rate at clients

Given the stream rates that can flow through the links in the multicast tree, the next step is to find the delivered rates at the clients. The maximum amount of data that can flow into a client over each prediction interval spanning the playout duration depends on the maximum loss free rate supported by the links in its path over each of these prediction intervals. Hence, by finding the minimum of the stream rates that can flow through the links in each client's path over a prediction interval, we find the loss-free delivered rate at the client for that prediction interval. The algorithm for this step is presented in Figure 7.17.

Client id	Links in path	max. loss-free data capacity supported
1	L_1, L_2, L_4	1728*20
2	L_1, L_2, L_3, L_5	1088*20
3	L_1, L_2, L_3, L_6	1088*20

Table 7.10: Maximum loss-free data supported by links in clients' path

PI	Delivered rates		
	C1	C2	C3
1	320	208	208
2	320	208	208
3	320	208	208
4	320	208	208
5	320	208	208
Average	320	208	208

Table 7.11: Link-by-link algorithm: delivered rates

Illustration of determining delivered rates at clients

Given the predicted bandwidth as in Figure 7.3, the maximum available bandwidths on each link for transmitting data are:

Link 1: 2368*20; Link 2: 1728*20; Link 3: 1088*20;
Link 4: 1728*20; Link 5: 1280*20; Link 6: 1600*20

Now, we consider the links in the path of each client and find the amount of data that can flow into the client without loss. This is presented in Table 7.10. Note that the lowest rate supported by any link in a client's path over a prediction interval determines the upper bound for the delivered rate for that client over that prediction interval.
With reference to the example in Figure 7.3, the delivered rates at the clients are computed and presented in Table 7.11.

7.6.4 Delivered rates: L-by-L algorithm vs. optimization formulation

From our discussion thus far, it is clear the optimization function discussed in Section 7.1 and the L-by-L algorithm have the same objective; the same constraints are applied in both to find the loss-free delivered at the clients. For the example we considered for illustrating the solution approaches, we compare the results of the optimization function as presented in Table 7.3 and the L-by-L algorithm as presented in Table 7.11. The average delivered rates at clients C_1, C_2, and C_3 are: 345.6 kbps, 209 kbps, and 209 kbps using the optimization function as compared with 320 kbps, 208 kbps, and 208 kbps using the L-by-L algorithm. We present a detailed evaluation in Section 7.7.

The higher average using the optimization function results from the fact that the optimization function considers every possible valid value to find the maximum stream rate that can flow through the links over each prediction interval spanning the session duration. In the L-by-L algorithm, we have used a heuristic that starts with a constant stream rate across all prediction intervals; this stream rate is adjusted at each prediction interval to ensure loss-free playout over $\mathcal{T}$. Also, this algorithm considers discrete intervals for calculating the break point. While the L-by-L algorithm does not guarantee optimal delivered rates at the clients, as shown later, computation time of this algorithm renders it practical for finding loss-free delivered rates at clients in a multicast tree having hundreds of nodes.

Computation time of the algorithms

In this section, we estimate the computation time required for the algorithms and their order of complexity. As discussed in Section 7.2, the optimization function has exponential time complexity which renders it impractical for use with networks having hundreds of nodes.

Let there be L links, C clients in the multicast tree, and D be the number of levels. Let I be the number of prediction intervals in the session duration and P be the number of prediction intervals in the playout duration $\mathcal{T}$.

The link-by-link algorithm involves the following steps:

1. find base stream rate: This step involves O($I^2 \times L$) computations as algorithm *find_base_stream_rate* iterates over the prediction intervals spanning the active period of each link, till it finds the stream rate that is sustained by the link across all prediction intervals. Note that I is the upper bound on the number of prediction intervals in the active period of a link.

2. find the stream rate through each link for each prediction interval over its active period: This step involves finding the sustainable rate over the active period of the link; the number of computations is of O($I \times L$).

3. find the delivered rate at each client considering the links in its path. This involves $D \times C$ computations (which is $>$ L). This step is repeated for each prediction interval spanning the playout duration $\mathcal{T}$ and involves O($I \times L$) computations.

Thus, the total number of computations required for the link-by-link algorithm is: $O(LI^2)$. From the above discussion, we claim the following two advantages of the L-by-L algorithm over the optimization formulation:

1. By starting with the base stream rate that remains constant across the prediction intervals spanning $\mathcal{T}$, the search space to find the loss-free delivered rates is reduced. While the solution may not be optimal, it is close to optimal in cases where the maximum stream rate captures the fluctuations in the available bandwidth of links in the network.

2. Comparing the delivered rates at the clients using the optimization function to the L-by-L algorithm, while the average delivered rates are higher with the optimization function, the delivered rates at the clients vary significantly over the prediction intervals; this results in variation in the quality of reception at the clients. In contrast, our heuristic finds loss-free stream rates that do not vary much over the prediction intervals as we use discrete intervals. This is desirable from the point of view of a client.

In the next section, we analyze the performance of the link-by-link algorithm as compared with the optimization function.

7.7 Experimental evaluation of the L-by-L algorithm

In this section, we experimentally evaluate the performance of the link-by-link algorithm comparing it with the optimization function. (The reader is referred to Section 7.5.5 for a discussion on complexity comparison). We consider the following two parameters: (i) average delivered rates at the clients and (ii) computation time of the algorithms. As discussed, considering the exponential time complexity of the optimization formulation, we present results for a topology

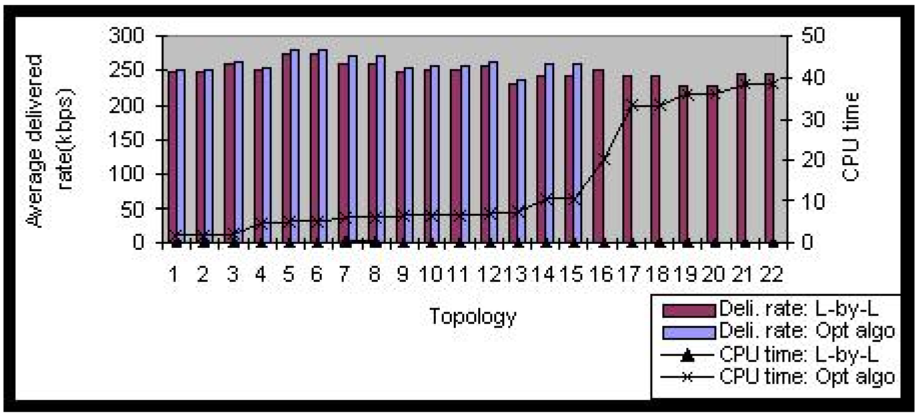


Figure 7.18: Comparison of Optimal and L-by-L algorithms

with 10 nodes and 4 clients as shown in Figure 7.18, selecting link bandwidths from a uniform distribution of bandwidth values which are multiples of 64 ranging from 144 kbps to 1024 kbps.

From Figure 7.18, we find that the average delivered rates computed by the L-by-L algorithm is less than that computed by the optimal algorithm on average by 2.64%. As seen from the graph even for a small topology having 10 nodes, for 7 out of 22 runs, (about 32%) the optimization function did not converge even after having taken about 1200 times the time taken by the L-by-L algorithm.

Having established that L-by-L algorithm finds the loss-free delivered rates close to the optimal rates, we now consider the impact of the following parameters on the performance of this algorithm:

- *Parameter1: Number of prediction intervals*: To study the impact of this parameter, we double the number of prediction intervals while keeping the predicted bandwidths the same for every two intervals; Note that we have doubled the prediction intervals, without changing the predicted bandwidth values over the session duration. We then increase the number of prediction intervals to 4 times the original number while keeping the predicted bandwidths same for every 4 intervals. Our objective is to study the impact of number of prediction intervals on (i) average delivered rates at the clients and (ii) CPU time taken by the algorithm to compute the delivered rates at the clients.

  We consider 20 topologies each having 100 nodes. In case 1 there are 7 prediction intervals. Each link is randomly assigned a bandwidth value. In case 2, we split each prediction interval into two, with each new prediction interval having the same bandwidth as the original prediction interval. In Case 3, we double the number of prediction intervals one more time; compared with case 1, now we have 4 prediction intervals for one prediction interval; however, the predicted bandwidth for all the 4 prediction intervals are the same as in the single prediction interval in Case 1. We calculated the average delivered rates at the clients for each case for the 20 topologies and observed that even though the number of prediction intervals has increased, as the predicted bandwidths over these split intervals did not change, the average delivered rates as determined by the algorithm remained almost the same.

The CPU time for the algorithm to determine the delivered rates at the clients, are compared for the three cases in Figure 7.19. Topology index is plotted along the X-axis. In order for the data points to be within range for comparison, we have plotted the CPU time in logarithmic scale on the Y-axis, sorted by the values of Case 1 in ascending order. Note that the values for Case 2 and Case 3 follow a similar pattern to Case 1. Reasoning for the increase in CPU time is as follows:

– L-by-L algorithm first finds the base stream rate. If the predicted bandwidth in an interval is such that the link is under-utilized, the algorithm iterates till the base stream rate is found. When the number of prediction intervals is doubled, the link is under-utilized over two prediction intervals, requiring base stream rate computations over two prediction intervals. Similarly, when the number of prediction intervals is increased by four times, base stream rate computation over four prediction intervals require iterations.

– Next step in the L-by-L algorithm is to find the link stream rates. This involves determining loss-free rates that can flow through a link over its active period. The dynamics of break point changes when the number of prediction intervals is increased. The extra iterations required, add to the CPU time. Also, note that these differences may manifest as small differences in the delivered rates at the clients.

• *Parameter 2: Fluctuations in bandwidth*: Here we examine the impact of small variations in bandwidth in two consecutive intervals while the total bandwidth across the two intervals remains the same. Let ϵ be the constant by which bandwidth is fluctuating. We consider 20 topologies having 100 nodes. In Case 1, there are 7 prediction intervals. In Case 2, each prediction interval is split into two intervals with same bandwidth. Let $b_1, b_2, \ldots, b_{14}$ be the bandwidths on a link over 14 prediction intervals. In Case 3, for each

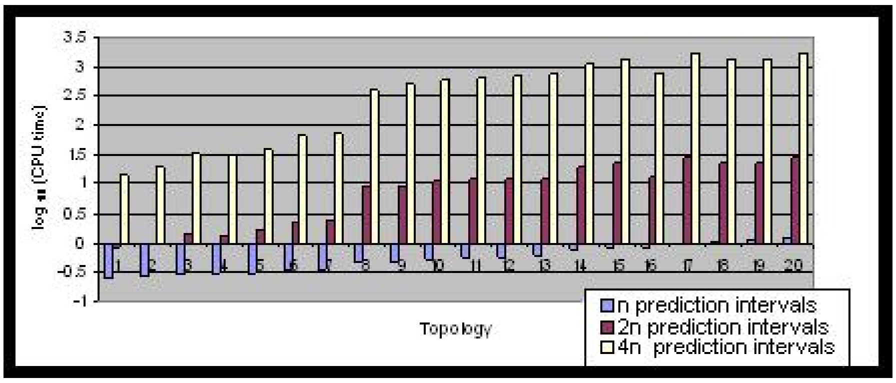


Figure 7.19: Effect of increasing number of prediction intervals on CPU time

pair of intervals, b_1, b_2, we assign: $b_1 = b_1 - \epsilon$ and $b_2 = b_2 + \epsilon$. In Case 4, for each pair of intervals, b_1, b_2, we assign: $b_1 = b_1 + \epsilon$ and $b_2 = b_2 - \epsilon$. We chose a value of 32 for ϵ for these experiments.

In Figure 7.20, average delivered rates for each topology is plotted. Again, we find the average delivered rates are almost the same, as the total bandwidth available over two prediction intervals in Case 3 and Case 4 are the same as the total bandwidth available over two prediction intervals in Case 2, and the bandwidth available over each prediction interval in Case 1.

Figure 7.21 depicts the effect of fluctuating bandwidth on CPU time. We have some interesting results here. When the number of prediction intervals is doubled as in Case 2, the CPU time increases. In Case 3, when we reduce the bandwidth in one interval and increase the bandwidth by the same amount in the next prediction interval, for every pair of prediction intervals, we find that the CPU time is almost the same as Case 2. However, when we reverse this pattern, adding to bandwidth and reducing bandwidth by the same amount for every pair of intervals, as in Case 4, we find that the algorithm converges very fast, performing more like Case 1. This again can be explained as in the previous case when the number of prediction intervals vary: Both algorithms *find_base_stream_rate* and *find_link_stream_rates* would converge faster in Case 4.

These experiments are helpful in determining the number of prediction intervals based on the knowledge of variability of bandwidth over the session duration.

