Multimedia Streaming Rate Optimization in Peer-to-peer Network

By

Gm Tareq Hossain

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

December, 2011

Nashville, Tennessee

Approved:

Prof. Yi Cui

Prof. Yuan Xue

Prof. Aniruddha Gokhale

Prof. William H. Robinson

Prof. Kenneth R. Pence

To my father AKM Rafiqul Islam and mother Umme Hani Chowdhurani

ACKNOWLEDGEMENTS

I would like to thank my advisor Prof. Yi Cui for his support, advice, and for giving me the freedom and opportunity to work on this exciting and challenging project. I would also like to thank Prof. Xue, who was always willing to spend time on technical discussions during the course of this project. I express my sincere gratitude to the other members of my dissertation committee - Profs. Gokhale, Robinson, and Pence - for providing with suggestions to refine my research ideas. I would like to thank Prof. Pence for his invaluable input while writing my dissertation and during the preparation of my defense.

I am forever grateful to Prof. Ashish Khisti for helping me with the theoretical aspect of my thesis. During the course of our collaboration, Ashish has become a friend to me and he was always available to verify the theoretical foundations of my thesis. I would also like to thank Ashish for hosting me as a Visiting Researcher at the University of Toronto. Discussions with him was invaluable and has formed the backbone of my thesis.

Lastly, my ultimate gratitude goes to my family for providing me with their never-ending love and support. I cannot express in words, the sacrifices my parents have made for my education. I would like to acknowledge my siblings - Showkat, Johora, and Awlad - for their encouragement. I especially would like to acknowledge my brother Showkat for guiding me throughout my academic career. Needless to say, I would not be the person I am today without his tireless support and guidance. I would like to thank Jannath for believing in me, for her constant encouragement and emotional support through good-times and bad. It has been an incredible journey with her and I am looking forward to the beginning of a new chapter in my life.

Tareq Hossain
July 27, 2011
Nashville, TN

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

CHAPTER I

INTRODUCTION

Optimization has become a powerful tool in the design and implementation of computer and engineering systems. Many problems in real-world applications in these fields arise due to various constraints inherent in a system consisting of many distinct local components working towards a shared global objective.

Consider a system comprised of a set of components and a set of resources that are available to each of the component. Each component is only responsible for a small subset of the overall system objective and depends on a subset of resources allocated to it. The resources, however, can only provide services to a limited number of components at any given time. Furthermore, the components naturally interact with neighboring components only. The overall system objective in this case is to optimally allocate these limited resources among the set of components so as to maximize the global objective.

Finding a solution to this resource-optimization problem in a distributed environment, in the absence of a central coordinator, is one of the oldest and well-studied problem. The distributed and resource constrained nature of the Internet generally encompasses this optimization problem.

A Motivation

The recent advancement in compression techniques and networking technologies have resulted in wide deployment of novel content distribution applications. These applications enable the end-users to have ubiquitous access to media streaming services such as live broadcasting, video-on-demand, and video conferencing.

Since the emergence of Napster in 1999, Peer-to-peer (P2P) networks have experienced explosive growths, and P2P-based applications have become the most dominant form of

Internet traffic [1]. This rapid expansion of P2P is further enhanced by the popular P2P-based BitTorrent file sharing network. The growth of P2P traffic is expected to continue along with the rapid increase in Internet connectivity and the development of popular end-user based applications such as pplive, sopcast, justin.tv, that take advantage of P2P's cost-effective implementation and instant deployability. The application scenario considered in this thesis is P2P multicast.

In P2P multicast, a media stream is disseminated to a large number of peers over the Internet. Participants contribute their uplink bandwidth and other resources to relay media streams to neighboring peers in the network. Compared to content delivery network (CDN), this type of distributed system is cost-effective due to its lack of dedicated infrastructure and is scalable to the number of users due to resource-sharing.

However, the clients that connect to the media streams are becoming more and more heterogeneous and are connecting through a wide variety of access medium. These days, typical devices used for media consumptions can range from mobile devices with low processing power and small display sizes to high performance workstations with high-definition (HD) displays. Furthermore, users are connected to the Internet through access medium such as Wi-Fi, 3G, 4G, Ethernet, etc. In the case of media stream, the best effort nature of the Internet does not provide acceptable *Quality of Service* (QoS) guarantees that are required for multimedia streaming. Therefore, in order to ensure QoS, it is essential to optimize the the available network resources.

B Problem Statement

The ubiquitous access to high-speed connectivity to the Internet for the end users has resulted in a rapid growth in media-related traffic. However, the underlying infrastructure of the Internet that was built during the 1990s and the first half of the last decade was for services such as e-mails and bulk data transfers. The traffic associated with these applications are sensitive to losses but not to delay and therefore, are largely insensitive to bandwidth

limitations. First proposed by Jacobson, et al. [2], TCP is the most widely deployed transport protocol of today's Internet. It was introduced during the early days of Internet with the goal of ensuring reliable data transport for traffic that are insensitive to delay. TCP provides end-to-end QoS support through its AIMD-based (Additive Increase/Multiplicative Decrease) mechanism and is mainly responsible for the remarkable stability of the Internet despite its rapid growth. While the congestion-control mechanism of TCP is appropriate for bulk data transfers, it is not naturally suitable for media traffic. Its rapidly-varying congestion penalty on data-rate is extreme to the strict delay requirements of streaming media applications [3].

The QoS for media streams is extremely delay sensitive because of the time constraint imposed by end users' perceived quality of experience. Since the generated video quality is directly related to the video bit-rate during the encoding process at the source, the QoS of the media stream is also sensitive to the availability of network bandwidth. The dominance of TCP means that any congestion-control protocol designed for media streaming must coexist with TCP and be *TCP friendly* [4] because imposing unfair competition to TCP traffic could lead to possible congestion collapse [5]. TCP friendliness means that under resource constraint, media-data will experience the same delay and network bandwidth limitations as TCP. However, being TCP-friendly also means that in the presence of constraint media-data will experience severe data-loss. Such loss of data may result from dropped packets when the underlying transport protocols (e.g. UDP) does not guarantee TCP-like QoS against data losses. Data is also considered lost because of delayed arrivals in the presence of congestion.

Current media technologies achieve TCP-friendliness through delay-loss tolerance. This is done by introducing redundancies into the media stream during the content generation process. However, the ability to handle data loss meant that many video streaming services rely on TCP-based protocol (e.g. browser based HTTP used in YouTube) to deliver media data. The continued dominance of TCP at the transport layer and the

continuous increase in media traffic means that along with TCP-friendly media protocol, resource-optimization at the application-layer is necessary to reduce cost and increase perceived quality of experience by end-users.

P2P has become the most popular means for media distribution because it provides an application-layer platform that can abstract the underlying heterogeneity at the transport and the physical-layer [6]. In addition, deploying a P2P system is extremely cost-effective due to the resource-sharing by the participating peer machines. However, due to this resource-sharing and the network heterogeneity, available bandwidth of peers is unavoidably constrained and fluctuates. Peer churning (arbitrary joining and leaving by peers to and from a network) further exacerbates this problem. Furthermore, unlike in a client-server system, data is often relayed via several peers, resulting in additional delays, especially in the presence of network congestion and/or bandwidth constraint. Therefore, P2P systems must implement proper resource-allocation techniques to optimally utilize available bandwidth.

Bandwidth is the most demanding resource in a P2P system. In the context of streaming media, optimization can be classified into two broad categories: local-rate optimization between two communicating peers and global-rate optimization across all peers in the network. Global-rate optimization for all peers depend on the type of encoding used to generate the media itself. This thesis develops optimization solutions for media streaming across both categories.

C Thesis Contributions

The contribution in this thesis can be divided into the following categories:

- In **local-rate optimization** between two communicating peers, a MEdia-friendly congestion-aware real-Time Streaming (METS) protocol is proposed that can be used to transport media data between two peers. Unlike existing congestion-aware

protocol that uses regular feedback for rate-control, METS integrate feedback data with packet-retransmission or rate-control messages.

- In **continuous-stream** encoding, media data is encoded in such a way that the perceived quality of the decoded video by the end-user is a continuous and monotonically increasing function of the streaming rate. Framework used to globally optimize this type of media stream is referred to as the *convex optimization framework.* A new optimization algorithm under this framework is proposed for video streaming in P2P networks that simultaneously considers uplink capacity of a peer and the number of cumulative of a peers.

- In **scalable-stream** encoding, media data is encoded in a such a way that the quality function is a discontinuous stair-case type function of the streaming rate and is not amenable to convex optimization techniques. Existing optimization solutions involve heuristic-based algorithm. This thesis proposes a new heuristic-based solution to scalable-stream optimization. On the sender-side, the algorithm focuses on load-balancing and priority-based video stream to a set of receivers. On the receiver-side, video requests are made to maximize the received video data.

- Finally, this thesis also proposes a new **message-passing framework** for optimization of scalable video stream. Advantage of this simple but elegant approach over other heuristic-based approach is that the optimization algorithm itself is independent of the underlying constraints. The algorithm iteratively updates resource allocation decision based on a given set of codewords. The codewords are binary representation of various network and video constraints. Therefore, any number of constraints can be used to generate a set of codewords without modifying the algorithm. To the best of our knowledge, this is the first work that systematically addresses the problem of scalable-stream optimization.

D Thesis Outline

The thesis is organized as follows.

- **Chapter II:** This chapter provides a background on optimization. First, the network topology that is used through-out the thesis is presented. Background on media streaming protocols, optimization for continuous-stream and scalable-stream is also presented.

- **Chapter III:** In this chapter, the congestion-aware streaming protocol for real-time media is presented. The algorithm improves on the data-loss rate by implementing a recovery mechanism to address lost or delayed data.

- **Chapter IV:** In this chapter, the optimization for continuous stream video for P2P networks is presented. Simulation shows that this new algorithm improves the performance over existing algorithm.

- **Chapter V:** Heuristic algorithm for optimization of scalable video stream in P2P network is presented. The algorithm optimizes the rate across the network by considering available bandwidth of peer for topology construction, load-balancing, and data-request from child peers to parent peers in the network.

- **Chapter VI:** In this chapter, the message-passing based optimization framework for scalable video stream is presented. It is based on existing works in iterative decoding in the field of Information Theory. Iterative decoding refers to the process of iterative update of outgoing data based on all incoming data.

- **Chapter VII:** This chapter concludes the thesis and provides an outline of future work.

CHAPTER II

BACKGROUND

The availability of high-speed Internet connectivity and the powerful computing devices have resulted in a growing demand for multimedia communication services such as video-on-demand, video conferencing, or live video streaming. This growing demand is evidenced by the increasing popularity of online media services such as YouTube, Hulu, Joost, etc. However, ubiquitous access to these services are still not guaranteed due to the network infrastructure deficiencies. Therefore, any solution to cope with such lack of QoS require rate adaptation of the data stream.

In this chapter, a brief introduction is provided to the network topology and network model that is used in this thesis. The issue of heterogeneity and the need for resource allocation is discussed. The use of video distortion minimization as an optimization objective is discussed in the context of rate-adaptation. Finally, an overview of various optimization algorithms and protocols are presented.

A Network Model

Existing approaches for P2P streaming can be divided into two classes: *tree-based* and *mesh-based*. In a tree-based topology, peers are connected to a single parent, while peers are connected to multiple parents in a mesh-based topology. The tree-based approach extends the idea of end-system multicast [7]. A mesh-based P2P streaming is derived from file swarming mechanisms (such as BitTorrent), where participating peers form a randomly connected mesh. Due to the multiple incoming connections for each peer, a multi-path mesh can fully utilize the network resources of its peers. Furthermore, peers experience a higher degree of stability [8] in a mesh-based approach compared to a tree-based approach. This results in higher quality for the delivered video. However, this higher

quality and stability compared to tree-based approach comes with extra computational/data overhead required to coordinate data reception from multiple parents.

In this thesis, only a cycle-free network is considered. A cyclic mesh structure is common in P2P applications, mostly in downloading applications, where data relaying is coordinated in receiver-driven mode on a piece-by-piece basis, e.g., BitTorrent. Many P2P TV applications, albeit supporting video streaming, function in the same manner. Most importantly, these applications are not concerned with video adaptation issues because all peers receive exactly the same content. However, the target applications in this thesis is designed towards streaming scenarios in which rate adaptation can help, such as the presence of network heterogeneity (streaming to hand-held devices) or strong real-time requirement overshadowing quality (multi-party video conferencing). It is not uncommon to see a DAG (Directed Acyclic Graph) distribution structure in these scenarios. For example, in a live broadcasting event, if a new peer always find its parent(s) among the current online peers, a DAG structure will form where the new peer can be sequenced according to its joining time.

A.1 Model Description

Consider a P2P network consisting of H end hosts. The set of hosts is denoted as $\mathcal{H} = \{h_i \,|\, i = 0,1,\ldots,H\}$. Define a flow f that directs from a peer h_j to peer h in a cycle-free network by the relation $h_j \xrightarrow{f} h$, where $h(f) = h$ is the peer that f directs to. In this connection, peer h_j is the parent of peer h (similarly, peer h is the child of peer h_j). Consequently, $\mathcal{F}$ is defined as the set of flows $\mathcal{F} = \{\, f_i \,|\, i = 0,1,\ldots,F \,\}$. For peer h, $\mathcal{F}(h)$ is the union of the set of incoming flows $\mathcal{F}_i(h) = \{\, f_j \,|\, \forall j, h(f_j) = h \,\}$ and the outgoing flows $\mathcal{F}_o(h) = \{\, f_j \,|\, \forall j, h \xrightarrow{f_j} h_j \,\}$. Here $\mathcal{F}_i(h) = \emptyset$ for server peers and $\mathcal{F}_i(h) = \emptyset$ for leaf peers. This follows that the set of parent peers of h is $\mathcal{H}_i(h) = \{\, h_j \,|\, \forall f \in \mathcal{F}_i(h), h_j \xrightarrow{f} h \,\}$ and the set of child peers is $\mathcal{H}_o(h) = \{\, h_j \,|\, \forall f \in \mathcal{F}_o(h), h(f) = h_j \,\}$. The flows f in the network takes place on a unicast path that connects two peers and encompass a set of physical links $\mathcal{N}$

on the Internet such that $\mathcal{N}(f) \subseteq \mathcal{N}$ is the set of links encompassed by f. Based on this definition, the flow-set of each link n is defined as $\mathcal{F}(n) = \{f \in \mathcal{F} \mid n \in \mathcal{N}(f)\}$ (i.e., the set of flows that pass through n). Therefore, the total receiving rate of a peer h is:

$$x_h = \sum_{f \in \mathcal{F}_i(h)} x_f \tag{1}$$

The maximum streaming rate for a flow depends on the uplink capacity c_n of each of the links traversed along the flow path (i.e., the smallest uplink capacity along the path of f determines the maximum achievable streaming rate for f). However, the physical topology considered in this thesis is based on the assumption that the bottleneck link of an end-to-end connection only happens at the uplink of the sending peer [9]. As such, the maximum streaming rate of a sending peer h depends only on its own uplink capacity, c_h. This effectively reduces the link set $\mathcal{N}$ to contain only the uplink of peers and the server. Consequently, the uplink capacity of the peers in the network are collected into a capacity vector $z = (z_h, h \in \mathcal{H})$. As illustrated in Fig. 1, this topology is termed as the *star topology*. The notations used in this section are collected in Tab. 1

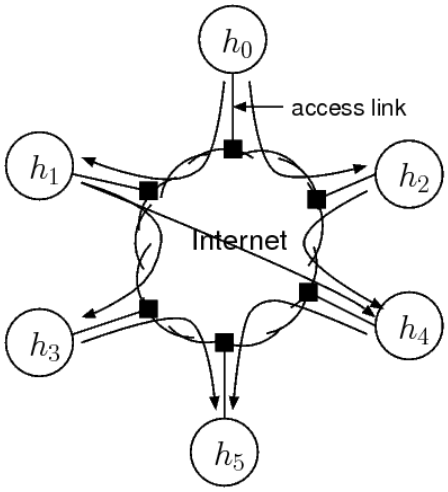


Figure 1: P2P Streaming Illustration: Star topology

Notation	Definition
$h \in \mathcal{H}$	End host or peer
$f \in \mathcal{F}$	Unicast flow in overlay multicast
$h(f) = h_i$	Peer h_i that flow f directs to
$\mathcal{F}_i(h)$, $\mathcal{F}_o(h)$	Set on incoming and outgoing flows of peer h
$\mathcal{H}_i(h)$, $\mathcal{H}_o(h)$	Set on parent and child peers of peer h
$\mathcal{N}(f) \subseteq \mathcal{N}$	Set of links encompassed by flow f
$\mathcal{F}(n)$	Set of flows passed through link n
x_f, x_h	Flow rate for flow f and total receiving rate for peer h
$z_h \in z$	Uplink capacity of peer h
$h_j \xrightarrow{f} h$	Peer h_j is the parent of peer h connected by flow f

Table 1: Notations used for network model

A.2 Best-Effort Network - Leveraging Heterogeneity

As previously mentioned, the present day Internet is a best-effort network that does not guarantee the QoS required for media streaming. The decentralized nature of the Internet means that packets belonging to a single stream between a source and a destination may traverse through unique routes and experience uncorrelated channel effects along the routes. This has led to the idea of *distributed video stream* [10, 11], where the authors proposed a distributed scheduling algorithm. Majumdar et al. [12] proposed a delivery scheme based on distributed Forward Error Correction (FEC). These efforts have shown a significant quality improvement in media streaming over the Internet compared to a traditional client-server model. The idea of distributed streaming naturally became popular in the context of P2P networks once the bandwidth required for video streaming became affordable to the average users.

The work by Hafeeda, et al. [13] introduces a dynamic peer selection scheme to media transmission. Padmanabhan, et al. [14], proposes a bandwidth adaptation protocol is proposed for increased transmission robustness in P2P network, while Agarwal, et al. [15] focused on the quality adaptive delivery of the media streams. Various QoS improvements

offered by these works mainly follows the general idea that participating clients that have already received the stream are able to help alleviate the server load by contributing to the media distribution process through *peer relaying*. Peers that relay received media data to other peers are called *relay peers*.

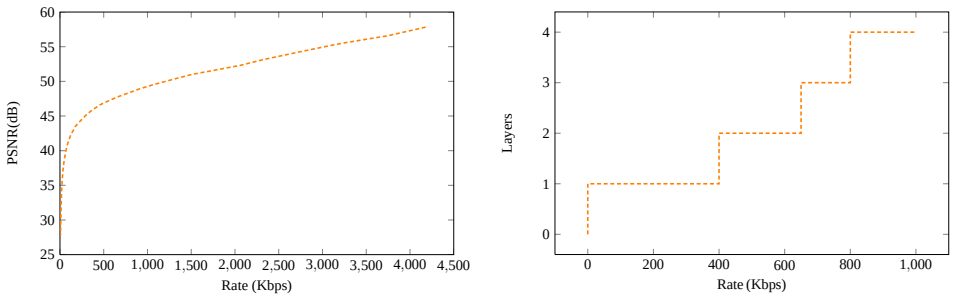
B Rate Adaptation

Due to the lack of QoS provisioning in P2P networks, proper utilization of the available network bandwidth is necessary to ensure user satisfaction. For video stream, this implies that the peers must perform rate adaptation to address bandwidth constraint and heterogeneous receiver capabilities inherent in a P2P network. Rate adaptation is particularly important in the context of peer-relaying because a relay peer must re-encode the incoming video stream according to its uplink capacity limit before it can relay the video to the a child peer.

Rate-adaptation implies that the video quality received by each peer is different. In the case of multicast, having multiple streams of varying bit-rates may force the overlay network to divide into smaller subgroups, each receiving only one version of the source stream. However, this clustering exposes the peers to bandwidth fluctuation and peer churning. Scalable video stream [16] has been proposed as a way to prevent clustering in a distributed network by offering a single video stream that serves a variety of bit rates. The video signal is encoded into several layers at the source, and the receiver only needs to receive a subset of the layers to recover the signal with a certain level of quality degradation. Therefore, layered video prevents clustering in the network because heterogeneity is addressed at the local nodes, where the sender selectively forwards the layers that fit within the allocated bandwidth. However, this increased flexibility of layered video also produces data overhead and reduces the coding efficiency. Throughout this thesis, the term *layered video* is used to refer to scalable video stream.