Having established that the L-by-L algorithm effectively handles the varying bandwidth case, we explore a more realistic variation of this algorithm in the next section, motivated by the

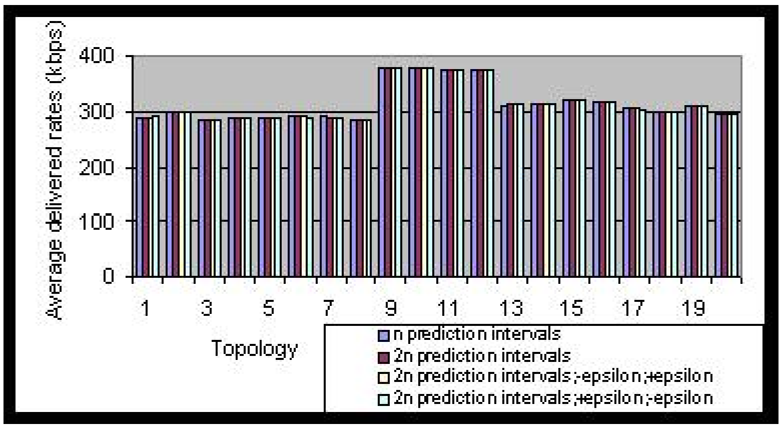


Figure 7.20: Effect of bandwidth fluctuation on average delivered rates

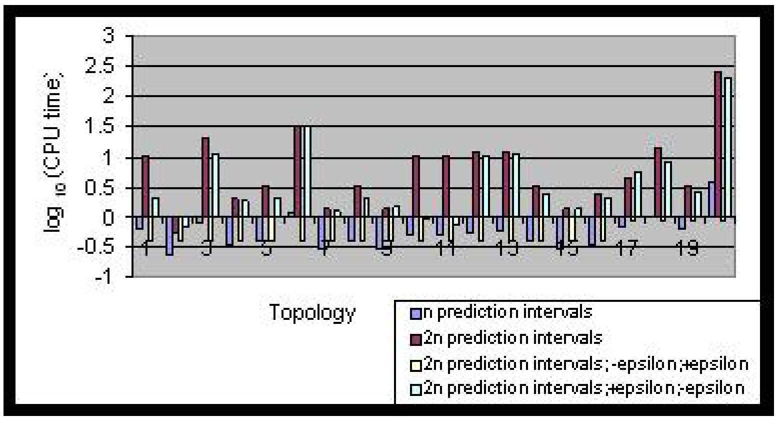


Figure 7.21: Effect of bandwidth fluctuation on CPU time

question: suppose we can get an updated estimate of the available bandwidth on the links for each prediction interval at the beginning of the prediction interval, how can we adjust the stream rate such that the bandwidth is maximally utilized while ensuring loss-free transmission?

7.8 Adjusting stream rates based on refined estimates of available bandwidth

Using algorithm *find_delivered_rates*, we find the delivered rates at the clients for each prediction interval. Note that we assumed the predicted bandwidths to be known apriori and that the prediction module accurately estimates the link bandwidths for the entire session duration. If there are j prediction intervals spanning the session duration, we assumed a model having *look-ahead of n prediction intervals*. We refer to this estimate of bandwidths as the *advance estimate*, where we assume that the bandwidth over each prediction interval = estimated bandwidth for that interval.

In this section, we relax this assumption and present an algorithm that adjusts the stream rate based on *look-ahead of 1 prediction interval* at the beginning of each prediction interval. Suppose at the beginning of every prediction interval, we receive an estimate for the bandwidths available on each link for the next prediction interval. Note that this estimate, termed *refined estimate*, could be the same, higher, or lower than the predicted bandwidth for a given prediction interval on a link as per the advance estimate. However, to ensure that the adjusted rate does not fall below the minimum required rate of clients, we assume that the revised estimate does not fluctuate more than a defined constant.
The following are the steps in this algorithm:

1. Let A_1 be the advance estimate of predicted bandwidths available at the beginning of the session. Algorithm *find_link_stream_rates* is used to find the stream rates supported by the links for each prediction interval.

2. For each link, the algorithm starts with the stream rate calculated in step 1. At the beginning of the second prediction interval, let F_1 be the refined estimate of the predicted bandwidths on the links.

3. Using the refined estimate, if the bandwidth available at a prediction interval (after sending any data in the buffer) is: (i) higher than the original available bandwidth using the

advance estimate, increase the stream rate by the difference, the upper bound on the stream rate being the base encoding rate. (ii) lower than the original available bandwidth using the advance estimate, decrease the stream rate by the difference.

This model is more realistic when the stream rate is adjusted based on the available bandwidth on a link, predicted for one prediction interval at a time at the beginning of that prediction interval. Note that we assume a feedback based bandwidth prediction mechanism such as the one used in [43]. We present the algorithm *find_adjusted_stream_rates* in Figure 7.22.
We illustrate the intuition behind the proposed algorithm through an example in the next section.

7.8.1 Example to illustrate algorithm *find_adjusted_stream_rates*

We use the same example as shown in Figure 7.3. The predicted bandwidths as per the static estimate are given in Table 7.1. Consider Link 1.

We find the stream rates through Link 1 using the advance estimate. As explained before, we find the stream rates that can flow this link over the playout duration as: 469.3 kbps in the first 3 prediction intervals and 448 and 512 kbps in the 4th and 5th prediction intervals of the playout duration.

The predicted bandwidth for the first prediction interval is taken to be the same as the advance estimate. Starting with second prediction interval, at the beginning of every interval a refined estimate is received for the available bandwidth on Link 1.

In every prediction interval, any data in the buffer is sent first; the remaining predicted bandwidth is available for sending data from that prediction interval.

- Using the advance estimate, the stream rates that can flow through the link are calculated using *find_link_stream_rates* at the beginning of the first prediction interval. No adjustment would be required in this prediction interval.

- Starting with the second interval, revised estimate is received at the beginning of every prediction interval. After sending any buffered data from the previous interval, the bandwidth available for sending data from that interval is calculated as: (predicted bandwidth - buffered data) for both the advance estimate and the revised estimate.

- The difference in bandwidth available for sending data from that interval with the two estimates is calculated. If this difference is negative, the pre-calculated stream rate is re-

```
Additional global variables used:
estimate1: static estimate of predicted bandwidths on the links
           for prediction intervals spanning the session duration
estimate2: refined estimate of available bandwidth on a given link
           over a given prediction interval.
Output:
st_rates: Vector with adjusted stream rates

function find_adjusted_stream_rates(linkid, start_point, num_PI, delta);
for each link L
    st_rates = find_link_stream_rates(linkid, start_point, num_PI, delta);
    data_buffered1 = 0;
    data_buffered2 = 0;
    for each prediction interval PI in the active period of the link
        stream_rate = st_rates[L, PI]; pred_bw1 = estimate1[L,PI];
        curr_stream_sent1 = (pred_bw1 - data_buffered1);
        data_sent1 = (data_buffered1 + curr_stream_sent1);
        data_buffered1 = (stream_rate - data_sent1);
        pred_bw2 = estimate2[L,PI];
        curr_stream_sent2 = (pred_bw2 - data_buffered2);
        data_sent2 = (data_buffered2 + curr_stream_sent2);
        data_buffered2 = (stream_rate - data_sent2);
        if current_stream_sent2 is less than current_stream_sent1
           deficit = (current_stream_sent1 - current_stream_sent2);
           stream_rate = stream_rate - deficit;
           st_rates[L, PI] = stream_rate;
           data_buffered2 = (stream_rate - data_sent2);
        elseif current_stream_sent2 is greater than current_stream_sent1
           if current_stream_sent2 is greater than BASEENCODINGRATE
              st_rates[L, PI] = BASEENCODINGRATE;
           else
              st_rates[L, PI] = (stream_rate + data_sent2);
           end
           data_buffered2 = 0;
        end
    end
end
```

Figure 7.22: Algorithm: *find_adjusted_stream_rates*

PI	Calc. st.rate	Estimate	Pred. b/w	avail.b/w in interval	Diff. b/w	Adjusted st. rate	Data sent	Data buff.
1	469.3	Advance	384	384			384	85.3
		Revised	384	384			384	85.3
2	469.3	Advance	384	298.7			298.7	170.6
		Revised	320	234.7	64 (-)	405.3	234.7	170.6
3	469.3	Advance	384	213.4			213.4	255.9
		Revised	256	85.4	128 (-)	341.3	85.4	255.9
4	448	Advance	256	0			0	448
		Revised	448	192	192 (+)	512	192	320
5	512	Advance	448	0			0	512
		Revised	448	128	128 (+)	512	128	384
6	0	Advance	512	512			512	0
a		Revised	384	384		384	384	0
b		Revised	448	448		384 (link under-utilized)	384	0
c		Revised	256	256		256 (buffer not cleared)	256	128

Table 7.12: Adjusting stream rates through link 1

duced by the difference; if the difference is positive, the stream rate is increased, bounded by the base encoding rate.

The dynamics of this example is presented in Table 7.12. The revised estimate of the last prediction interval can not be adjusted and hence based on the remaining data in the buffer, the stream rate for the last interval is decided. With reference to 7.12, we consider three cases for the sixth prediction interval. In case (a), the revised estimate is equal to the data in the buffer and the stream encoded at 384 kbps can flow through the link. In case (b), the revised estimate is higher than the data in the buffer. In this case the link is under-utilized. In case (c), the revised estimate 256 kbps falls short of the data in the buffer, 384 kbps. As discussed before, since the link bandwidths are varying, we use layer encoder as the resource to provide each client with its delivered rate. The last case is handled by the layer encoder such that layers which add up to bandwidth less than or equal to the revised estimate flows through the link.

Suppose the stream is encoded such that 128 kbps is the base layer and 128 kbps is the first enhancement layer, and 128 kbps is the second enhancement layer. In case (a) and case (b) all three layers flow through Link 1. In case (c) only the base layer and first enhancement layers flow through Link 1. Note that by appropriately designing the layers, loss-free transmission can be achieved across all prediction intervals spanning the playout duration. We discuss the layering mechanism that can be used in conjunction with the L-by-L algorithms in the next section.

7.9 Layering mechanism used

A layer encoder compresses multimedia data into one or more streams having different priorities. *Base layer* is the layer with highest priority that contains the most important footprint of the contents. Clients at the least need this layer to receive a meaningful playout. The layer encoder also generates additional layers called *enhancement layers* which can be used in conjunction with the base layer to refine the reception quality of the contents. There are two different approaches to adapt the encoding rate of the contents using layering: (i) receiver-driven algorithms and (ii) sender-driven algorithms. In the first approach, the source sends the base and enhancement layers as different streams through the network. Based on the availability of bandwidth, receivers initiate adding or pruning of layers. In the source-driven approach, the source sends the base layer and enhancement layers in one flow, with appropriate priority marking. With the help of network congestion control mechanisms (such as feedback from the network nodes) layers are dropped to serve the clients with appropriate rates. We refer the reader to Section 2.4.3 in Chapter 2 for a detailed discussion of the layering mechanisms. In this section we discuss the layering mechanism as it applies to our solution approaches.

7.9.1 Source-driven layering mechanism

In our solution approaches discussed so far, we assume a layer encoder at the source. When the predicted bandwidths over the session duration are known apriori, using our algorithm, the delivered rates at the clients over each prediction interval can be calculated. Note that the client delay tolerance values are used to find the stream rates that can flow through the links and the delivered rates at the clients. Source sends the stream starting in the first prediction interval for intervals spanning the playout duration. However, data flows through links over prediction intervals spanning the *active period* of the link using buffers. Hence it is not possible to dynamically change the rates at which layers are encoded. Based on the delivered rates at

Client	Delivered rates over prediction interval				
	1	2	3	4	5
C1	405.3	405.3	405.3	448	448
C2	384	384	384	384	384
C3	384	384	396.8	384	396.8

Table 7.13: Example to illustrate layering mechanism

the clients over the prediction intervals spanning its playout duration, the encoding rates for the base and enhancement layers are decided. Note that in our model the source is capable of dropping some layers or generating additional layers based on the delivered rates at clients over prediction intervals spanning the playout duration.

Illustration of the layering mechanism used

In our example, given the delivered rates at the clients as presented in Table 7.11, the following could be the choice of rates of the layers:
Base layer: 208 kbps
I enhancement layer: 112 kbps
Note that in this simple example, the source generates the base and the first enhancement layers for all the prediction intervals. Client C_1 gets the base and first enhancement layers for all the prediction intervals whereas clients C_2 and C_3 get only the base layer.
To understand the layering mechanism required to serve the clients maximally, we consider another set of predicted bandwidths for the same topology and determine the delivered rates at the clients which are presented in Table 7.13. In this case the following could be the choice for the layers:
Base layer: 384 kbps
I enhancement layer: 12.8 kbps
II enhancement layer: 8.5 kbps
III enhancement layer: 42.7 kbps

In the first 3 prediction intervals, source generates the base layer and the two enhancement layers. Client C_1 gets all the three layers whereas client C_2 receives only the base layer; client C_3 receives the base layer in the first two prediction intervals, receives the base and first enhancement layers in the third prediction interval. In prediction intervals 4 and 5, all four layers are generated at the source; C_1 gets all the layers in prediction intervals 4 and 5; C_2 gets the base layer; C_3 gets the base layer in prediction interval 4 and base and first enhancement layers in prediction interval 5. In this case, the source created an additional enhancement layer in PI 4; however, the base layer and other enhancement layers remained the same.

7.10 Conclusions

In this section, we presented solutions to find the delivered rates at the clients when the link bandwidths vary over the session duration. In a Closed User Group (CUG) model where clients subscribe to the service, a CSP has knowledge to predict the available bandwidths over the session duration. In such a model, we assume that the session duration is divided into prediction intervals of equal duration over which the bandwidths remain constant. We start our analysis with simple solutions, converting the problem to a static bandwidth model. An interesting simple solution is presented by the I-by-I algorithm which may be useful in practical scenarios where the client's link to the network is always the weakest link. We then propose a L-by-L algorithm where each link is considered and the loss-free delivered rates at the clients are determined. To capture the effect of small variations in available bandwidth, we present a solution that adjusts the stream rates flowing through the links at every prediction interval based on feedback from the network on the available bandwidth on each link at the beginning of every prediction interval. Using the layering mechanism and assuming that the feedback based prediction would be very close to the actual available bandwidth on the links, we can guarantee loss-free transmission to the clients.