B.1 Video Encoding and Adaptation Techniques

This work focuses on two main encoding schemes: continuous-rate and scalable-rate encoding. Consequently, video streams encoded with these schemes are called continuous stream and scalable stream. In continuous-rate encoding, perceived video quality is directly proportional to the encoding rate. Fig. 2a shows the relation between media quality and the encoding rate in a continue stream. Media quality here is a logarithmic function of the rate. Rate adaptation for continuous stream is done by *transcoding*, where incoming video signal is changed by a relaying peer to meet lower uplink rate requirements, through either re-encoding or adjusting key video parameters such as quantization values, etc.



(a) Continuous rate encoding (b) Scalable rate encoding

Figure 2: Video encoding types

For scalable-rate encoding, raw video sequences are compressed into non-overlapped layers. The *base layer* contains data that constitutes the most important features of the video. Additional layers - called *enhancement layers* - contain data that progressively enhances the quality of the reconstructed video [17] on the receiver side. Fig. 2b shows the discrete stair-case relation between the encoding rate and the perceived media quality for scalable stream. The properties of scalable video dictates that a successful reconstruction of a layer depends on the availability of all previous layers. Therefore, rate-optimization for layered video is essentially maximization of the number of consecutive layers received by each peer in the network.

While the adaptation techniques adapt streams to a given rate, the rate adaptation algorithms determine the rate that maximizes the overall quality of the video streams received by all peers in the network.

B.2 Video Properties

The goal of the optimization algorithm is to globally optimize the video streaming rate and improve perceived video quality. In this regard, global rate optimization is dependent on the video technology.

Continuous Stream Video

The goal of the optimization algorithm is to reduce distortion to improve perceived video quality. Video distortion is defined as the loss in quality of the video upon decoding on the receiver side. It is composed of two components:

$$D^{dec} = D^{enc} + D^{loss} \tag{2}$$

where D^{enc} denotes the distortion introduced by quantization at the encoder, and D^{loss} represents the additional distortion caused by packet loss [18]. Typically, the distortion characteristics of the encoded video stream can be fit into a parametric model [18] as a function of rate x:

$$D^{enc}(x) = \frac{\theta^s}{x - x_0^s} + D_0 \tag{3}$$

The parameters (θ^s, x_0^s, D_0) depend on the coding scheme and the content of the video. They can also be estimated from trial encodings [18]. The distortion introduced by packet loss due to transmission errors and network congestion can be derived from [19]:

$$D^{loss} = \zeta P^{loss} \tag{4}$$

where the sensitivity factor ζ reflects the impact of packet loss P^{loss} and depends on both the video content and its encoding structure. For simplicity, throughout the rest of the thesis, it is assumed that packet losses due to transmission errors are addressed at the lower layers in the network stack (e.g., retransmission at the MAC layer or channel coding at the PHY layer). Hence, this thesis only focuses on the application-layer distortion due to encoding loss as mentioned in Eq. 3. The overall video quality can be evaluated by using the *Peak Signal-to-Noise Ratio (PSNR)* metric. The *PSNR* is defined as:

$$PSNR = 10\log_{10}(255^2/D^{dec}) = 10\log_{10}(255^2/D^{enc}) \tag{5}$$

where the term 255 is an encoder constant related to 8-bit pixel representation of color. Given the inverse relation between received rate and distortion, resource allocation of video streams in P2P networks involves minimizing the overall distortion (i.e., maximizing the allocated rate).

Scalable Stream Video

In scalable stream, the goal is to maximize the number of layers received. The basic network model described in Sec. A.1 is extended for scalable stream. Define a set of layers $\mathcal{L} = \{l_i \,|\, i = 1,2,\ldots,L\}$. The rate assigned to a particular layer l as x_l. The set of layers in flow f is defined as $\mathcal{L}(f)$. Therefore, each flow f has a rate x_f:

$$x_f = \sum_{l \in \mathcal{L}(f)} x_l \tag{6}$$

The set of layers received by a peer *h* from all of its parents is defined as:

$$\mathcal{L}(h) = \cup_{f \in \mathcal{F}_i(h)} \mathcal{L}(f) \tag{7}$$

Consider peers h_j and h_k such that $h_i \xrightarrow{f_{jk}} h_j$. Define λ to be a binary variable with the following properties:

$$\lambda_{jk}^{l} = \begin{cases} 1 & \text{if layer } l \text{ is present in } f \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

The objective function[1] is adopted from the definition in Eq. 1 and is given as:

$$\textbf{max.} \quad \sum_{h \in \mathcal{H}} x_h \tag{9}$$

$$\textbf{subject to} \quad \sum_{f_{km} \in \mathcal{F}_o(h_k)} \sum_{l \in \mathcal{L}(f_{km})} \lambda_{km}^{l} \, x_l \;\leq\; z_{h_k} \tag{10}$$

$$\sum_{l \in \mathcal{L}} \sum_{f_{jk} \in \mathcal{F}_i(h_k)} \lambda_{jk}^{l} \leq 1 \tag{11}$$

$$\sum_{l \in \mathcal{L}} \sum_{f_{jk} \in \mathcal{F}_i(h_k)} \left[\lambda_{jk}^{l+1} - \lambda_{jk}^{l} \right] \leq 0 \tag{12}$$

$$\sum_{l \in \mathcal{L}} \left[\sum_{f_{jk} \in \mathcal{F}_i(h_k)} \lambda_{jk}^{l} - \sum_{f_{km} \in \mathcal{F}_o(h_k)} \lambda_{km}^{l} \right] \geq 0 \tag{13}$$

where Eq. 9 is a non-linear function of the total allocated rate for all the peers in the network. The rate x_l for each layer is derived from the set $\mathcal{L}$ based on a constant rate that can be generated empirically. Eq. 10 and Eq. 11 are associated with network constraint. Eq. 10 states the capacity constraint for each peer. The duplicity constraint in Eq. 11 states that a peer should not receive same layer from multiple parents. Implementing Eq. 11 is not mandatory, however, algorithms that satisfy this constraint generally provides better optimization results because it reduces bandwidth waste. Eq. 12 and Eq. 13 are related to scalable video constraints. The continuity constraint in Eq. 12 states that the layers received by a peer from all of its parents in $\mathcal{L}(h)$ are consecutive. For example, if a peer has layer 3, the peer must have layer 1 and layer 2. The relay constraint in Eq. 13 states that

[1] For simplicity purposes, assume that the objective function excludes the rate maximization for root peers that act as servers.

a peer cannot receive a particular layer if none of its parents have that layer. The notations used for the extended network model and layered video properties are collected in Tab. 2.

Notation	Definition
$l \in \mathcal{L}$	Layer l in scalable video stream
$\mathcal{L}(f)$	Number of layers in flow f
$\mathcal{L}(h)$	Number of layers received by peer h
x_l	Rate required to stream layer l
λ_{ij}^{l}	Binary variable indicating the presence/absence of layer l between the flow from peer h_i to peer h_j

Table 2: Notations used in the extended network model for scalable video stream

C Congestion-Control Optimization

TCP is the most widely deployed transport protocol in the Internet. TCP provides end-to-end QoS support for delay insensitive bulk-data transfers. Multimedia-streaming applications are better served by slowly varying congestion-control mechanism that produces smoother bandwidth usage profile compared to TCP. This has lead to the emergence of equation-based congestion-control protocols, such as TFRC [4], XCP [20]. TFRC [21] (TCP Friendly Rate Control) is the most widely used TCP-friendly congestion control protocol. In TFRC, the sending rate is adjusted as a function of measured packet loss, instead of reducing it by half as in the case of TCP. RAP (Rate Adaptive Protocol) [22] is another TCP-friendly rate-control protocol that uses the AIMD-based algorithm to achieve inter-protocol fairness. LDA (Loss-Delay Based Adjustment) [23] is a RTP-based [24] (Real Time Protocol) rate-control protocol that essentially uses a AIMD-type congestion-control scheme and relies on RTCP (Real-time Control Protocol) feedback information. TCP-MR (TCP Minimum Rate) [25] is a variant of TCP that maintains minimum rate to ensure QoS for real-time multimedia data. Congestion-aware rate-control by adjusting the

encoding rate of the video is addressed in [26].

D Continuous-Stream Optimization

Continuous-stream optimization is based on the *Network Utility Maximization* (NUM) framework. The core idea of NUM is to decompose a centralized optimization problem (e.g., optimizing global utility of all peers) into sub-problems that are optimized locally (by each peer) in a distributed fashion. In the context of video streaming, the basic form of NUM attempts to maximize the sum of source utilities that are function of rates, under linear flow constraints.

D.1 Network-Utility Maximization

Let x_f be the rate assigned to a flow f in the network. Each flow is assigned a utility function U_f that describes the utility value $U_f(x_f)$ of the streaming rate x_f for a given application. Let $\mathcal{F}$ be the set of flows. The flows pass through a set of links $n \in \mathcal{N}$ that are capacity constrained by z_n. Therefore, the optimization problem maximizes the sum of the utilities of all flows, while satisfying the capacity constraints on each link in the network:

$$\textbf{max.} \quad \sum_{f \in \mathcal{F}} U_f(x_f) \tag{14}$$

$$\textbf{subject to} \quad \sum_{f \in n} x_f \leq z_n, \qquad \forall n \in \mathcal{N} \tag{15}$$

$$\textbf{over} \quad x_f \geq 0, \qquad \forall f \in \mathcal{F} \tag{16}$$

Given the role of the utility function as a metric that quantifies the efficiency of the rate allocation algorithm, the following question naturally arise: how to pick utility function?

In general, there are two types of utility functions: user-side utility and server-side utility. Utility on the user-side depends on user priorities. Typically, this means that the user-side QoS are direct functions of the received rate (as shown in Eq. 14). The user-side QoS can also be a function of delay, reliability, jitter, etc. The server-side utility primarily

addresses network-wide cost minimization for the operator. Typically, this involves setting utility as a function of congestion, load-balancing, energy-efficiency, etc. Overall, the goal of the utility function is to maximize the rate at a minimum cost, while maintaining fairness among users. The utility functions are assumed to be a smooth, concave and twice differentiable function of the data rate.

D.2 Optimization Algorithm

Since the utility function is continuous and twice differentiable, there exist a rate vector consisting of all flows that ensure global minimum distortion in Eq. 14. This technique is called *decomposition-based optimization* or *convex optimization*.

The basic idea of decomposition is to divide the original *primal problem* into smaller *sub-problems*, which are then co-ordinated by a *master problem*. Decomposition techniques can be classified into *primal-decomposition* and *dual-decomposition*. Primal decomposition decomposes the original problem, while the dual version decomposes the Lagrangian dual of the problem. In primal decomposition, the master problem directly gives each sub-problem a certain amount of resources it can use. Therefore, the role of the master problem is to optimize resource allocation to sub-problems. Under dual decomposition, the master problem sets the price for resource usage by each sub-problem. Depending on this price, the sub-problems decide the amount of resources to use. Therefore, the role of the master problem is to use the best pricing strategy. The heterogeneity associated with the constraint in Eq. 15 means that a dual decomposition is the most suitable way to achieve rate optimization in P2P networks. The dual decomposition with the Lagrangian duality [27] property, relaxes the primal problem by transferring the constraints to the objective function in the form of weighted sums.

In this thesis, an optimization algorithm is proposed under the NUM framework. The solution is based on dual decomposition. The proposed utility function simultaneously incorporates the capacity and the relay constraints to minimize rate distortion.

D.3 Related Work

Many of the rate optimization works are inspired by the seminal work by Kelly et al. [28, 29], which initiated a new approach of optimization-based modeling and decomposition-based solution to rate-optimization. As previously mentioned, this is called the *Network Utility Maximization (NUM)* framework. In NUM, peers in a distributed network collectively decide how much bandwidth each peer should receive. Each peer uses a *utility* as a function of its receiving rate. The goal is to maximize the aggregate utility of all peers, subject to various network constraints.

Low et al. [30] extended this pricing strategy to a distributed algorithm. This price-based approach has also been applied to multi-rate multicast by Kar et al. [31] by using sub-gradient projections and proximal approximation techniques [32]. Other works include multi-path unicast [33] and streaming [34], multicast over wireless network [35]. A comprehensive review can be found in [36]. Cui, et al. [37] applied this framework to overlay multicast, upon which this work is extended.

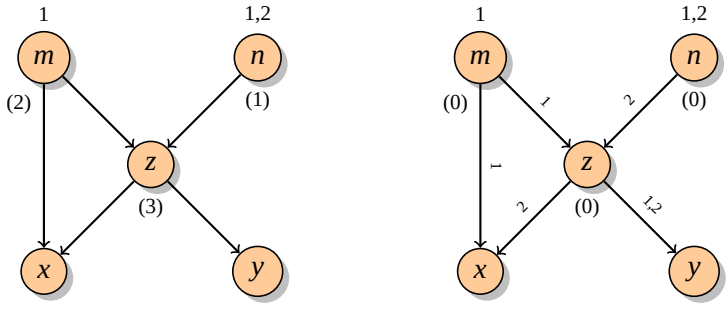
Recently, this framework has been applied to multicast tree construction in P2P systems [9]. A long line of works focus on different routing structures including single tree [7], multiple trees [38], and mesh [8] topology. This work deliberately avoids the routing functionality, but focuses on the optimized rate-allocation within any given routing structure. As such, this solution can work with any tree or mesh construction solutions.

The rate distortion function used for this thesis is first proposed in [18] using a parametric model. There also exists many analytic models [39] to describe rate distortion. Finally, Hsu et al. [40] provides a comprehensive review on this subject. There have been works on extending this framework under various networking scenarios. These works target to achieve optimized rate allocation by addressing the missing QoS provisioning mechanism of the underlying network. The works in [26, 41] address network congestion, while [42, 43] address the fairness and scheduling respectively. Finally [9, 34] target rate allocation by optimized routing.

E Scalable-Stream Optimization

Scalable videos are not amenable to the NUM framework due to the non-concave, discontinuous nature of its utility function. Any solution claiming global optimality is likely to be intractable in the context of large distributed network. Existing solutions use heuristic-based algorithm that locally optimizes a set of network or video related parameters. The optimization problem gets even more complicated in the presence of mesh network because peers can receive different layers from different peers. Therefore, unlike continuous stream video, receiving maximum rate does not alway translate into maximum effective layers received if the layers can not be decoded in consecutive order.

Example 1. *Fig. 3a illustrates the layer allocation problem in a P2P mesh. Assume that peer m and n holds layer* 1 *and layer* $1,2$ *respectively. Furthermore, delivering each layer requires unit rate. Peer m and n has a capacity of* 2 *and* 1*. In this example, peer z will act as a relay peer having capacity* 3*. A relay peer relays data received from its parent peer to its child peer. The goal is to distribute these layers from the server peers* $\{m,n\}$ *to the client peers* $\{z,x,y\}$ *so that the number of consecutive layers received by the clients are maximized. In this problem, peer m has layer* 1 *with* 2 *unit capacity and peer n has* 2 *layers*



(a) Multi-layer mesh (b) Optimal layer allocation

Figure 3: P2P mesh example

but 1 unit capacity. In addition, peer z can relay received layer with up to 3 *unit capacity. Fig. 3b shows the optimal layer allocation solution. In this case, peer m sends layer* 1 *to*

both child x and z. Since z already has layer 1 *from m, it receives layer* 2 *from peer z. Since peer x has already received layer* 1 *from peer m, it now receives layer* 2 *from peer z. Since z still has* 2 *unit uplink capacity left, z allocates* 2 *layer to peer y. This example illustrates the fact that unlike continuous video stream, the optimization for scalable video stream is non-convex and non-differentiable.*

McCanne, et al. [16] first proposed a Receiver-driven Layered Multicast (RLM) approach to address network heterogeneity. In RLM, layers are mapped to various multicast groups and the peers perform rate-adaptation by subscribing to these groups. Scalable stream has been proposed to address receiver heterogeneity [44] in the context of P2P streaming. Optimization of scalable stream has been proven to be NP-hard [44]. Research work on layered video optimization mainly proposes heuristics-based solutions and can largely be classified into three broad categories: *overlay construction*, *utility maximization*, and *layer scheduling*. Existing works based on these three broad categories are discussed below.

E.1 Overlay-Based Optimization

Proper overlay construction is indispensable in achieving layer optimization. A good overlay construction methodology is critical in order to send and receive layers from its neighboring peers. Approach to QoS-aware overlay construction can be divided into two mesh-based [7, 45] and tree-based [46, 47] topology. In both cases, newly joining peers first contact a predefined rendezvous point and successively probe existing peers to determine suitable joining position in the network. A tree-based topology can be trivially extended for multi-layered video streams. However, in the presence of peer churning, tree-based topologies are vulnerable to single point-of-failure. The multi-source approach of mesh-based topologies addresses this concern. However, it also introduces additional overhead that is required to properly receive multi-layered video streams.

Generally, a mesh-topology is constructed with gossip-based methodology due to its

inherent robustness against failure. Recently it has been used in various P2P media streaming systems such as PRO [48], PRM [49], and Chunkyspread [50]. In a gossip-based system, there is no defined parent-child relation and peers forward data to other peers that are expecting data packets. This approach is also called data-driven approach. Neighbor relation is managed by membership management protocols such as SCAMP [51], where peers use a random peer selection policy to determine its neighbors. However, lack of QoS awareness makes such selection policy unsuitable for large P2P network with hundreds of peers.

The mesh topologies previously mentioned only address single layered video. The joining criterion in these cases involve network conditions such as bandwidth availability and delay. However, a multi-layered video stream also requires finding neighbors who can supply sufficient number of layers. In this regard, Xiao, et al. [52] proposes a two-stage overlay construction approach for mesh network. Here QoS awareness is used on top of the gossip protocol to select neighbors. In the first stage of the join process, the joining peer probes the networking condition of its prospective neighbors along with their ability to supply a complete layers. The second stage ensures the improvement of the the QoS of the joining peer as well as the QoS of the neighbors.

Another approach to overlay construction is to use multi-tired topology. In this approach, each tier optimizes a single layer. Optimization of higher layer requires optimization of the overlay network associated with all the previous layers and may result in unique overlay network for each layer. Zhu, et al. [35] starts with a mesh network and generates Application Layer Multicast (ALM) tree for each layer. Initially, a parent determines it's ability to send a complete layer to its child. Once a layer is allocated to a child, a parent peer updates the available bandwidth used to allocate the next layer. If a peer is unable to allocate bandwidth for a layer, it is removed from the network of available peers used to construct the overlay for a layer. Guo, et al [53] proposes this multi-tiered approach for mesh construction on LSONet, where a QoS-aware data-driven method is taken for

neighbor selection.

Gossip-based mesh-topology construction offers robustness and multi-tired approach allows for the use of multi-layered video. However, layered video optimization based on successive multi-tired overlay construction does not always result in optimal allocation of layers.

Example 2. *Fig. 4 illustrates the problem with multi-tired mesh construction approach for scalable video optimization. Assume that each layer requires unit rate. The initial*

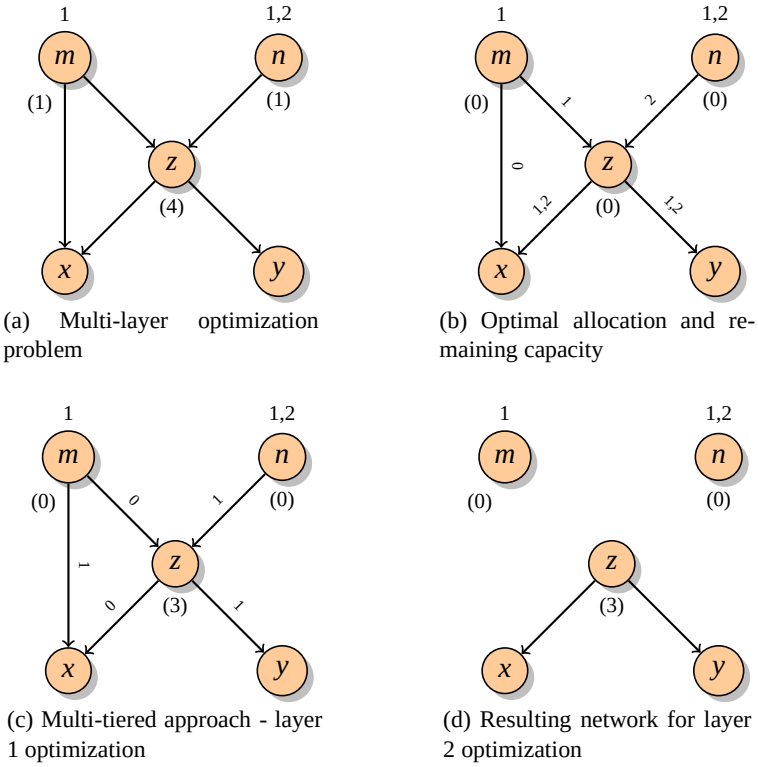


Figure 4: Multi-tier approach to layered video optimization example

configuration is shown in Fig. 4a. The capacity for each peer is shown in parenthesis. Fig. 4b shows the optimal allocation for this configuration. In this case, the servers m and n sends layer 1 *and* 2 *to peer z respectively. m does not allocate layers to peer x. Upon receiving layers, peer z sends both layers to peer x and y.*