Chapter 8

Solutions for on-demand streaming when link bandwidths are static

8.1 Introduction

In the previous chapters, we considered synchronous streaming, where streaming of a given content is prescheduled and starts at time t_0 for all clients. In this chapter, we consider on-demand streaming which has the following characteristics: A client C_i requests for contents at any time t_i specifying its requirements: a minimum acceptable rate γ_i^{min} and delay tolerance δ_i, time it is willing to wait for the transmission to start while staying connected. Streaming from the source $\mathcal{S}$ can start only when links in the path from $\mathcal{S}$ to C_i are free. Note that playout of the contents encoded at least at γ_i^{min} must start at time (t_i + δ_i), for C_i to be successfully serviced.

As we discussed in Chapter 1, we focus on a CSP's distribution network comprising of a proxy server serving as a source for clients over a multicast tree. When clients request for the same content over a period of time, the objective of the CSP would be to service as many clients as possible. Placement of streaming servers with caching capability at appropriate nodes in the network, to service future requests for the same contents, is a common technique used to achieve this.

With reference to Figure 1.2, the CSP's distribution network is typically highly provisioned. Let us consider a link l_i having bandwidth b_i in the distribution network. When $\mathcal{S}$ is streaming the contents encoded at a rate r_k, for a client C_i, maximum rate that flows through links in C_i's path in the distribution network is r_k. When $b_i >> r_k$, l_i is under-utilized. If another client C_j, which shares l_i with C_i, requests for the same content before C_i's transmission is over, it is not admitted if l_i is not free or if its minimum rate requirement can not be fulfilled when l_i becomes free. Note that by using all the available bandwidth on l_i, it can be freed at an earlier time such that C_j may also be serviced.

Based on this observation, we propose a Hybrid Streaming Mechanism (HSM) where a client's request triggers the selection of an intermediate node as a streaming point (SP) to which multimedia contents are dynamically transferred from $\mathcal{S}$, and this streaming point streams the contents to the client. Transferred contents are temporarily cached at the streaming point to service future requests for the same content. HSM helps a Content Service Provider's objective of satisfying as many client requests as possible and providing enhanced delivered rates at clients leveraging their delay tolerance.

In this chapter, we have included just a formal analysis of the problem and a solution for it. Simulation studies of the solution were conducted as part of the M. Tech. thesis of Mr. Annanda th. Rath and reported in [32].

Before we pose the questions that we address in this chapter, we present some definitions and assumptions in addition to the ones presented in Chapter 1.

8.1.1 Additional definitions and assumptions

Streaming server: A streaming server is capable of sending the stream at the exact rate at which it is encoded. We assume that the streaming server is also capable of transcoding a given stream to a given lower encoding rate.

Data transfer mechanism: Data transfer mechanism refers to the following mechanism which is assumed to be available at $\mathcal{S}$: multimedia data is packetized into chunks such that packets are transferred without any loss across links having high bandwidth from $\mathcal{S}$ to a node n_i.

Pure Streaming Mechanism (PSM): Mechanism where the only streaming server in the CSP's distribution network is placed at source $\mathcal{S}$.

Hybrid Streaming Mechanism (HSM): A mechanism that combines a data transfer mechanism up to a node n_k and streaming from n_k to a client C_i such that the stream is transmitted loss-free from $\mathcal{S}$ to C_i.

Streaming Point (SP): Streaming point refers to a chosen relay node at which the streaming server is placed. Streaming point is denoted by SP_i, where i is the index of the streaming point.

Region nodes: With reference to Figure 1.2 in Chapter 1, region nodes G_is are the relay nodes that are the edge nodes in the distribution network of the CSP. These nodes connect the distribution network to the access network.

For the analysis presented in this chapter, we assume that all client requests arrive at the region node serving that client. When the region node finds the requested contents in its own cache (if it is a streaming point) or at a upstream relay node, it triggers transmission to the client if the links to the client are free; else it transfers control to $\mathcal{S}$ for initiating appropriate action. Architecture of the nodes to support HSM is presented in Chapter 2.

Caching enabled streaming server: While sending out a stream encoded at a given rate, if a streaming server also stores the contents in a cache which has a fixed validity time, the streaming server is said to be caching enabled.

Time To Live of the Content (TTL): Time To Live of the Content (TTL) is the duration for which the contents in the cache is valid. TTL is initialized when a given content is streamed from the streaming server and updated every time the content is accessed from the cache. Simple strategies with minimal overhead can be chosen to define and update TTL.

Observation period: A time interval over which client request arrivals are monitored is referred to as the observation period.

Start of streaming session: For a given client C_i, the time t_i^s when transmission starts from $\mathcal{S}$ is the start of the streaming session.

Note that t_i^s is later than the start of connection t_i, when links in $p(C_i)$ are busy; When the links from $\mathcal{S}$ to C_i are free when C_i connects, t_i^s is equal to t_i. Also, t_i^s is earlier than the start of playout at C_i. Note that if t_i^s is greater than (t_i + δ_i), C_i can not be serviced as its requirement is violated.

The definitions related to connection, session, and playout from Chapter 2 are relevant to the discussion presented in this chapter.

We address the following questions in this chapter: In a streaming scenario where link bandwidths are static and client requirements are known, given client request arrivals for a specific content over an observation period,

- Where should the streaming server be placed to service the maximum number of clients?

- At what rates will the content be delivered to clients admitted for service over the observation period?
- How will the performance of the hybrid streaming mechanism compare with the pure streaming mechanism?

8.1.2 Solution approach

As discussed, when a streaming server is placed at $\mathcal{S}$, it sends out data at the encoded rate. Given a highly provisioned link l_i (having bandwidth greater than the base encoding rate Γ) from $\mathcal{S}$, when $\mathcal{S}$ streams the contents, the link is underutilized and occupied for the playout duration $\mathcal{T}$ of the stream. If l_i is shared by other clients requesting for the stream at a later time, the following possibilities occur based on the available bandwidth b_i on l_i:

- when b_i is much greater than Γ multiple streams can be simultaneously supported by l_i. Note that if more than one link are shared by the clients, all the links should have enough bandwidth to support multiple streams.
- when $b_i > \Gamma$ such that b_i is not big enough to support multiple streams, some of the available bandwidth is wasted. In such a case requests have to wait till the shared link is freed.

Thus, instead of streaming from $\mathcal{S}$, which allows the data to flow through the link encoded at the stream rate, a data transfer mechanism can be used; such a mechanism transports the contents to a network node closer to the client termed *streaming point*, utilizing the available bandwidth on the link effectively, freeing the link faster. Contents are streamed from the streaming point. This facilitates servicing more clients. We use this logic in developing an algorithm for servicing on-demand requests from clients for the same content over an observation period. By allowing temporary caching at the streaming point, we can increase the number of clients serviced when they request for the same contents in the future. Thus, the algorithm involves the following two steps:

1. Choosing an appropriate streaming point
2. Finding the number of clients serviced and the delivered rates at the clients

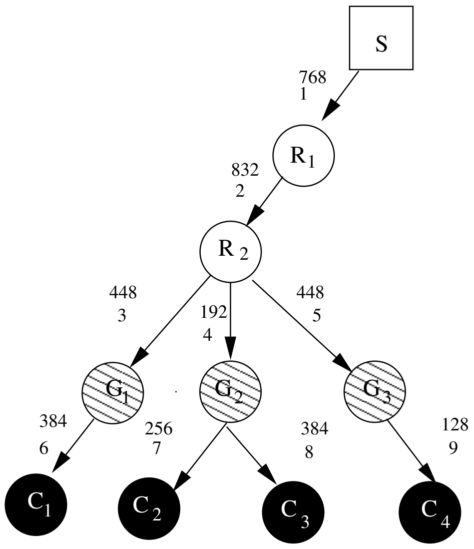


Figure 8.1: Example to illustrate HSM

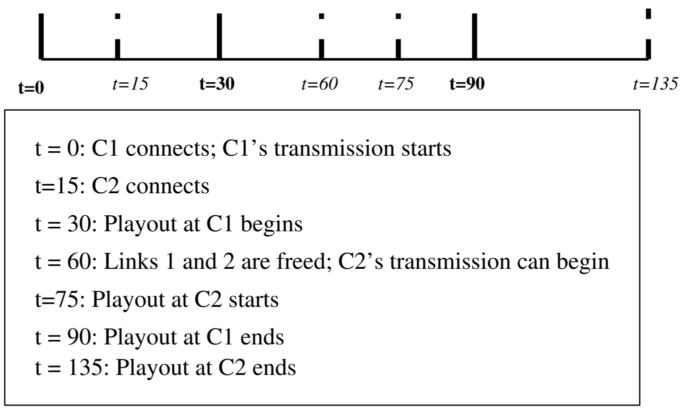
In our solution, we use streaming servers with transcoding capability as the resource. Before we discuss the solution approach, we present an example to illustrate the intuition behind the algorithm.

8.1.3 Example to illustrate the intuition behind the algorithm

Consider a simple multicast tree as shown in Figure 8.1. $\mathcal{S}$ is streaming contents to clients C_1, C_2, C_3, and C_4, connected through links 1 to 9, with bandwidths in kbps as indicated in Figure 8.1. R_1, R_2 are the relay nodes and G_1, G_2 are the region nodes in the path of clients $C_1, \ldots C_4$. The base encoding rate of the file is 512 kbps and the duration of the playout is 1 hour. Let C_1 request for the file at time $t = 0$. Let C_2 request for the same content at time $t = 15$. Let the delay tolerance values be $\delta_1 = 30$ minutes and $\delta_2 = 1$ hr. Let the minimum rate requirement for both the clients be 256 kbps. Our objective is to serve both the clients, C_1 and C_2. We consider the following scenarios to understand the intuition behind the algorithm:

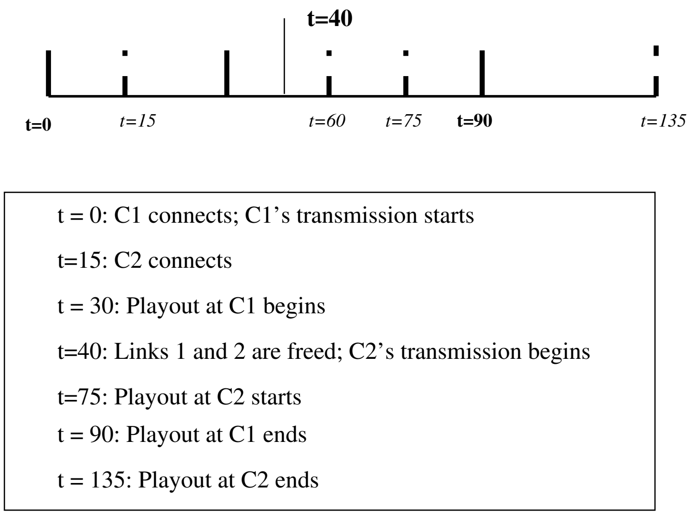
- *Case 1: Using PSM: Streaming server only at $\mathcal{S}$*: At $t = 0$, C_1 requests for the content. Weakest link in C_1's path is 384 kbps. Given that δ_1 is 30 minutes, maximum deliverable rate at C_1 is 512 kbps as calculated by Theorem 4.2. $\mathcal{S}$ streams the contents at 512 kbps. Links 1 and 2 are underutilized. These two links are occupied for 1 hr., the playout duration of the steam.

  C_2 requests for the same content at $t = 15$ minutes. Figure 8.2 illustrates the sequence



t = 0: C1 connects; C1's transmission starts
t=15: C2 connects
t = 30: Playout at C1 begins
t = 60: Links 1 and 2 are freed; C2's transmission can begin
t=75: Playout at C2 starts
t = 90: Playout at C1 ends
t = 135: Playout at C2 ends

Figure 8.2: Case 1: Streaming server at $\mathcal{S}$



t = 0: C1 connects; C1's transmission starts
t=15: C2 connects
t = 30: Playout at C1 begins
t=40: Links 1 and 2 are freed; C2's transmission begins
t=75: Playout at C2 starts
t = 90: Playout at C1 ends
t = 135: Playout at C2 ends

Figure 8.3: Case 2: Data server at $\mathcal{S}$ and streaming server at R_2

of events. Given that δ_2 is 60 minutes, C_2's playout starts at $t = 75$ minutes and ends at $t = 135$ minutes. Note that links 1 and 2 are busy till $t = 60$ minutes. Thus, C_2 effectively has a delay tolerance value of 15 minutes. Since the weakest link bandwidth in C_2's path is 192 kbps, maximum deliverable rate at C_2 is 240 kbps as calculated by Theorem 4.2, given the effective delay tolerance value of 15 minutes. Since C_2 needs a minimum of 256 kbps encoded stream, C_2's request can not be satisfied.

- *Case 2: Using HSM: Data server at $\mathcal{S}$ and streaming server at R_2*: In this case, the streaming server is located at R_2 and the content is sent across links 1 and 2 as data packets. Note that in this case, data can be transferred from $\mathcal{S}$ to R_2 at a data rate of 768 kbps. Let the base encoding rate of the file, Γ be 512 kbps and the duration of the playout, $\mathcal{T}$ be 2 hours. As stated in Chapter 2, we assume that the file size is a function of its encoded rate, and playout duration $\mathcal{T}$; thus, the maximum file size is given by: $\Gamma \times \mathcal{T} \times 3600$ kb. It takes 30 minutes to transfer the data encoded at 384 kbps to R_2 and 40 minutes if the data is encoded at 512 kbps. Since C_1's request arrived first, it is serviced at 512 kbps and links 1 and 2 are freed at $t = 40$. Figure 8.3 illustrates the sequence of events.

Given that δ_2 is 60 minutes, 35 minutes are available for C_2, when links 1 and 2 are free for transmission. The maximum deliverable rate at C_2 is 304 kbps as calculated by Theorem 4.2. Note that this rate is higher than C_2's minimum required rate. Hence both the clients can be serviced.

- *Using HSM with caching: Data server at $\mathcal{S}$ and streaming server at R_2 with caching capability*: We consider scenario 2; Let the contents be cached at R_2 till the Time To Live of the Contents (TTL = say, the playout duration). C_2 requests for the same contents at $t = 15$ minutes. Stream encoded at 512 kbps is cached at R_2 for 15 minutes (from $t = 0$ to $t = 15$). Hence C_2 can be serviced directly from R_2. Note that in this example given that δ_2 is 60 minutes, C_2, can be serviced at 384 kbps as calculated by Theorem 1. Hence, R_2 transcodes the stream from 512 kbps to 384 kbps to serve C_2.

Thus, by choosing an appropriate relay node for placement of the streaming server with a simple caching mechanism, client requests can be serviced efficiently.

8.2 Analysis of network properties

We consider a streaming scenario where link bandwidths are static and client requirements are known; client request arrivals for a specific content over an observation period are given. We need to address the following question: how does a CSP decide on a streaming server placement strategy to service maximum number of clients?

To answer this question, we analyze the properties of the network which are relevant in deciding the placement for streaming servers.

8.2.1 Expression for time to transfer data

Lemma 8.1

Consider $\mathcal{S}$ connected to node n through m consecutive links $l_1, l_2, \ldots\ l_m$. Let $b_1, b_2, \ldots\ b_m$, be the bandwidths available on these links, each of which is greater than the base encoding rate Γ The time required to transfer the file of size Z from $\mathcal{S}$ to node n at the end of link l_m, ignoring transmission and propagation delays, is given by:

$$T = Z/min(b_1, b_2, \ldots\ b_n) \tag{8.1}$$

Proof:
We use induction to derive equation 8.1.

- *Base case*: Let l_1 be a link connecting $\mathcal{S}$ to node n_1. Let b_1 be the bandwidth available on l_1, where $b_1 >> \Gamma$. T_1, the time to transfer file of size Z from $\mathcal{S}$ to n_1 is given by:
  $T_1 = Z/\ b_1$
  as trivially, $min(b_1) = b_1$.

- *Hypothesis*: Consider a path made up of links $l_1, l_2, \ldots\ l_m$ connecting $\mathcal{S}$ to node n having bandwidths $b_1, b_2, \ldots\ b_m$ which are greater than Γ. Let $min(b_1, b_2, \ldots\ b_m) = b_{min}$. Equation 8.1 holds for these consecutive links.

- *To prove*: We need to prove that Equation 8.1 also holds for links $l_1, l_2, \ldots\ l_{m+1}$ up to node $n + 1$ having bandwidths $b_1, b_2, \ldots\ b_{m+1}$.

  We have one additional link having bandwidth b_{m+1}. If b_{m+1} is greater than or equal to b_{min}, equation 8.1 is not affected. Let b_{m+1} is less than b_{min}. Suppose b_{min} is used as the rate to transfer the packets, data will be lost over link l_{m+1} as its bandwidth is not adequate to support the data rate. Hence, to guarantee loss-free transfer of the contents to node $n + 1$, $min(b_1, b_2, \ldots\ b_m) = b_{m+1}$ has to be chosen as the data transfer rate.

  Thus, when there are m consecutive links having bandwidths greater than Γ, the maximum data that can be transferred without any loss is bounded by $min(b_1, b_2, \ldots\ b_m)$. $\square$

Using equation 8.1, we can find the time required to transfer the contents across consecutive links having high bandwidths.

8.2.2 Options for streaming server placement

When a client C_i requests for contents, equation 8.1 is used to find the time it takes to transfer the contents from $\mathcal{S}$ to the streaming point in the path of C_i. With reference to the Figure 1.2 in

Chapter 1, we present an observation to decide on the placement of streaming server when the CSP's distribution network is highly provisioned.

Lemma 8.2

Given a client C_j requesting for a file encoded at Γ kbps at region node G_i. For all links l_i from $\mathcal{S}$ to region node G_i, if all $b_i >> \Gamma$placing streaming server at G_i is sufficient.

Proof:

Let C_j be a client having G_i in its path. Let $n_1, n_2, \ldots n_m$ be the nodes in the path from $\mathcal{S}$ to G_i. Let $l_1, l_2, \ldots l_{m+1}$ be the links in the path from $\mathcal{S}$ to G_i having bandwidths $b_1, b_2, \ldots b_{m+1}$. By equation 8.1, $min(b_1, b_2, \ldots b_{m+1})$ is the rate at which the data can be transferred from $\mathcal{S}$ to G_i. Any node in the path to G_i is a candidate for placement of the streaming server. However, given that that G_i serves multiple clients from that region, it is sufficient to place the streaming server at G_i. $\square$

Corollary 8.1

When all links in the distribution network are highly provisioned it is sufficient to place streaming servers at all the region nodes.

Proof:

Extending Lemma 8.2, when every link in the distribution network is $>> \Gamma$, contents can be transferred to the region nodes earlier by using the data transfer mechanism, compared to streaming from $\mathcal{S}$. In the latter case, all the links in the distribution network would be underutilized. Since all client requests are processed at the region nodes, trivially it is sufficient to place streaming servers at all the region nodes. $\square$

Now we consider the case when links in the distribution network are provisioned but there is no guarantee that all links in the distribution network are $>> \Gamma$. To develop thumb rules for the placement of streaming servers, we need an understanding of the occurrence of the weakest link in the path of a client, as the weakest link determines the maximum deliverable rate at a client, as determined by Theorem 4.2.

Lemma 8.3

Let b_{min} be the bandwidth of the weakest link in the path of a client C_j in the distribution network. Let r_j be the maximum deliverable stream rate at C_j calculated using Theorem 4.2. If r_j $<= b_{min}$, the weakest link in C_j's path occurs in the access network, when $\delta_j > 0$.

Proof:

By Theorem 4.2, if the weakest link in the path of C_j occurs in the distribution network b_{min} has to be $< r_j$ when $\delta_j > 0$. Hence the weakest link in C_j's path occurs in the access network. □

Lemma 8.4

Let b_{min} be the bandwidth of the weakest link in the path of a client C_k in the distribution network. Let r_k be the maximum deliverable stream rate at C_k calculated using Theorem 4.2. If $r_k > b_{min}$, the weakest link in C_k's path may occur in the distribution network, when $\delta_k > 0$.

Proof:

Let b_w be the weakest link in the path of C_j.

- Suppose $b_w < b_{min}$. This implies that the weakest link occurs in the access network as b_{min} is the weakest link in the distribution network. By Theorem 4.2, r_k depends on b_w and δ_k. For large values of δ_k, r_k can be $>= b_{min}$. In this case, when $r_k > b_{min}$, b_{min} is not the weakest link in C_k's path.

- Now we consider the case when $b_w = b_{min}$. By Theorem 4.2, $r_j > b_{min}$ when $\delta_j > 0$. In this case, when $r_k > b_{min}$, b_{min} *is* the weakest link in C_k's path. Hence if $r_j > b_{min}$, the weakest link in C_j's path *may occur* in the distribution network, when $\delta_j > 0$. □

Based on the above lemmas, we come up with the following thumb rules for the placement of streaming servers:

1. When $r_j <= b_{min}$, a stream encoded at r_j traverses links in the distribution network without introducing any delay. Hence any node in the distribution network can be a candidate for placement of streaming server. However, we need to choose a node that would serve maximum number of future requests for the same content (from the cache valid until TTL at the chosen node). The node having most outgoing links in the distribution network in the path of C_j, is a suitable candidate.

2. When $r_j > b_{min}$, b_{min} is a candidate for the weakest link in C_j's path. When the streaming server is placed at the node at the end of b_{min}, future requests for the same content can be served without incurring delays due to b_{min}, which *may be* the weakest link in these clients' paths.

In our algorithm, a simple cache management policy with very little overhead is used based on the client arrival patterns. TTL is initialized when a given content is streamed from the streaming server and updated every time the content is accessed from the cache.
We present algorithm *find_max_clients* which uses the lemmas and thumb rules discussed above to find the number of serviced clients when they request for the same contents over a given period of time.

8.3 Algorithm: *find_max_clients*

Algorithm *find_max_clients* takes the following parameters as input: (i) network topology with static link bandwidths, (ii) client requirements, and (iii) client request arrivals over an observation period. The following stream characteristics as defined in Chapter 1 are used: base encoding rate Γ and playout duration of the stream $\mathcal{T}$. The algorithm finds the number of serviced clients and the rates delivered to these clients. It invokes the appropriate streaming server option based on the thumb rules to maximize the utilization of available bandwidth in the distribution network to service the maximum number of clients. The schematic for the algorithm is presented in Figure 8.4. Note that the data structures and global variables as defined in Figure 4.10 are used in these algorithms also.
We present algorithms *place_streaming_server* and *find_max_clients* in Figures 8.5 and 8.6 respectively. While the algorithms deal with the basic case where all clients request for the same content, the algorithms can be extended for multiple contents by including an identifier for the content.

Performance analysis and results of simulations are presented in [32]. Performance of HSM depends on the following factors: (i) network topology and link bandwidths, and (ii) clients' requirements. We used 50 topologies termed as Group 1 having high link bandwidths from source to region node. The bandwidths are chosen randomly from the range (256 Kbps - 768 Kbps). The next 50 topologies termed as Group 2 had low bandwidth (weak links) in the distribution network, from source to region node. The bandwidths are chosen randomly from the range (128 Kbps - 256 Kbps). We observe that for Group 1 topologies HSM is a better scheme as the available bandwidth can be better utilized with this mechanism. For Group 2 topologies since links from the source to the region nodes have low bandwidths, using the data

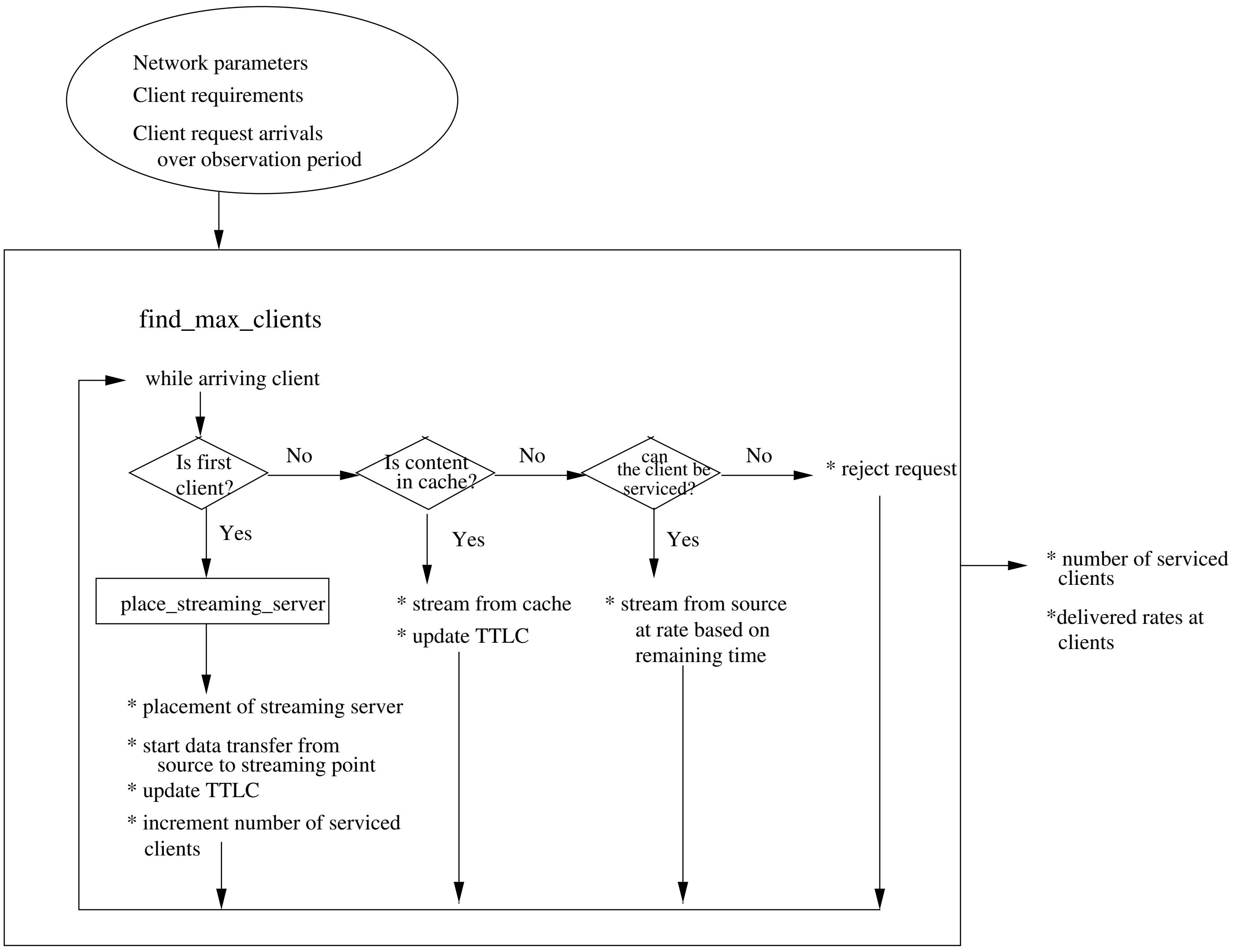

Network parameters
Client requirements
Client request arrivals over observation period
find_max_clients
while arriving client
Is first client?
No
Is content in cache?
No
can the client be serviced?
No
* reject request
Yes
place_streaming_server
Yes
* stream from cache
* update TTLC
Yes
* stream from source at rate based on remaining time
* placement of streaming server
* start data transfer from source to streaming point
* update TTLC
* increment number of serviced clients
* number of serviced clients
*delivered rates at clients