In the case of multi-tiered approach, Fig. 4c shows the first step with the allocation of layer 1. Fig. 4d shows the resulting network after subtracting the required rate to deliver this layer. However, server n in this case does not have enough capacity to send layer 2 *to peer z. Therefore, the network capacity is under-utilized and the layers are not optimally allocated.*

E.2 Utility-Based Optimization

Network-utility maximization (NUM) for convex function has been studied extensively. Naturally, the non-convex nature of the layered video has led to the use of sigmoidal-like approximation-based utility function such that the NUM framework can be applied. Under this scenario, algorithms developed for optimization of convex functions can be used to optimize scalable video streams. Fig. 5 shows a sigmoidal utility function used to approximate stair-case utility function associated with scalable video. However, attempting to

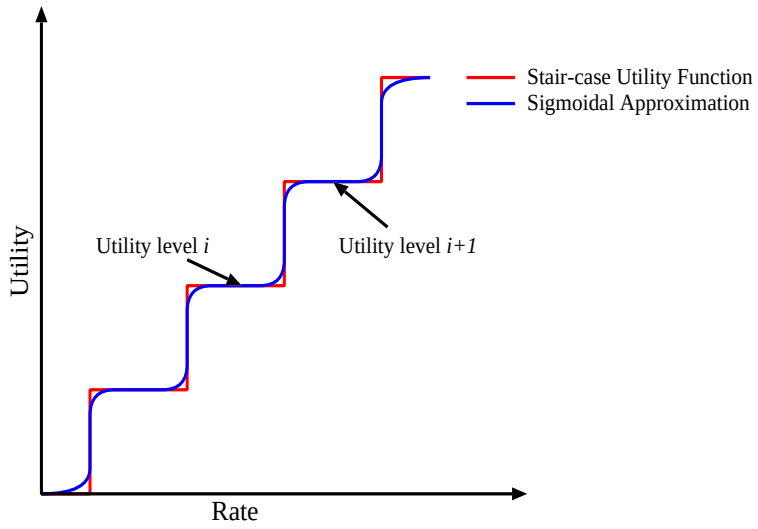
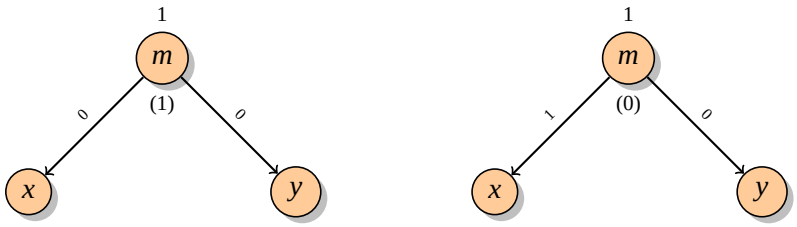


Figure 5: Sigmoidal utility function used to approximate stair-case utility function

smooth a stair-case utility function in order to take advantage of the properties of a convex optimization does not always lead to optimality.

Example 3. *Fig. 6 provides a simple example to illustrate one of the flaws of using sigmoidal-*

like function to optimize non-convex utility. In this example, parent m has layer 1 *with unit*



(a) Allocation with sigmoidal approximation

(b) Possible optimal allocation

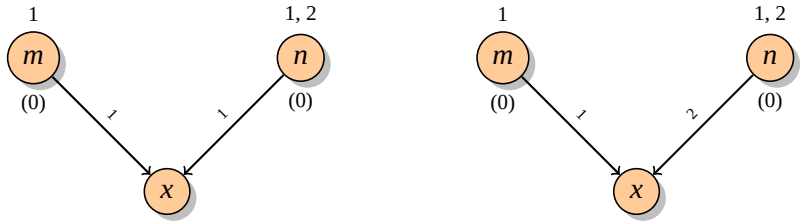
Figure 6: Sigmoidal Approximation for layered video optimization

capacity and allocation of a layer requires dedicating unit bandwidth as well. The utility-maximization algorithm for m allocates equal rate of 0.5 *for both x and y. Since, it is a discrete function and acceptable allocations are* 0 *and* 1 *only, m does not allocate layer to any of its children. Fig. 6a shows this resulting scenario. However, an optimal layer allocation considers the discrete nature of the utility function and allocate layer* 1 *to any one of the children and* 0 *layer to the other as shown in Fig. 6b.*

Heuristic-based algorithms have been proposed that are based on this sigmoidal-like utility function framework. Lee, et al. [54] proposed a distributed price-based heuristics that can self-regulate user access to resources. Hande et al. [55] developed conditions under which a distributed price-based algorithm can globally converge to an optimal rate at the presence of non-convex utility function. However, none of these solution truly addresses the concerns related to scalable video optimizations in a mesh network. A sigmoidal approximation assumes that the utility increases progressively with the allocated rate. However, in a mesh topology, peers can subscribe to multiple parents and receive different layers from different parent.

Example 4. *Consider the example in Fig. 7. Here the sigmoidal approximation algorithm allocates layer* 1 *from both parents of x as shown in Fig. 7a. This allocation not only results in redundancy but it also produces sub-optimal allocation for x. The optimal solution -*

shown in Fig. 7b - for x will be to receive layer 1 *from m and layer* 2 *from n.*



(a) Allocation with sigmoidal approximation

(b) Optimal allocation

Figure 7: Sigmoidal Approximation for layered video optimization

E.3 Scheduling-Based Optimization

Most of the scheduling-based approaches [43, 44, 56] are heuristic-based algorithm that mainly adheres to the following optimization procedure. The algorithm assumes the presence of a set of sender from which a peer will receive layers. The receiver uses optimization algorithm to maximize the number of received layers based on the available layer information received from the sender. Cui, et al. [44] uses a greedy-based approach to receive maximum number of layers, while PALS [56] determines the number of received layers by maximizing the throughput as well as the number of received layers. PALS proposes a diagonal buffer distribution and employs round-robin method to request data.

Recently, a chunk-based mesh-pull strategy has been proposed in LayerP2P [43, 57] to optimize multi-layered video in P2P mesh. Here each layer is broken into smaller chunks, which are then pulled from neighbors in a data-driven approach. LayerP2P in [43] uses a 3-stage strategy to address various QoS requirements such as minimizing the number of useless packets and maximizing the number of layers delivered. Useless packets in this case refers to packets belonging to layer l that cannot be decoded due to missing packets from previous layers. Liu et al. [57] optimizes the number of received layers by using an incentive-based tit-for-tat data dissemination strategy to ensure fairness.

F Message-Passing Based Optimization Framework

The heuristic-based algorithms discussed in Sec. E attempt to optimize the number of layers received. However, there is generally no guarantee that such algorithm provides optimal solutions, or even converges. The message-passing based algorithm discussed next, seeks to address this issue for non-convex constraint-based optimization problem.

F.1 Related Work

Recently, there has been an explosion of interest towards *message-passing algorithm* to solve problems that can be modeled under node-based graphical structures. They have been independently discovered in a number of different fields. Introduced by Gallager [58] in the context of low-density parity-check (LDPC) codes, message-based sum-product algorithm was first invented for *a posteriori* probability (APP) decoding. Tanner et al. [59] first explicitly introduced graphs to describe LDPC codes, constraints and the optimality of the sum-product and min-sum algorithms for decoding codes on a cycle-free graph. Wiberg et al. [60] introduced states in tanner graphs. Eventually, the application of the sum-product algorithm has expanded to a variety of fields: statistics, communications, signal processing, probability theory, etc.

The seminal work of Ahlswede et al. [61] extended this into the field of computer networks (initially defined by a point-to-point communication network where a number of information sources are to be multicast to a certain sets of destinations) by introducing network coding and the idea of admissible coding rate region. Here the admissible coding rate region is defined as a coding scheme satisfying the link capacity between two communicating points in a network.

F.2 Application to Distributed Systems

In the context of a graph consisting of nodes and edges, the message-passing algorithm operates by passing messages with neighbors along edges. Messages are updated iteratively until convergence is achieved. Messages are only passed between nodes connected to each other. Because of this localized nature, message-passing algorithms can be implemented in a decentralized and asynchronous fashion.

Interest in message-passing algorithm has largely been triggered by the success of *turbo decoding* [62]. It is routinely used in communication systems that employ error-correcting codes. Turbo decoding aims to solve an NP-hard problem. Separately, inspired by ideas from statistical physics, message-passing algorithm have been proposed for solving NP-hard combinatorial optimization problems such as graph coloring [63].

One may claim this to be another ad-hoc, heuristic-based approach. However, in the instances noted above, it is the state-of-the-art method for optimization. Furthermore, it provides a rich structure that can be used as an analytical tool. Recently, work has been done to define message-passing framework for inelastic rate optimization [64]. However, no work has been done to apply message-passing algorithm for scalable video streaming in the context of distributed systems, such as P2P networks.

F.3 New Application Framework for Scalable Stream

In this thesis, the admissible code region for a peer is defined as a code that satisfies the requirements of network and layered video streams in the context of the P2P network. Based on these admissible codes, the iterative sum-product update algorithm is then applied to determine layer subscription from neighboring peers.

Formally, the thesis contribution is two fold. First, a flexible optimization approach is presented that codifies the optimization requirements. Second, the constraints to determine admissible codes and a sum-product update algorithm that iteratively applies incoming messages on the codes to generate updated outgoing messages are presented. On sender

side, the algorithm works by iteratively updating probability for allocating a layer to a child. On the receiver side, the algorithm works by determining probability of receiving a layer from a parent. Convergence is reached when both sender and receiver agrees on a probability value.

Intuitively, the algorithm can be interpreted as a type of positive feedback loop, where peers send and receive likelihood probability values in order to establish active parent-child relation. A sending peer increases the outgoing probability along the path it receives the maximum incoming probability. Similarly, the receiver also performs the same action on its outgoing probability messages. A probability value of 1, passed along a parent-child connection implies that the child will receive a layer. However, this value is contingent upon the fact that the parent must also receive this layer from its own set of parents.