Figure 8.4: Schematic of *find_max_clients*

```
Input:
client_id: unique identifier for the first client in a subtree
            requesting for a given content

% the following global variables are used by the algorithm
linkinfo, pathinfo, numclients, numlinks, numnodes, clients,
BASEENCODINGRATE, TRANSDURATION

Output:
Streaming_point: id of the relay node chosen as streaming point
cli_deli_rate: delivered rate at the client

function place_streaming_server(client_id)
Calculate the maximum deliverable rate r_i for the given client C_i;
% Theorem 4.2 is used %
if r_i is less than r_min, the minimum rate required by C_i
    reject request
else
    if for each link in the path of C_i
        Find b_min, the bandwidth of weakest link in the
        client's path from source to region node
        if  r_i is less than or equal to b_min
            streaming_point = relay node with maximum number of
            outgoing links
        else
            streaming_point = relay node at the end of link with b_min
            as its bandwidth
         end
end
cli_deli_rate = r_i;
```

Figure 8.5: Algorithm: *place_streaming_server*

```
Input:
client_arrivals: client request arrivals over an observation period

Output:
num_serviced_clients: number of serviced clients
deli_rates: rates delivered at clients

function find_max_client (client_arrivals)
num_serviced_client = 0;
for each arriving client
    if this is the first client in that subtree
        place_streaming_server
        % returns the streaming server node and the
         rate delivered at the first client (refer to Figure 8.5) %
        deli_rates = maximum deliverable rate at client
          % calculated using Theorem 4.2 %
        Increment num_serviced_client;
    elseif the links are busy
        if content available in cache
            start streaming from the streaming server
            Increment num_serviced_client;
            update TTL;
            deli_rates = maximum deliverable rate at client
             % considering path of client from the streaming point %
        elseif constraints can be satisfied when the links become free
            deli_rates = maximum deliverable rate at client
             % considering the remaining delta of client,
               when the link becomes free %
               Increment num_serviced_client;
        else
            reject request
        end
    end
end
```

Figure 8.6: Algorithm: *find_max_clients*

transfer mechanism does not provide any advantage, as the time for transferring the content is the same even when the 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. We refer the reader to the publication [33] that resulted from this work for details.

8.4 Conclusions

In this chapter, we dealt with the on-demand requests that arrive over a specified period. A Content Service Provider (CSP) catering to the needs of clients dispersed around the world on a subscription based model, may receive requests for the same content over a period of time. Distance education is an example, where each lecture for a given course can be accessed by the students enrolled for the course over a specified period. By deploying steaming servers at appropriate nodes, on-demand requests for contents can be serviced efficiently. Note that the proposed algorithm ensures efficient utilization of available bandwidth in the CSP's distribution network in addition to leveraging the client's delay tolerance for enhancing the delivered rates at the clients.

Chapter 9

Buffer management issues: when a client device is memory constrained

9.1 Introduction

In the previous sections, we discussed the various algorithms for finding the delivered rates at clients and placement of resources to achieve those rates: (i) when the link bandwidths are static and (ii) when the link bandwidths are varying, but predicted over short intervals. Given that mobile devices are increasingly used as full-fledged computers running voice and video applications at the client end, as the concluding part of this thesis, we discuss the applicability of our algorithms to delay-tolerant applications where client devices are memory constrained.

In Chapter 2, we pointed out that in the node architectures required to support delay-tolerant multimedia applications *all nodes require buffers*. In all our analysis so far, we have assumed buffers at nodes to be unconstrained resources. Given the context of our work, while it is reasonable to assume that the nodes that are within the distribution network of the CSP would have enough memory to support required buffers, the assumption may not hold for client devices. In this chapter, we first look at the size of buffers required at the nodes and the factors that affect the size of buffers required at a client.

Recall that there are three types of nodes in the multicast tree: (i) Source, (ii) Relay, and (iii) Client. We first present a discussion on the size of buffer required at a relay node, considering the flow of data through the node. Since data flows into the source's buffers from its disks in a similar way, the same analysis holds for the source node also. Considering a client node, there are two factors that affect the size of buffer required at a client:

- *Bandwidth of the last link to the client*: depending on the bandwidth available at the last link to a client, buffers can be located at a relay node when the client device is memory-constrained.

- *Delay tolerance of the client*: delay tolerance of the client determines the start time for the playout at the client; thus, the amount of data to be buffered at a client and the time that the data remains in the buffer, both depend on the delay tolerance of the client.

We find that when the delay tolerance value goes beyond a threshold value, entire content may be buffered at a client. This is equivalent to *downloading* the content at the client. We analyze the implication of such a scenario, which can be leveraged to admit more clients by changing the schedule for the transmission.

9.2 Size of buffer required at a source/relay node

With reference to Figure 3.10 in Chapter 2, the source node buffers data when the bandwidth of a outgoing link is less than the encoded rate of the stream required to serve the clients having the link in their path. This is similar to buffering at a relay node as we explain below.

Consider a node n_i. Let l_i be the incoming link and l_j be the outgoing link of n_i, having bandwidths b_i^p and b_j^p where $p = 1, 2, \ldots P$, where P is the number of prediction intervals in the active period of l_i. P will be 1 in the case when the bandwidth is static over the active period of l_i. Note that size of buffer at n_i depends on the data that flows through l_i over its active period. Let $\mathcal{P}$ be the duration of each prediction interval.

At a given prediction interval the maximum amount of data that can flow into n_i is determined by the bandwidth available on l_i. The maximum amount of data that can flow out of n_i depends on the bandwidth available on l_j. Thus the maximum buffer size at n_i is given by the expression:

$$B_{ni}^{max} = \sum_{p \in (1,P)} max((b_i^p - b_j^p), 0) \times \mathcal{P} \tag{9.1}$$

Equation 9.1 also gives the size of buffer required at the source node. Note that the source encodes/transcodes the stream to the highest delivered rate for any client in each of its subtrees and sends the appropriately encoded stream through each of its outgoing link. With reference to Equation 9.1, for each stream sent out by the source, b_i^ps represent the encoded rates of

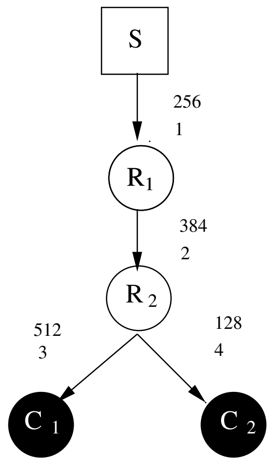


Figure 9.1: Example to illustrate use of buffers

the stream sent over each prediction interval and b_j^ps represent the bandwidths available on the outgoing link from the source over each prediction interval.

At a client node the buffer size depends on the arrival rate of the data and the delay tolerance of the client. As discussed, arriving data is buffered at the client till the start of the playout. Before we discuss the size of buffer required at a client node, we present a simple example to illustrate the flow of data and the use of buffers at the client nodes.

9.3 Buffer requirement at a client node: An illustrative example

Source $\mathcal{S}$ is streaming contents to clients C_1 and C_2 connected through links 1 to 4, with bandwidths in kbps as indicated in Figure 9.1. Let the base encoding rate of the contents, Γ, be 512 kbps and the duration of the playout, $\mathcal{T}$ be 1 hour. We assume the same minimum rate requirement for both clients: a minimum rate of 128 kbps. Let $\delta_1 = 1/2$ hr. and $\delta_2 = 1$ hr. Let us consider the buffers required for delivering optimal rates to C_1 and C_2.

- Applying algorithm *find_opt_rates_I*, the rates delivered at C_1 and C_2 are: $r_1 = 384$ kbps and $r_2 = 256$ kbps, respectively. Since the highest rate delivered at either of the clients is 384 kbps, $\mathcal{S}$, transcodes the stream to 384 kbps. It buffers 128 kbps and sends 256 kbps on l_1. Size of buffer required at $\mathcal{S}$ is 128×3600 kb.

- Considering C_1, its optimal delivered rate $r_1 = 384$ kbps. Playout at C_1 begins $1/2$ hr. after $\mathcal{S}$ starts streaming. Hence, stream encoded at 384 kbps needs to be buffered for $1/2$

hr. at C_1. Since links l_2 and l_3 have enough bandwidth to support the stream, instead of C_1 buffering (256×1800) kb of data, the data can be buffered at either of relay nodes R_1 or R_2. However, if the buffer is maintained at R_1 no data will flow to R_2 for $1/2$ hr. which will starve C_2 for this period. If the buffer is maintained at R_2, both the following objectives can be achieved: (i) no buffering is required at C_1; this is desirable when C_1 is memory constrained. (ii) data received at R_2 can be delivered at the appropriate rate to C_2 using a transcoder at R_2.

- Considering C_2, its delivered rate $r_2 = 256$ kbps. Note that the last link is the weakest link in this case. Hence buffering at R_2 does not help. In this case, memory required for buffering at $C_2 = 128 \times 3600$ kb.

From the above example we can infer the following:

1. When the last link to the client supports the optimal delivered rate at a client, buffers can be moved to an upstream relay node. For example, when the client is a memory constrained wireless device, it is necessary that the link from the base station has adequate bandwidth to support the optimal delivered rate. In such a case the buffers are managed at the base station.

2. When the last link is the weakest link in the client's path, it is necessary that adequate buffers are available at the client to support playout at the optimal rate. In the worst case, the entire content is buffered at the client. In the above example, even though C_2's last link is constrained, when $\delta_2 = 3$ hrs., the stream can be delivered at 512 kbps; however, $3/4^{th}$ of the content has to be buffered at C_2, as the playout begins only 3 hrs. after transmission begins.

According to Theorem 4.1, the maximum stream rate and hence the delivered rate at a client depends on the δ values of the clients sharing links with it. Let us consider the case when none of the shared links is the weakest link in any of the sharing clients paths; we refer to this scenario in the rest of this chapter as *no sharing link constraint*. When there is no sharing link constraint, given a high enough value of δ, any client should be able to receive the stream at the best possible quality, Γ, irrespective of the bandwidth of the weakest link in its path. When the last link in a client C_i's path has enough bandwidth to support its delivered rate, contents can be buffered at the previous node n_i in its path which may serve multiple clients. Note that if $r_i =$

Γ and n_i has transcoding capability, it can serve all the clients just in time without the overhead of buffers at every client.
We formalize these insights in the next section.

9.4 Observations related to buffer management at client nodes

As stated before, a Content Service Provider's (CSP) goal is to maximize the utilization of the available resources to best serve its clients. In practical scenarios, even when the relay nodes may not be buffer constrained, we may have a client device that has very limited capacity to buffer data.

When buffer at a given node is a constrained resource, it is prudent to utilize the available buffers at other nodes. This is especially true when memory constrained client devices are connected to nodes with large storage. In this section, we first examine some basic properties to find the buffer size required at a client, for a given δ value. We then analyze the effect of different δ value on the size of buffer required at the client.

9.4.1 Observations on buffer size at a client for a given δ value

Consider a client C_j having a delay tolerance value δ_j. Size of the buffer required at C_j depends on the bandwidth of the last link that terminates at C_j. We consider the following cases to understand the implication of buffer size for C_j:

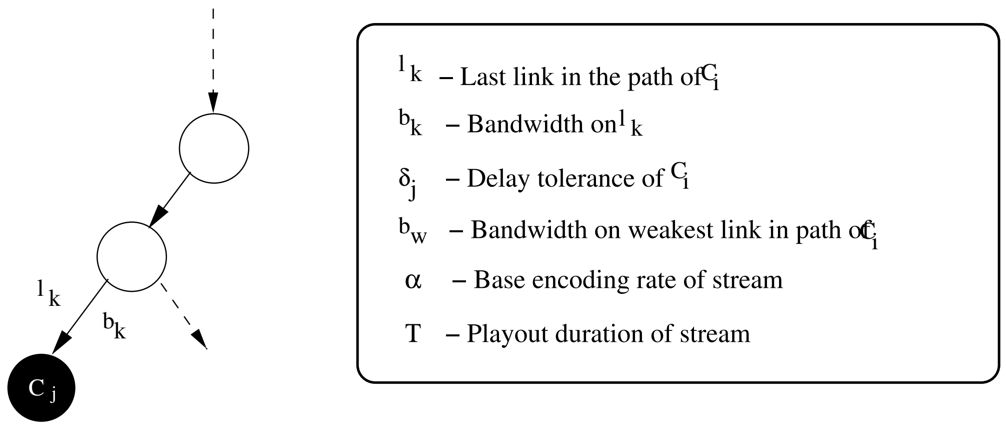
1. when the last link to the client is the weakest link and the bandwidth on this link is less than Γ.