CHAPTER III

CONGESTION CONTROL PROTOCOL

The TCP-friendly congestion-control protocols discussed in Sec. II.C uses regular feedback for each packet sent and depends on the round-trip time (RTT) formulated in [65] to adjust the sending rate. In this thesis, a congestion-aware MEdia-friendly real-Time Streaming protocol (METS) is presented. METS does not require sending explicit ACK for each transmitted packet. Instead, METS periodically exchanges state information between the sender and the receiver and updates the RTT value. METS addresses the issues of network congestion and packet-loss based on the measured *inter-packet gap* (IPG) information on the sender and the receiver side. In particular, it is assumed that the packet-size and the video encoding rate are fixed.

A Protocol Overview

Streaming in METS follows a sliding-window based technique, where a window consists of k slots. Each slot corresponds to a packet. The sender maintains the sliding-window buffer of k slots that are determined by the delay-bandwidth-product (DBP). The DBP refers to the product of the link-capacity (in bit/second) and the end-to-end delay between two communicating nodes in a network. Packets are transmitted periodically from the sender-side with sending IPG defined as δ^s. Similarly, the receiving IPG is defined as δ^r. The initial sending IPG value is set based on the fixed video generation rate. For a given packet of size β^{ps} and encoding rate β^e, the video IPG δ^v is defined as $\frac{\beta^{ps}}{\beta^e}$. Therefore, the initial sending rate is:

$$\beta^s = \frac{1}{\delta^v}\beta^{ps} \tag{17}$$

On the sender side, the window buffer is refreshed periodically after each IPG time and a new packet enters the buffer if available. Each new packet is given a sequence number n and a initial time-to-live (TTL) value (τ_{ttl}^{n}) equal to the window size k. After a buffer-refresh event, this TTL value is decreased by 1 for all packets currently in the buffer. Once the TTL value reaches 0, a packet is discarded. As soon as a packet enters the buffer, it is scheduled for transmission. Tab. 3 at the end of this chapter contains the list of notations used in this chapter.

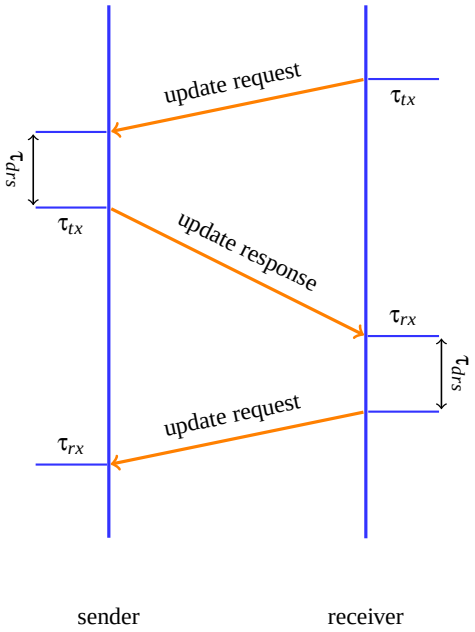


Figure 8: Periodical RTT update

A.1 Buffer Update

The receiver periodically initiates an update request message and measures the round-trip time (RTT). The sender measures its own RTT time when the receiver sends the next update request. Fig. 8 shows the RTT update process. Here τ_{drs} on the sender side is the elapsed time between receiving an update request and sending an update response. On the receiver side, this corresponds to the elapsed time between receiving an update response and sending the next update request. The τ_{drs} value in this case is embedded in the update

message. The RTT time can now be calculated as:

$$\tau_{rtt} = \tau_{rx} - \tau_{tx} - \tau_{drs} \tag{18}$$

Based on this RTT value, the sender can now calculate the DBP and determine the window size k:

$$k_s = \frac{\tau_{rtt} \cdot \beta^e}{\beta^{ps}} \tag{19}$$

The receiver-side buffer window is determined based on the video IPG and initial start-up delay, τ_{st}:

$$k_r = \frac{\tau_{st}}{\delta^v} \tag{20}$$

A.2 Sender Rate-Adaptation

The basic premise of the protocol is that the sender assumes each packet transmission event to be a success unless it receives an update request from the receiver. An update request can be of two types:

- **Rate-adjustment Request**: The receiver in this case requests the sender to increase the sending IPG (i.e., decrease the sending rate) to address the existing network congestion.
- **Retransmission Request**: The receiver requests the sender to re-transmit a particular packet that is assumed to be lost.

Definition 1. *Upon a successful transmission, the sender increases the sending rate by decreasing the IPG value for the next transmission event. For simplicity, only a linear increase/decrease of the sending IPG value is considered:*

$$\delta^s_{n+1} = (1 \pm \alpha)\, \delta^s_n \tag{21}$$

The value of α *is usually set between* 0.1 *and* 0.3 *[21]. Upon receiving a rate-adjust request*

from the receiver, sender records the current time-window κ *and periodically increases the IPG value during each transmission event until the following inequality is satisfied:*

$$k\, e^{-(n-\kappa)} \geq \varepsilon^{s} \tag{22}$$

The IPG-value increase is based on Eq. 21. Once Eq. 22 is satisfied and there are no pending rate-adjustment request, the sender resumes decreasing the IPG value.

A.3 Receiver Rate-Adaptation

The receiver measures the IPG of the arriving packets and also stores the maximum IPG value among the last *k* packets. Entropy of the IPG is used to calculate the effective IPG value on the receiver side.

Entropy

Entropy captures the uncertainties in a random variable. Shannon's entropy can be used to capture the randomness in the IPG data measured by the receiver. A high entropy value indicates the presence of congestion. First, the probability distribution function (PDF) of the IPG values of the last *k* packets is calculated. Let p_i be the probability of having IPG value *i* during the last *k* window-slots. Shannon's entropy can be calculated as:

$$H_S = \sum_i p_i \log p_i \tag{23}$$

However, Shanon's entropy increases in the presence of small spikes and does not fully capture the dominant PDF values in the IPG data. Reneyi's entropy [66] is used to capture the dominant IPG values:

$$H_R = \frac{1}{1-q} \log_2 \sum_i p_i^q \tag{24}$$

Renyi's entropy masks the low probability values by raising them to high powers (q). Thus it depresses small PDFs but raises the higher PDF values. A detailed discussed on clustering of IPG data based on entropy can be found in [67].

Definition 2. *Define δ^{max} as the maximum IPG recorded for the last k packets. Therefore, the average IPG time can be calculated as:*

$$\delta_{n+1}^{av} = \delta^{max} \cdot H_R \tag{25}$$

A.4 Receiver Feedback

The receiver feedback function involves sending requests to the sender regarding re-transmission of lost packets or rate adjustment. This involves answering the following questions:

- If a packet is missing, how long should the receiver wait before sending a re-transmission request?
- If the receiver receives data at a faster rate than it can process or that the packets are suddenly arriving at higher IPG rate than the sender is sending at (i.e., presence of congestion), when should the receiver send a rate-adjustment request?

Rate-adjustment Request

The receiver keeps track of the network-congestion by calculating the average IPG value based on Eq. 25. Based on this information, the sender decreases the sending rate (i.e., increases the IPG value) when the difference between the average receiving IPG and the video IPG goes above a certain threshold.

Retransmission Request

If a packet does not arrive within its expected time-interval, it is considered lost. The receiver must send a retransmission request for the lost packet before the TTL value falls

below a certain threshold. The threshold exist to ensure that a retransmission request for a packet reaches the sender before it is being discarded and that the packet reaches the receiver before the playback deadline.

Along with the fixed buffer of size k_r associated with fixed delay, the receiver also maintains a dynamic buffer that is equal to the size of the sender-side buffer. The buffer-size information can be passed to the receiver by embedding it in the update-response message. Therefore, for the dynamic buffer window-slots, TTL value is given based on the sender buffer-size k_s. After a refresh event n, a new packet with sequence number n is expected to arrive within the next IPG interval in the dynamic buffer. As with the sender-side buffer, the TTL value of the window-slot is decreased by 1. Once it reaches 0, it is passed to the read-only fixed-size (k_r) buffer for playback.

Definition 3. *For a missing packet n in the dynamic buffer, the client sends a re-transmission request if the following inequality fails:*

$$\tau_{ttl}^{n} \geq k_s - \frac{\tau_{rtt}}{\delta^{v}} \tag{26}$$

where τ_{rtt} is the round-trip time based on the latest update response received from the sender.

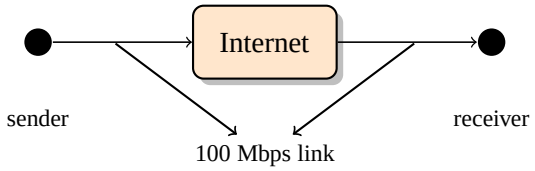


Figure 9: Experimental topology

B Experiment

The general network topology for this experiment is depicted in Fig. 9, where a single sender-receiver pair connects to the network through high-speed link. For this experiment,

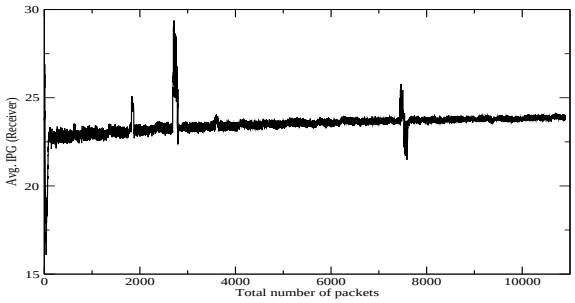


Figure 10: Average inter-packet gap of the received packets

the sender was located at a server at the ECE Dept. of University of Toronto and the receiver was located the Vanets server at the Vanderbilt University. The video is encoded at 87.5 KB/s. The packet size is set to 2 KB. Therefore, the sending IPG value is approximately 23.5 ms. The start-up delay is assumed to be 300 ms on the receiver side. The experiment is run for approximately 11,000 packets (i.e., 258 seconds).

B.1 Results

Fig. 10 shows the average IPG value on the receiver side. The spikes in this figure imply the presence of congestion in the network. This is confirmed in Fig. 11, based on the corresponding retransmission requests by the receiver. Fig. 11 also shows the results of this request from the receiver. A value of 1 corresponds to a successful retransmission, while a 2 corresponds to a failed retransmission because the sender has already discarded the packet. The IPG values in Fig. 10 also shows that the protocol quickly adjusts IPG value to account for the network congestion.

B.2 Message Overhead

The message overhead required to implement this algorithm is minimal outside of the update request and update response. The sender embeds the buffer size in the update

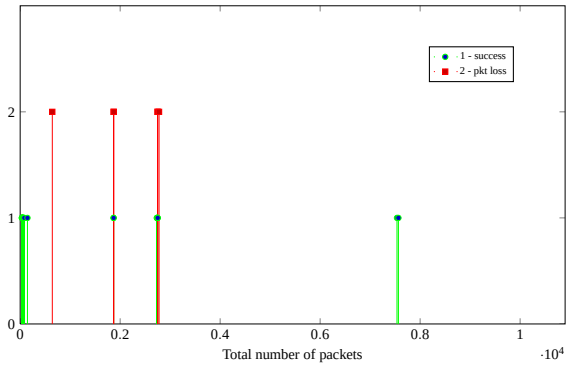


Figure 11: Successful and failed re-transmission by the sender

Notation	**Definition**
δ^v, δ^s, δ^r	IPG of video data generation, data transmission by the sender, and reception by the receiver
δ^{av}, δ^{max}	Average and maximum IPG
β^e, β^{ps}, β	Video encoding rate, packet size, and data sending rate
τ^n_{ttl}	Time-to-live value for packet number n
τ_{rtt}	Round-trip time
τ_{tx}, τ_{rx}	Sending and receiving time-stamp on a packet
τ_{st}	Start-up delay
τ_{drs}	Elapsed time between receiving a request and sending response
k_s, k_r	Sender and receiver-size window size
H_S, H_R	Shanon and Renyi's entropy
p_i	Probability of having IPG value i

Table 3: Notations used in METS

response and the receiver embeds the average IPG value in update request message. The only external message required is the re-transmission request by the receiver.

CHAPTER IV

CONTINUOUS STREAM OPTIMIZATION

In this chapter, rate-adaptation for continuous stream in a P2P network is presented. The goal is to minimize rate-distortion for video streams under the NUM framework. The solution takes into account peer relaying - a constraint unique in P2P distribution scenario in which a peer is both a receiver and a sender. While helping with content distribution, peer relaying constraint also ensures that the receiving rate of a child peer does not exceed the receiving rate of its parent. This is because during the rate-adaptation, once the video quality is lost, cannot be recovered. As such, the rate change occurred on one peer not only changes the video quality for itself, but also for all of its child peers. Therefore, price-based resource allocation that considers peer relaying, ensures that peers with more children receives higher bandwidth compared to peers with fewer children.

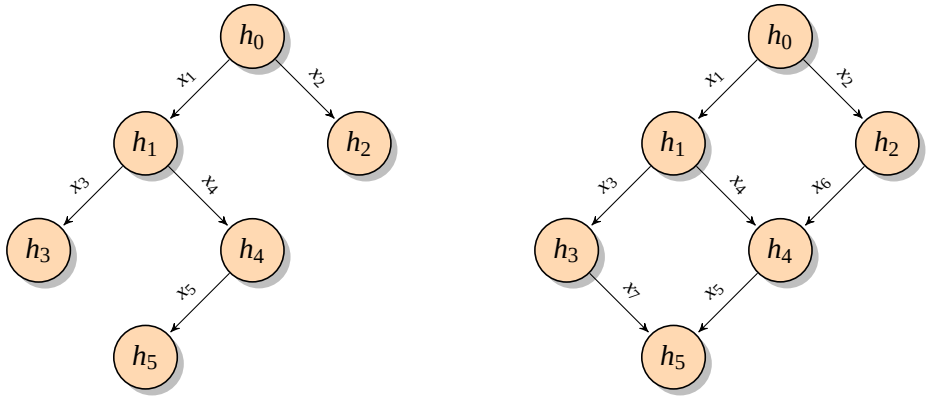
Simulation shows that simultaneously incorporating both network and relay constraints significantly reduces the aggregate rate distortion for all peers. For this simulation, mesh and tree-based topologies are considered [68, 69]. The optimization solution presented in this thesis is called *double-pricing* solution.

A Double Pricing Solution

As previously mentioned, the double-pricing solution simultaneously takes into account the *capacity constraint* and the *relay constraint* associated with the peers in a network. The capacity constraint states that for each link n, the total volume of its flow set $\mathcal{F}(n)$ cannot exceed its capacity z_n. Formally, let A be an $N \times F$ matrix, such that $A_{nf} = 1$ if flow f goes through the link n (i.e., $f \in F(n)$). Otherwise, $A_{nf} = 0$.

$$A \cdot x \leq z \tag{27}$$

The *relay constraint* states that the receiving rate of a child peer cannot exceed the receiving rate of its parent. This is illustrated with the overlay tree shown in Fig. 12a. In this picture, according to the relay constraint, the video quality received by h_3 and h_4 cannot exceed the quality received by their parent h_1. Therefore, the corresponding rates x_3 and x_4 cannot exceed the rate x_1. In case of P2P mesh shown in Fig. 12b, consider the peer h_4. It receives data from both peer h_1 and h_2. Consequently, the outgoing flow rate of h_4 can not exceed the total incoming flow rate. Formally, the relay constraint in this case states that $x_5 - x_4 - x_6 \leq 0$.



(a) P2P tree (b) P2P mesh

Figure 12: P2P overlay network

Since any peer can be the parent of any other peer, the total number of such parent-child pairs[1] is H^2. The relay constraint is formulated in a $H^2 \times F$ matrix B as follows:

$$B_{((h_k-1)H+h_i)\cdot f} = \begin{cases} -1 & \text{if } h_k = h(f) \text{ and } h_k \to h_i \\ 1 & \text{if } h_i = h(f) \text{ and } h_k \to h_i \\ 0 & \text{otherwise} \end{cases} \tag{28}$$

Since, $\boldsymbol{B}$ is a sparse matrix, where the $((h_k-1)H+h_i)$th row will only be active if there is

[1]There are in fact several special cases which forbid parent-child pairs. For example, the server h_0 cannot be the child of any peer, also a peer cannot be the parent of itself. These cases are not included in the formulation for simplicity purposes. Nevertheless, the actual number of parent-child pair number remains in the order of H^2.

a flow from h_k to h_i. The relay constraint can now be formalized as follows:

$$B \cdot x \leq 0 \tag{29}$$

Along with the network notations defined in Tab. 1, additional notations used in this chapter are collected in Tab. 4.

A.1 Utility Function

The overall objective is to minimize the aggregate distortion of the streaming video received by all peers. Given the rate distortion definition in Eq. 3, the rate allocation for a tree-based network can be formulated as the following non-linear optimization problem[2]:

$$\textbf{min.} \qquad \sum_{f \in \mathcal{F}} D_f(x_f) \tag{30}$$

$$\textbf{subject to} \qquad (\textbf{27}) \textbf{ and } (\textbf{29}) \tag{31}$$

$$\textbf{over} \qquad x \in I_f \tag{32}$$

where Eq. 30 is a convex function of the allocated rate. The rate of each flow x_f is adapted within the range $I_f = [m_f, M_f]$. For a mesh-based network, the goal of optimizing the rate of all the incoming flows of a peer under the same constraints as in Eq. 31 and Eq. 32 can be stated as:

$$\textbf{min.} \qquad \sum_{h \in \mathcal{H}} \sum_{f \in \mathcal{F}_i(h)} D_f(x_f) \tag{33}$$

By non-linear optimization theory, there exists a minimizing value for the rate vector x for the above optimization problem. Consider the Lagrangian form of this problem in Eq. 30 for a tree-based network (the Lagrangian for the mesh-based problem in Eq. 33 trivially

[2]Note that the objective function should exclude the rate distortion function for server h_0. For simplicity purpose, this detail is ignored in the rest of the chapter. This can be easily achieved by assigning value 0 to the rate distortion function of h_0.

follows the same steps):

$$L(x, \mu^{\alpha}, \mu^{\beta}) \tag{34}$$

$$= \sum_{f \in \mathcal{F}(h)} D_f(x_f) + \mu^{\alpha}(A \cdot x - z) + \mu^{\beta}(B \cdot x) \tag{35}$$

$$= \sum_{f \in \mathcal{F}} D_f(x_f) + \sum_{f \in \mathcal{F}} x_f \sum_{n \in \mathcal{N}} \mu_n^{\alpha} A_{nf} + \sum_{f \in \mathcal{F}} x_f \sum_{k=1}^{H^2} \mu_k^{\beta} B_{kf} - \sum_{n \in \mathcal{F}} \mu_n^{\alpha} z_n$$

where $\mu^{\alpha} = \{ \mu_n^{\alpha}, n \in \mathcal{N} \}$ and $\mu^{\beta} = \{ \mu_k^{\beta}, k = 1, 2, \ldots, H^2 \}$ are vectors of Lagrangian multipliers. Let us define two new vectors $\lambda^{\alpha} = \{ \lambda_f^{\alpha}, f \in \mathcal{F} \}$ and $\lambda^{\beta} = \{ \lambda_f^{\beta}, f \in \mathcal{F} \}$ as follows:

$$\lambda_f^{\alpha} = \sum_{n \in \mathcal{N}} \mu_n^{\alpha} A_{nf} = \sum_{n \in \mathcal{N}(f)} \mu_n^{\alpha} \tag{36}$$

$$\lambda_f^{\beta} = \sum_{k=1}^{H^2} \mu_k^{\beta} B_{kf} = \sum_{h \to h(f)} \mu_{(h-1)H+h(f)}^{\beta} - \sum_{h(f) \to h} \mu_{(h(f)-1)H+h}^{\beta} \tag{37}$$

Therefore, Eq. 34 becomes

$$L(x, \mu^{\alpha}, \mu^{\beta}) = \sum_{f \in \mathcal{F}} D_f(x_f) + \sum_{f \in \mathcal{F}} x_f (\lambda_f^{\alpha} + \lambda_f^{\beta}) - \sum_{n \in \mathcal{N}} \mu_n^{\alpha} z_n$$

where μ_n^{α} is the *link price*. Consequently, λ_f^{α} is the sum of prices of all links in f's path (i.e., the *network price* mentioned in Eq. 27 that f has to pay). In the star topology, f only has to pay the price for the uplink bandwidth of its sending peer. $\mu_{(h_i-1)H+h_k}^{\beta}$ is the *relay price* that peer h_k has to pay to its parent h_i for relaying the data. λ_f^{β} can be interpreted as relay price for f, which is the difference between the aggregated relay price of parent of $h(f)$ and the relay benefit h_f receives from all of its children.