2. when the last link to the client is the weakest link and the bandwidth on this link is greater than or equal to Γ.

3. when the last link to the client supports the delivered rate at the client.

4. when the last link does not support the delivered rate at the client.

Consider a client C_j having link l_k as the last link in its path and δ_j as its delay tolerance. Let b_k be the bandwidth of l_k and b_w be the bandwidth of the weakest link in C_j's path. As always, Γ is the base encoding rate and $\mathcal{T}$ is the playout duration of the stream.
We refer the reader to Figure 9.2 for all the derivations in this section.



l k
b k
C j
l k – Last link in the path of C i
b k – Bandwidth on l k
δ j – Delay tolerance of C i
b w – Bandwidth on weakest link in path of C i
α – Base encoding rate of stream
T – Playout duration of stream

Figure 9.2: Illustration used for derivations

Property 9.1:

Let $b_w < \Gamma$. If $b_w = b_k$, C_j must have a minimum buffer capacity of: $b_k \times \delta_j$ to buffer the data till the start of playout.

Proof:

When $\delta_j > 0$, r_j, delivered rate at C_j, is greater than b_w. Since $b_w = b_k$, r_j can not be supported by the last link l_k. Hence buffering is required at C_j till δ_j to provide loss-free playout at C_j. Since the bits flow across l_k at b_k kbps, a buffer of minimum capacity $b_k \times \delta_j$ is required at C_j. $\square$

Property 9.2:

With reference to Figure 9.2, consider client C_j. If $b_w >= \Gamma$, the stream encoded at Γ needs to be buffered at the client. The buffer size is given by: $\Gamma \times \delta_j$.

Proof:

When the bandwidth of the weakest link in the path of C_j is greater than or equal to Γ, the stream can flow across the path from $\mathcal{S}$ to C_j without any delay. In other words, the playout at Γ can immediately start at C_j. When $\delta_j > 0$, playout at C_j can begin only after δ_j. Hence it is necessary to buffer the data at the client. Since the data is buffered for δ_j and the stream is encoded at Γ the buffer size required is: $\Gamma \times \delta_j$. $\square$

Property 9.3:

With reference to Figure 9.2, consider client C_j. Let node n_{k-1} be the last node in its path. If $r_j <= b_k$, it is sufficient for node n_{k-1} to have a minimum buffer capacity given by: $(b_w \times \delta_j)$ to buffer the data till the start of playout.

Proof:

r_j, delivered rate at C_j, is less than b_k (i.e., $b_k > b_w$). Hence r_j can flow through l_k without introducing any delay. However, the playout at C_j begins only after δ_j. Hence the data is buffered at node n_{k-1} till δ_j when the stream is transmitted to C_j in real time. Since the data is flowing at b_w which is stored for δ_j, the size of the buffer required at $n_{k-1} = b_w \times \delta_j$. □

Property 9.4:

With reference to Figure 9.2, consider client C_j. Let node n_{k-1} be the last node in its path. If $r_j > b_k$, two cases arise: (i) l_k is the weakest link in C_j's path and (ii) l_k is not the weakest link in C_j's path. Let b_w be the bandwidth of the weakest link in C_js path. In either case, the buffer size required is: $b_w \times \delta_j$ and the buffer has to be located at the client.

Proof:

Since b_w is the weakest link in C_j's path and δ_j is the delay tolerance value, the extra data that can be delivered to C_j, that needs buffering is: $b_w \times \delta_j$. We will consider the two cases:

- Case (i): When l_k is the weakest link in C_j's path; $b_k = b_w$. To provide loss-free playout at C_j, data needs to flow at b_w. Hence the buffer has to be located at the client.
- Case (ii): When l_k is not the weakest link in C_js path; $b_k > b_w$. In this case, even though l_k can support b_k, the rate at which data is flowing is determined by b_w. Hence, the maximum rate at which data flows through l_k is b_w. Note that data needs to flow through l_k for ($\mathcal{T} + \delta_j$). Hence, buffer of size $b_w \times \delta_j$ is required at C_j. □

We summarize the above observations below:

- *Placement of buffer*:
  if $r_j <= b_k$, buffer can be placed at node n_{k-1}
  else buffer has to be placed at C_j
- *Buffer size required*:
  if $b_w > \Gamma$, buffer size = $\Gamma \times \delta_j$
  else buffer size $= b_w \times \delta_j$

In our discussion thus far, we considered the buffer size for a given δ value of a client. Since the buffer size depends on the value of δ, we now discuss the dynamics of buffers for different δ values. When there is no sharing link constraint, by increasing the δ value of a client, delivered rate at that client can be improved. However, beyond a certain δ value, when the delivered rate at the client is Γ, no further improvement in delivered rate can be achieved. We denote this δ value by: $\widehat{\delta_j}$. We explore the dynamics of buffers and its implication when the δ value is gradually increased to $\widehat{\delta_j}$ and beyond, in the next section.

9.4.2 Effect of δ value on buffer size

We consider the following:

- When there is no sharing link constraint, find the value of δ_j, $\widehat{\delta_j}$, for which C_j can have the stream delivered at Γ.
- What happens when $\delta_j > \widehat{\delta_j}$?
- What is the worst case buffer requirement for delivering stream encoded at Γ to C_j?

When the delivered rate at a client is not constrained by any other client's delay tolerance value, its delivered rate can be improved by increasing its delay tolerance value. However, increasing the clients δ value beyond a certain value, does not contribute to any improvement in its delivered rate, when its delivered rate is already equal to Γ. Playout at the client can start after δ; hence the client needs to buffer the data till the start of the playout. Suppose δ of a client is very large such that all contents are buffered at the client; in such a case there is no data flowing across the links when playout starts at the client. In other words, when all the data has been *downloaded* at a client, the links in the client's path may go into a dormant mode, where even though the streaming session is still on, there is no data flowing through the links. In this section, our objective is to understand this dynamics. Note that such an analysis is very important to make decisions on scheduling sessions such that all the resources are utilized for the maximum benefit of the CSP.

Property 9.5:
With reference to Figure 9.2, consider client C_j. None of the shared links in the path of C_j is the weakest link for any of the other sharing clients. Suppose C_j needs service at the best possible stream rate Γ, i.e., $r_j = \Gamma$. Let us denote the delay tolerance value of C_j that delivers stream encoded at Γ by $\widehat{\delta_j}$. $\widehat{\delta_j}$ is given by the expression:

$$\begin{aligned} \widehat{\delta_j} &= 0 && if\ b_w \geq \Gamma \\ &= \mathcal{T} * ((\Gamma/b_w) - 1) && otherwise; \end{aligned} \tag{9.2}$$

Proof:
This property follows from Theorem 4.2. When bandwidth of the weakest link in C_j's path is greater than or equal to Γ this rate can be delivered immediately, i.e., at zero delay tolerance. In order to deliver Γ to C_j when its weakest link bandwidth is less than Γ we derive the expression for $\widehat{\delta_j}$, from Equation 4.5, substituting Γ for the deliverable rate r_j and rearranging. □

Equation 9.2 gives the value $\widehat{\delta_j}$ that C_j needs to wait for if it needs the stream encoded at the best possible rate Γ given the weakest link bandwidth b_w in its path. $\widehat{\delta_j}$ defines the time when the playout starts at C_j. Note that at this time, optimal amount of buffers are used to service C_j with a stream encoded at Γ without any loss. For δ_j values greater than $\widehat{\delta_j}$, the rate can not be improved. But, more data is buffered as the playout can start only after the specified δ_j value. Thus, as δ_j value increases beyond $\widehat{\delta_j}$, the mechanism moves away from streaming toward downloading the content. At certain value of δ_j, the mechanism converges to a complete download. The next observation explores this transition.

Property 9.6:
With reference to Figure 9.2, consider client C_j. Suppose C_j needs service at the best possible stream rate Γ and $b_w < \Gamma$. By Property 9.5, $\widehat{\delta_j} = \mathcal{T} * ((\Gamma / b_w) - 1)$.
When C_j specifies delay tolerance δ_j where $\delta_j = \widehat{\delta_j} + \mathcal{T}$, the delivery mechanism converges to a complete download mechanism.
Proof:
When $b_w = \Gamma$, playout at C_j can start immediately as during $\mathcal{T}$, bits are pipelined to be played out. When $b_w < \Gamma$, $(\Gamma - b_w)$ bits are buffered over a period $\widehat{\delta_j}$ such that playout over $\mathcal{T}$ can be sustained without any loss. In other words, bits are flowing in while streaming is in progress between $\widehat{\delta_j}$ and $(\widehat{\delta_j} + \mathcal{T})$. When $\delta_j = \widehat{\delta_j} + \mathcal{T}$, all the data is buffered as the playout can start only after δ_j. Thus, at $\delta_j = \widehat{\delta_j} + \mathcal{T}$, the mechanism converges to a complete download of the content.
□

To summarize, we have the following claims from the analysis presented in this section:

- a positive δ value improves the delivered rate at a client when the bandwidth of the weakest link in its path is less than Γ.
- the nature of the last link in the path of a client as compared with the delivered rate at the client determines whether buffers can be located at the preceding node.
- $\widehat{\delta}_j$ defines the maximum value of δ_j for a client C_j when buffers are deployed to provide C_j with the best possible deliverable rate Γ. When δ_j takes values between $\widehat{\delta}_j$ and $(\widehat{\delta}_j + \mathcal{T})$, additional buffering takes place, with the mechanism moving from streaming to partial downloading to complete downloading at $(\widehat{\delta}_j + \mathcal{T})$.
- When δ_j takes values greater than $(\widehat{\delta}_j + \mathcal{T})$, it is equivalent to complete download and play back at a convenient time. In this case while part of δ_j is utilized to provide the client with the best possible rate, the entire content is stored at the client for a time $(\delta_j - (\widehat{\delta}_j + \mathcal{T}))$.

We present an example when the weakest link bandwidth in a client's path varies, to illustrate the following: (i) value of $\widehat{\delta}_j$, (ii) amount of data buffered, (iii) start and end time for the playout at the client, and (iv) amount of data buffered when $\delta_j = (\widehat{\delta}_j + \mathcal{T})$.

Example to illustrate observations

Let the base encoding rate of the contents, Γ, be 512 kbps and the duration of the playout, $\mathcal{T}$ be 1 hour. We consider a simple string topology with source $\mathcal{S}$ serving a single client C_j. For different values of the weakest link bandwidth, we find the value of delay tolerance $\widehat{\delta}_j$, for which C_j can get the stream delivered at Γusing Equation 9.2.

Table 9.1 illustrates for each case, the buffer requirement, start time for the playout, and end time of the playout. The last column in the table shows that when the client's $\delta_j = \widehat{\delta}_j + \mathcal{T}$, data is completely downloaded for each case.

9.5 Leveraging *residual delay tolerance*

From the discussion presented in the previous section, note that when many clients in a multicast session have δ values greater than their $\widehat{\delta}$ values, the original stream can be rescheduled without

Case	Weakest link b/w (b_w)	$\widehat{\delta}_j$ (hrs.)	Data in buffer	Playout start time	Playout end time	data in buffer if $\delta_j = \widehat{\delta}_j + \mathcal{T}$
1	128	3	$(3/4)^{th}$ content	$(t_0 + 3)$hrs.	$(t_0 + 4)$hrs.	512×3600
2	64	7	$(7/8)^{th}$ content	$(t_0 + 7)$hrs.	$(t_0 + 8)$hrs.	512×3600
3	256	1	(1/2) content	$(t_0 + 1)$hr.	$(t_0 + 2)$hrs.	512×3600
4	384	1/3	$(1/4)^{th}$ content	$(t_0 + 1/3)$hr.	$(t_0 + 4/3)$hrs.	512×3600

Table 9.1: Example to illustrate buffer requirements

compromising the delivered rates at these clients. We introduce the notion of residual delay tolerance and analyze admission control and scheduling issues in this section.
The question posed is: How can we optimize the use of buffers to deliver rates to clients such that more clients can be serviced over additional sessions, to maximize the overall benefit for the CSP?

Consider a client C_j. Let δ_j be the delay tolerance specified by C_j. Let $\delta_j > \widehat{\delta}_j$, the maximum delay tolerance value for buffer size when the stream is delivered to C_j at Γ. We term the difference $(\delta_j - \widehat{\delta}_j)$ as the *residual delay tolerance*, $\widehat{\delta}^R_j$.

Note that clients' positive $\widehat{\delta}^R$ values define the time over which data is stored in the buffers. During this time no data is flowing through the links in the clients' paths and no data is flowing out of the buffers. In other words, the buffers remain frozen for the period of $\widehat{\delta}^R$ which leads to inefficient use of the buffers. One way to exploit $\widehat{\delta}^R$ values of the client is to reschedule the streaming to start from the source at a later time.

We first consider the case where all clients in a subtree rooted at $\mathcal{S}$ have positive residual delay tolerance values. Let t_0 be the time at which a synchronous stream is scheduled. Let $C_1, C_2, \ldots C_n$ be the clients in a subtree Λ rooted at $\mathcal{S}$, having residual delay tolerance values $\widehat{\delta}^R_1, \widehat{\delta}^R_2, \ldots\ \widehat{\delta}^R_n$. Let $\widehat{\delta}^R_{min} = min(\ \widehat{\delta}^R_1, \widehat{\delta}^R_2, \ldots\ \widehat{\delta}^R_n)$.

Note that when every client has a positive $\widehat{\delta}^R$ value, the transmission can be postponed to $t_1 = (t_0 +\ \widehat{\delta}^R_{min})$ *without affecting the delivered rates at the clients.* A value $t_1 = (t_0 + \beta)$ can also be chosen where $\beta > \widehat{\delta}^R_{min}$ such that the minimum rate requirement of all the clients are met. Note that β defines the time by which the start of streaming can be postponed without violating any of the admitted client requirements. While $\widehat{\delta}^R_{min}$ and β define the bounds for the rescheduled start time for transmission, note that any value in between $\widehat{\delta}^R_{min}$ and β is valid. In order to choose an appropriate value for t_1, we propose a scheme based on pricing in the next section.

9.5.1 Rescheduling session: An illustration

A CSP has a business model based on the service it provides to its clients. While such a business model and issues related to pricing based on the quality of service provided to clients are beyond the scope of this thesis, to understand the issues related to scheduling the streaming sessions, we present a simple example. Suppose various price points for the enhanced rate at a client (above its minimum required rate) are defined by the CSP. Let p_{max} be the price paid by the clients when all of them receive the stream at Γ. Let p_{avg} be the average price paid by the clients.

Suppose the CSP has admitted m clients all having a positive $\widehat{\delta}^R$ value. We consider the two options: postpone the streaming session by (i) $\widehat{\delta}^R_{min} = min(\widehat{\delta}^R_1, \widehat{\delta}^R_2, \ldots, \widehat{\delta}^R_n)$, where all clients get the stream at Γ (ii) β, where all clients get at the least their minimum required rate γ_i^{min}. These options are explained below:

Option 1:

- Let $t_1 = (t_0 + \widehat{\delta}_{min})$; every client gets the stream at the best possible rate Γ. Thus, the revenue earned from existing clients is: $m \times p_{max}$.
- Based on the arrival schedule more clients may join the transmission in the additional time $\widehat{\delta}^R_{min}$. Let k be the number of additional clients joining the transmission. The revenue earned from the new clients is: $k \times p_{avg}$.
- Total revenue earned is given by the expression:
  $(m \times p_{max}) + (k \times p_{avg})$.

Option 2:

- Let $t_1 = (t_0 + \beta)$ such that β is the maximum time the streaming can be postponed without violating minimum rate requirements of any admitted clients.
- Revenue earned from existing clients is: $m \times p_{avg}$;
- Based on the arrival schedule more clients may join the transmission in the additional time β. Let l be the number of additional clients joining the transmission.
- Total revenue earned is given by the expression:
  $(m \times p_{avg}) + (l \times p_{avg})$.

Note that in the first option while all clients are provided with the best possible quality, there is less time to admit new clients as compared with the second option; Comparing the revenue, one of the options can be recommended.

In the case when only some clients have positive $\widehat{\delta}^R$ values, relationship between arrival distribution, price points, and start time of transmission have to be considered to decide whether rescheduling the transmission is beneficial.

9.6 Conclusions

In this chapter, we analyzed buffer sizes required to support delay tolerant multimedia applications. Given the bandwidth of the weakest link in a client's path, we found the value of delay tolerance that would deliver the stream at the base encoding rate at the client, when there is no shared link constraint imposed by other clients. We also found that beyond this value of δ, while the delivered rate can not be improved, the stream has to be stored at the client, as playout at the client can start only after δ. This property led us to define *residual delay tolerance*, the time when links in the path of a client are not utilized. We have suggested ways to leverage this residual delay tolerance to enhance revenues for the CSP through rescheduling the session. The insights gained from this chapter are very interesting from a business point of view; these ideas can be further developed to aid business analysis to understand the impact of costs vs. revenue.

Chapter 10

Conclusions and future work

In this chapter, we first summarize our contributions and then present a schematic for a tool, bringing together the algorithms presented in this thesis. Such a tool would aid a CSP to make decisions on resource allocation, deployment, service packages, and quality of service. We conclude with suggestions for future work in this area.

10.1 Summary of contributions

In this thesis, we have identified multimedia applications which we have termed *delay-tolerant applications*; in these applications, delivering the content with acceptable quality at the specified time is given more importance than delivering the contents as soon as possible. Given the nature of the contents in these applications (which are relevant for a period of time), it makes lots of sense to use the delay tolerance of clients to improve the quality of reception at the clients.

This idea is especially useful to the current CDN scenario, where bandwidth bottlenecks occur along the path to a client; even when the CSP provisions links in its distribution network, the *last-mile* problem persists.

Thus, our first contribution in this thesis is the idea of delay-tolerant applications. Having identified applications that fit the profile of delay-tolerant applications, we have explored various properties of such applications, given the context and parameters that affect such applications. We have also presented the architecture of nodes required to support such applications.

We identified three parameters that define the characteristics of the delay-tolerant applications: (i) *service type*: scheduled streaming, on-demand streaming, (ii) *bandwidth*: static, varying, and (iii) *resource used*: transcoders, layer encoders, streaming servers. Using combinations of these parameters, we identified four distinct problem areas. We then identified sub-problems under the first three problem areas, to define the scope of the issues we set to address in this thesis. We have not studied the fourth combination, (on-demand streaming, variable bandwidth) in this thesis. However, analysis and algorithms we have developed for the other three cases can be used as building blocks to solve the problems in this area.

In this thesis, we have also proposed the node architectures required to support delay-tolerant multimedia applications. In all our analysis, we have assumed buffers at nodes to be unconstrained resources. Given the context of our work, while it is reasonable to assume that the nodes that are within the distribution network of the CSP would have enough memory to support required buffers, the assumption may not hold for client devices. Considering a client node, there are two factors that affect the size of buffer required at a client: (i) *Bandwidth of the last link to the client*: depending on the bandwidth available at the last link to a client, buffers can be located at a relay node when the client device is memory-constrained. (ii) *Delay tolerance of the client*: delay tolerance of the client determines the start time for the playout at the client; thus, the amount of data to be buffered at a client and the time that the data remains in the buffer, both depend on the delay tolerance of the client.

When the last link to the client supports the optimal delivered rate at a client, buffers can be moved to an upstream relay node. For example, when the client is a memory constrained wireless device, it is necessary that the link from the base station has adequate bandwidth to support the optimal delivered rate. In such a case the buffers are managed at the base station.

We derived the expression for the delay tolerance value of a client required to deliver the stream encoded at the best possible rate to the client, given the bandwidth of the weakest link in its path. We analyzed the impact of delay tolerance on the buffer size at the client and found that beyond certain value of delay tolerance, data for the entire session is buffered at the client and the data remains stored at the client node till the start of playout. When the last link is the weakest link in the client's path, it is necessary that adequate buffers are available at the client to support playout at the optimal rate. In the worst case, the entire content is buffered at the client. If client device is memory constrained, such scenarios pose problems. We leave the in-depth analysis of delay-tolerant applications on wireless enabled client devices to future work.

We summarize the contributions made by this thesis below:

In a scheduled streaming application where link bandwidths remain stable over the session duration, our analysis and algorithms can be readily used by a CSP to make decisions on:

- *Quality of service provided to clients*: based on the availability of transcoders at relay nodes, optimal rates delivered to clients can be determined.

- *Number of transcoders required to provide the best quality of service across clients*: By eliminating redundancy in placement of transcoders, the optimal number of transcoders required for delivering best rates to clients can be determined.

- *Placement for a given number of transcoders*: With limited resources, maximize the quality of service across all clients.

While developing these algorithms, we have also come up with some interesting properties of the delay-tolerant applications. The concept of equi-deliverable rates is interesting in that, even when different values of δ are specified by clients in a subtree, if their maximum deliverable rates are the same, their delivered rates are also the same. In such a situation, only one transcoder is required at the root of the subtree serving the equi-deliverable rates clients. Even when link bandwidths vary over the session duration, the CSP can find the loss-free delivered rates at the clients and serve the clients with appropriate rates using a layering mechanism.

Our analysis for the on-demand streaming case when link bandwidths are static is relevant to many streaming solutions where popular contents are accessed on-demand. The idea of combining a data transfer mechanism with the streaming mechanism that maximizes the use of the provisioned bandwidth is extremely useful from a CSP's perspective to serve as many clients possible even in the presence of bandwidth constrained links in the network.

In the next section, we bring together all our algorithms, to develop schematic for a set of tools, to aid a CSP.

10.2 Schematic for CSP tools

We present the schematic in 10.1 having the following components:

- TOPRATES: finds the delivered rates at the clients for a given set of network parameters, client requirements, and resource placement.

- TOPPLACEMENT: Using TOPRATES to find the delivered rates at the clients for different resource placement, finds the best placement for a given set of resources.

- TOPREVENUE: Given a choice of schedules based on client requirements and price points for the service provided, finds the option that maximizes the revenue for the CSP.

  Using TOPRATES and TOPPLACEMENT, along with the investment specifications, finds option that maximizes revenue for CSP.

In this thesis, we started with the simple case when all link bandwidths are static to understand the impact of various parameters on the delivered rates at the clients; we also devised algorithms to find the resources required for providing best possible playout at the clients and algorithms for best placement of resources when the resources are limited. Then we relaxed the assumption of static bandwidths to include variability in bandwidth over prediction intervals spanning the session duration. We devised algorithms that would find the loss-free delivered rates at the clients when bandwidths are varying. We also presented a formal analysis of the on-demand streaming case when bandwidths are static.

In summary, we have developed and implemented the algorithms that can be used by TOPRATES and TOPPLACEMENT. We have also provided the analysis and outline for building TOPREVENUE. Since layering algorithms are readily available we have assumed that given the delivered rates at the clients and their minimum required rates, a layering module would generate the appropriate number of layers. We have developed algorithms for the various sub-problems identified and demonstrated their usefulness in aiding a CSP to administer delay-tolerant multimedia applications.

10.3 Future work

Our work has shown that multimedia applications with soft deadlines can be serviced efficiently by leveraging client delay tolerance. We have laid the foundation for research in this interesting area which holds lots of potential for disseminating high quality multimedia data even in the presence of bandwidth bottlenecks. Building a CSP tool, implementing a test-bed for evaluation of the proposed algorithms, refining algorithms to include random arrival of clients (where the topology may also change dynamically) are some of the interesting problems to be solved.

In the Video-on-Demand(VoD) scenario, the following problems pose interesting challenges: (i) handling new requests during a scheduled streaming session and (ii) handling multiple servers with independant or partially overlapping contents. We discuss these briefly below:

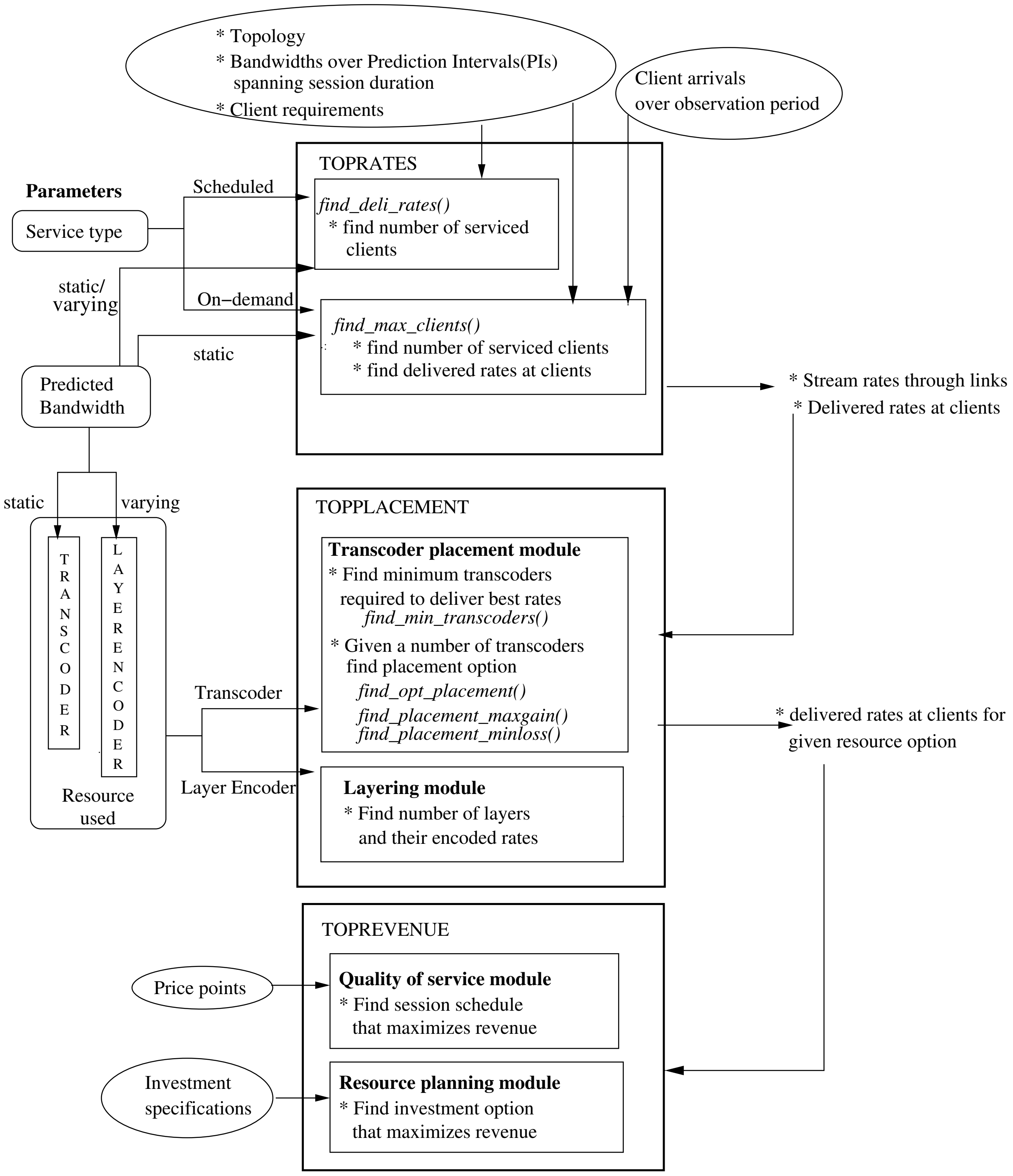

* Topology
* Bandwidths over Prediction Intervals(PIs) spanning session duration
* Client requirements
Client arrivals over observation period
TOPRATES
Parameters
Service type
Scheduled
find_deli_rates()
* find number of serviced clients
static/ varying
On−demand
find_max_clients()
* find number of serviced clients
* find delivered rates at clients
static
Predicted Bandwidth
* Stream rates through links
* Delivered rates at clients
static
varying
TOPPLACEMENT
TRANSCODER
LAYERENCODER
Transcoder placement module
* Find minimum transcoders required to deliver best rates
find_min_transcoders()
* Given a number of transcoders find placement option
find_opt_placement()
find_placement_maxgain()
find_placement_minloss()
Transcoder
* delivered rates at clients for given resource option
Layer Encoder
Layering module
* Find number of layers and their encoded rates
Resource used
TOPREVENUE
Price points
Quality of service module
* Find session schedule that maximizes revenue
Investment specifications
Resource planning module
* Find investment option that maximizes revenue

Figure 10.1: Schematic for tools to aid a CSP

1. In the existing literature, several techniques such as patching, stream merging etc.[14] are proposed to handle new client requests that arrive during the course of a scheduled streaming session. These techniques assume multiple channels from the source to get the initially missed part of the stream. Such techniques can be implemented in delay-tolerant applications also, if additional channels from the source to the client can be established.

2. In our on demand streaming model, we assume that caches *live* for a certain duration and hence can be used to serve clients with requests for the same content. Thus the caches play the role of a server, from where contents can be streamed to clients. Given this, we believe that the multiple server case can be reduced to the multiple caches case. In a similar vein, techniques used to retrieve content segmented and stored at multiple servers can be used with streaming points that cache partially overlapping contents.

The *on-demand streaming, varying bandwidths* case needs to be studied in depth. Extending our analysis for scheduled streaming when bandwidths are varying, to the Hybrid Streaming Mechanism, this case can be handled.

While we have analyzed buffer management issues when the client is device is memory - constrained, we have left the in-depth analysis of running delay-tolerant applications on wireless devices for future work. We introduced the notion of residual delay tolerance to determine whether the business model for the CSP can be improved by rescheduling the streaming session. This concept can help a CSP during the planning, scheduling, and admission control phases. We leave the in depth treatment of these issues dealing with the business model of the CSP to future research with the focus on business management issues.

In the declarative networking frameworks discussed in [11] [12], semantic data tagging is used to provide content level information to data streams flowing through a network. Such a framework can be leveraged to provide nodes with information on the content adaptation required in our delay-tolerant applications which assume in-built intelligence at every network node. Exploring the options and possibilities of extending such frameworks to facilitate the implementation of our proposed algorithms is another interesting angle for further research.

Bibliography

[1] C.C. Aggarwal, M.S. Squillante, J.L. Wolf, P.S. Yu, J. Sethuraman, Optimizing Profits in the Broadcast Delivery of Multimedia Products, *Fifth International workshop on Multimedia Information Systems*, October 1999.

[2] C. Albuquerque, B. J. Vickers, T. Suda, Source-adaptive Multilayered Multicast Algorithms for Real-time Video Distribution, *IEEE/ACM Transactions on Networking*, pp. 720-733, 2000.

[3] C. Albuquerque, B.J. Vickers, T. Suda, Multicast Flow Control with Explicit Rate Feedback for Adaptive Real-Time Video Services, *SPIE Performance and Control of Network Systems*, November 1998.

[4] Akamai: The business Internet
http://www.akamai.com/

[5] J.C. Bolot, T. Turletti, I.Wakeman, Scalable Feedback Control for Multicast Video Distribution in the Internet. *Proceedings of ACM SIGCOMM*, pages 58-67, August 1994.

[6] M. Castro, M. B. Jones, A-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, A. Wolman, An Evaluation of Scalable Application-level Multicast Built Using Peer-to-peer Overlays, *IEEE Infocom*, 2003.

[7] http://www.willas-array.com/prod/products/directory/pdf/vweb/ExplorerII.pdf

[8] Geography of Cyberspace Directory
http://www.cybergeography.org/

[9] A.Henig, D. Raz, Efficient Management of Transcoding and Multicasting Multimedia Streams, *Integrated Network Management, 9th IFIP/IEEE International Symposium on*, 2005.

[10] H. Kanakia, P. P. Mishra , A. Reibman, An Adaptive Congestion Control Scheme for Real-Time Packet Video Transport, *IEEE/ACM Transactions on Networking*, Vol.3, Issue 6, pp. 671-682, 1995.

[11] S. B. Kodeswaran, A. Joshi, Content and Context Aware Networking Using Semantic Tagging, *International Workshop on Semantics Enabled Networks and Services*, pp.77, 2006.

[12] S. B. Kodeswaran, O. V. Ratsimor, F. Perich, A. Joshi, Utilizing Semantic Tags for Policy Based Networking, *IEEE Globecom 2007, Internet Protocol Symposium*, pp. 1954-1958, 2007.

[13] P. Krishnan, D. Raz, Y. Shavitt, The Cache Location Problem, *IEEE/ACM Transactions on Networking*, Vol. 8, No. 5, October 2000.

[14] S. Krithivasan, Mechanisms for Effective and Efficient Dissemination of Multimedia, *Technical report*, September 2004.
www.it.iitb.ac.in/s̃aras/Papers

[15] S. Krithivasan, S. Iyer, To Beam or to Stream: Satellite-based vs. Streaming-based Infrastructure for Distance Education, *Edmedia*, June 2004.

[16] S. Krithivasan, S.Iyer, Enhancing Quality of Service by Exploiting Delay Tolerance in Multimedia Applications, *ACM Multimedia*, Nov. 2005.

[17] S. Krithivasan, S. Iyer, Strategies for Efficient Streaming in Delay-tolerant Multimedia Applications, *Proceedings of the Eighth IEEE International Symposium on Multimedia*, pp. 419-426, December 2006.

[18] J. Kurose, K. Ross, Computer Networking: A Top Down Approach Featuring the Internet, *Addison Wesley*, 2003.

[19] http://en.wikipedia.org/wiki/Last_mile

[20] T.V. Lakshman, P.P. Mishra, K.K. Ramakrishnan, Transporting Compressed Video over ATM Networks with Explicit Rate Feedback Control, *Proceedings of IEEE Infocom*, 1997.

[21] J. Liu, B. Li, Adaptive Video Multicast over the Internet, *IEEE Multimedia*, January-March 2003.

[22] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma , S. Lim, A Survey and Comparison of Peer-to-Peer Overlay Network Schemes, *IEEE communications survey and tutorial*, March 2004.

[23] A. Mahanti, On-Demand Media Streaming on the Internet: Trends and Issues, *Comprehensive Examination Paper, Department of Computer Science, College of Arts and Science, University of Saskatchewan*, December 2001.

[24] http://www.mathtools.net/MATLAB/index.html

[25] S. McCanne, V. Jacobson, M. Vetterli, Receiver-Driven Layered Multicast, *Proceedings of ACM SIGCOMM*, pp. 117-130, August 1996.

[26] The MPEG Homepage
http://www.chiariglione.org/mpeg/

[27] L. Patil, Video Transmission Over Varying Bandwidth Links, *M.Tech. Project, Department of Computer Science, IIT Bombay*, 2006.

[28] P. Paul, S. V. Raghavan, Survey of Multicast Routing Algorithms and Protocols, *Proceedings of the 15th International Conference on Computer Communication*, pp. 902-926, 2002.

[29] S. Paul, X. Li, M. Ammar, Layered Video Multicast with Retransmissions (LVMR): Evaluation of Hierarchical Rate Control, *Proceedings of IEEE Infocom*, April 1998.

[30] A. Pentland, R. Fletcher, A. Hasson, DakNet: Rethinking Connectivity in Developing Nations, *Computer*, vol.37, no.1, pp.4-9, January 2004.

[31] M. Ramalho, Intra- and Inter-Domain Multicast Routing Protocols: A Survey and Taxonomy, IEEE communications, *Surveys and Tutorials*, 2000.
http://www.comsoc.org/livepubs/surveys/public/1q00issue/ramalho.html

[32] A. Th. Rath, HSM: A Hybrid Streaming Mechanism for Delay-Tolerant Multimedia Applications, *M.Tech. Project, Department of Computer Science, IIT Bombay*, 2006.

[33] A. Th. Rath, S. Krithivasan, S. Iyer, HSM: A Hybrid Streaming Mechanism for Delay-tolerant Multimedia Applications, *MoMM 2006*, December 2006.

[34] www.mobilitypr.com/clients/files/RGB_DBM_press_release_06112007.doc

[35] RTP, Real-time Transport Protocol
www.cs.columbia.edu/ hgs/rtp

[36] RTSP, Real-time Streaming Protocol
www.cs.columbia.edu/ hgs/rtsp

[37] SearchNetworking.com
http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci1252357,00.html

[38] Y. Shang, M. P.J. Fromherz, T. Hogg, Complexity of Continuous, 3-SAT-like Constraint Satisfaction Problems, *IJCAI-01 Workshop on Stochastic Search Algorithms*, Aug. 2001.

[39] B. Shen, S-J Lee, S. Basu, Streaming Media Caching with Transcoding-Enabled Proxies, *Proceedings of the 6th IASTED International Conference on Internet and Multimedia Systems and Applications*, August 2002.

[40] B. Shen, S-J. Lee, Transcoding-enabled Caching Proxy for Video Delivery in Heterogeneous Network Environments, *Proceedings of Internet and Multimedia Systems and Applications*, pp. 360-365, 2002.

[41] B. Shen, S-J Lee, S. Basu, Caching Strategies in Transcoding-Enabled Proxy Systems for Streaming Media Distribution Networks, *Proceedings of IEEE Transactions on multimedia*, vol.6, no.2, April 2004.

[42] B. Shen, and S. Roy, A Very Fast Video Special Resolution Reduction Transcoder, *Proceedings of International Conference on Acoustics Speech and Signal Processing (ICASSP)*, May 2002.

[43] D. Sisalem, H. Schulzrinne, The Loss-Delay Based Adjustment Algorithm: A TCP-friendly Adaptation Scheme, *NOSSDAV*, July 1998.

[44] http://en.wikipedia.org/wiki/Tree_data_structure

[45] X. Tang, J. Xu, Replica Placement for QoS-Aware Content Distribution, *IEEE INFOCOM*, 2004.

[46] B. Vandalore, W. Feng, R. Jain, and S. Fahmy, A Survey of Application Layer Techniques for Adaptive Streaming of Multimedia, *Journal of Real-time systems*, 2000.

[47] B. J. Vickers, M. Lee, T. Suda., Feedback Control Mechanisms for Real-Time Multipoint Video Services, *IEEE Journal on Selected Areas in Communications*, vol.15, no.3, April 1997.

[48] B.J. Vickers, C. Albuquerque, T. Suda, Adaptive Multicast of Multi-Layered Video: Rate-Based and Credit-Based Approaches, *Proceedings of IEEE Infocom*, April 1998.

[49] L. Visciano, J. Crowcroft, TCP-like Congestion Control for Layered Multicast Data Transfer, *Proceedings of IEEE Infocom*, April 1998.

[50] J. Wang, A Survey of Web Caching Schemes for the Internet, *Cornell Network Research Group*, 2001.

[51] X. Wang, and H. Schulzrinne, Comparison of Adaptive Internet Multimedia Applications, *Invited paper, Special issue on distributed processing for controlling telecommunications systems*, June 1999.

[52] F. Warthman, Delay-Tolerant Networks(DTNs), A Tutorial,*DTN Research Group Internet Draft*, March 2003.
www.ipnsig.org/reports/DTN_Tutorial11.pdf