Solving Eq. 30 involves two sets of prices, each corresponding to one of the two constraints defined in Eq. 27 and Eq. 29. Therefore, it is called the double-pricing solution, in contrast to many existing solutions in the literatures [28–30] that only considers the capacity constraint of Eq. 27 by treating all flows as independent from each other. Comparison

Notation	Definition
$A = (A_{nf})_{N\times F}$	Link capacity constraint matrix
$B = (B_{kf})_{F\times F}$	Relay constraint matrix
$D_f(x_f)$	Rate distortion function for flow f
$\mu_n^{\alpha} \in \mu^{\alpha}$	Link price for link n
$\mu_f^{\beta} \in \mu^{\beta}$	Relay price for flow f
$\lambda_f^{\alpha} \in \lambda^{\alpha}$	Sum of link prices in $\mathcal{N}(f)$
$\lambda_f^{\beta} \in \lambda^{\beta}$	Aggregate relay price
$x_f \in x$	Flow rate collected in flow vector x
x_0^s, θ^s	Encoding parameters specific to a video sequence

Table 4: Notations used in continuous stream optimization algorithm

of these two solutions with respect to the tree and mesh-based topology will be the main theme of the simulation study in this chapter.

A.2 Distributed Algorithm

The problem in Eq. 34 can be solved in a distributed fashion, following the gradient projection method adopted by many existing works. The dual of problem can be written as:

$$\min_x L(x, \mu^{\alpha}, \mu^{\beta}) = \min_x \sum_{f\in\mathcal{F}} (\Phi(x_f)) - \sum_{n\in\mathcal{N}} \mu_l^{\alpha} z_n$$

where $\Phi(x_f)) = D_f(x_f) + x_f(\lambda_f^{\alpha} + \lambda_f^{\beta})$ is the *total cost* for flow f (i.e., the rate distortion it receives and the aggregated cost), which is the product of the flow rate and the combined network and relay prices. By the separation of Lagrangian form, minimizing $L(x, \mu^{\alpha}, \mu^{\beta})$ can be decomposed into separately minimizing $\Phi(x_f)$ for each flow f. Since D_f is strictly convex and twice continuously differentiable, a unique minimizer of $\Phi(x_f)$ exists when

$$\frac{d}{dx_f}\Phi(x_f) = \frac{d}{dx_f} D_f(x_f) + (\lambda_f^{\alpha} + \lambda_f^{\beta}) = 0$$

Thus, the optimal rate for a flow f can be calculated as:

$$x_f(\mu^\alpha, \mu^\beta) = arg \min_{x_f \in I_f} \Phi(x_f) = \left[D_f^{'-1}(\lambda_f^\alpha + \lambda_f^\beta) \right]_{m_f}^{M_f} \tag{38}$$

Combining Eq. 38 with the rate distortion function in Eq. 3 along with the network prices λ_f^α and relay price λ_f^β, the optimal rate can then be calculated as follows:

$$x_f(\mu^\alpha, \mu^\beta) = \left[x_0^s + \sqrt{\frac{\theta^s}{\lambda_f^\alpha + \lambda_f^\beta}} \right]_{m_f}^{M_f} \tag{39}$$

To this end, the distributed algorithm presented here proceeds in rounds denoted as $t = 1, 2, \ldots$. Each round involves two steps. The first step is the price update, where price vectors are adjusted in opposition direction to the gradient $\nabla D(\mu^\alpha, \mu^\beta)$:

$$\mu_n^\alpha(t+1) = \left[\mu_n^\alpha(t) + \gamma \left(\sum_{f \in \mathcal{F}_o(n)} x_f(t) - z_n \right) \right]^+ \tag{40}$$

$$\mu_{(h-1)H+h(f)}^\beta(t+1) = \left[\mu_{(h-1)H+h(f)}^\beta(t) + \gamma \left(x_f(t) - \sum_{f \in \mathcal{F}_i(h)} x_f(t) \right) \right]^+ \tag{41}$$

The second step is the rate update, where the rate of flow f is adjusted according to the price change. The rate x_f is calculated based on Eq. 39. This step requires the knowledge of network price λ_f^α and relay price λ_f^β, whose definitions can be found in Eq. 36 and Eq. 37 respectively.

Tree-based Implementation

The algorithm is a sender-driven implementation, starting with the price update. As seen in Eq. 40, to update the price of link *n*, one needs to know about its old price, and the rate of all flows going through it. Since this solution builds upon the star-topology assumption,

n must be a uplink of either the server or a peer, and all flows on this link must be generated from this peer as well. Obviously, the peer owning n is the best candidate to be the bookkeeper of its price μ_n^α.

The relay price given in Eq. 41 applies to the parent-child pair $h \rightarrow h(f)$, where flow f directs from peer h to $h(f)$. To update it, one needs to know the old price, as well as peer $h(f)$'s receiving rate x_f and the receiving rate $\sum_{f \in \mathcal{F}_i(h)} x_f(t)$ of its parent h. Therefore, the best candidate to calculate and maintain the relay price is h. It can easily measure the rates of both flows since one of them enters h and the other exits from it.

To understand how to implement the rate update step for flow f, consider the parent-child pair $h \rightarrow h(f)$. As outlined in Eq. 39, calculating x_f requires the knowledge of network price λ_f^α and the relay price λ_f^β. Given the definition of λ_f^α in Eq. 36 and the star-topology assumption, it can be easily seen that λ_f^α is the price of the uplink of h, the sender of flow f. Since h is also the bookkeeper of its own uplink price, λ_f^α will involve no messaging overhead if the sender-based approach is used (i.e., let h be in charge of the rate of flow f). Based on Eq. 37, λ_f^β is the difference between the relay price of parent-child pair $h \rightarrow h(f)$ and the sum of the relay prices of all parent-child pairs originating from $h(f)$. Since the bookkeeper of a parent-child pair's relay price is the parent, calculating λ_f^β requires that h receives message from $h(f)$ reporting all the relay prices managed by $h(f)$.

Calculating x_f requires additional video specific parameters (θ^s and x_0^s). The server can embed them into video packets at the beginning of the P2P streaming.

Mesh-based Implementation

The rate-allocation algorithm for mesh-based network follows the tree-based approach with one exception: calculation of the relay price. The algorithm integrates the multi-path scenario of [70] to adjust for relay price. Algorithm 1 presents the distributed algorithm that each peer executes to update its rate.

For each flow $f \in \mathcal{F}_i(h)$, the link prices μ_n^α for each link n associated with the flow and

Algorithm 1 Distributed Rate Allocation Algorithm

Each end host h at times $t = 1, 2, \ldots$

Initialization

$\mathbb{F}_i = \{ f \mid f \in \mathcal{F}_i(h) \}$

$\mathbb{R}_i = \emptyset$

for each $f \in \mathbb{F}_i$ **do**

  for each $n \in \mathcal{N}(f)$ **do**

    $\mu_n^{\alpha}(t+1) \leftarrow \left[\mu_n^{\alpha}(t) + \gamma\big(\sum_{f \in \mathcal{F}_o(n)} x_f(t) - z_n \big) \right]^+$

  end for

  $\lambda_f^{\alpha}(t) \leftarrow \sum_{n \in \mathcal{N}(f)} \mu_n^{\alpha}(t)$

  $\mu_f^{\beta}(t+1) \leftarrow \left[\mu_f^{\beta}(t) + \gamma\big(x_f(t) - \sum_{f^p \in \mathcal{F}_i(h(f'))} x_{f^p}(t) \big) \right]^+$

  for each $f \in \mathcal{F}_o(h)$ **do**

    $\mu_f^{\beta}(t+1) \leftarrow \left[\mu_f^{\beta}(t) + \gamma\big(x_f(t) - \sum_{f^i \in \mathcal{F}_i(h)} x_{f^i}(t) \big) \right]^+$

  end for

  $\lambda_f^{\beta}(t) \leftarrow \left[\mu_f^{\beta}(t) - \sum_{f^i \in \mathcal{F}_o(h)} \mu_{f^i}^{\beta}(t) \right]$

end for

$\lambda_{f'}^{\alpha}(t) = \min_{f \in \mathcal{F}_i(h)} \lambda_f^{\alpha}(t)$

$\lambda_{f'}^{\beta}(t) = \min_{f \in \mathcal{F}_i(h)} \lambda_f^{\beta}(t)$

$\lambda_{f'}(t) \leftarrow \lambda_{f'}^{\alpha}(t) + \lambda_{f'}^{\beta}(t)$

Source rate $x_h(t+1) \leftarrow \left[D'^{-1}(\lambda_{f'}(t)) \right]_{m_f}^{M_f}$

for each $f \in \mathcal{F}_i(h)$ update flow rate **do**

  $x_f(t+1) \leftarrow \left[x_f(t) - \gamma\Big(\lambda_f^{\alpha}(t) + \lambda_f^{\beta}(t) - \lambda_{f'}(t) \Big) \right]^+$

end for

for each $f' \in \mathcal{F}_i(h_i)$ with minimum price $\lambda_{f'}$ **do**

  $x_{f'}(t+1) \leftarrow \left[x_{h_i}(t+1) - \sum_{f \in \mathcal{F} \setminus f'} x_f(t+1) \right]^+$

end for

the relay price μ_f^β are calculated . After calculating the network price λ_f^α and the net relay price λ_f^β for each flow f, a peer determines the minimum price among all the incoming flows and uses this to calculate the incoming flow rate based on Eq. 39. In the last two steps of the algorithm, the rate of each flow $f \in \mathcal{F}_i(h_i)$ is adjusted based on this minimum price. The flows with higher prices will have their rates reduced and the flows with the minimum price will have their rates increased. The goal is to have equal price among all incoming flows for each receiver. This ensures that at optimality, the prices of each incoming flow of the receiver is minimum and therefore, the flow rate is optimum [70].

The implementation of the optimization algorithm for mesh-based topology is receiver-driven (i.e., receiver of a flow is the owner of that flow). Updating the link price of n requires the knowledge of old price and rate of all flows going through the link. In the star-topology, peer owning n maintains the link price μ_n^α. Since the receiver is the owner of a flow, a peer needs to receive message about the incoming rate from its parents to calculate the relay price μ_f^β. To update the rate, one must calculate λ_f^α and λ_f^β. The receiver of a flow receives μ_n^α from its parents and children and calculate λ_f^α. It also receives μ_f^β from its children and calculate λ_f^β for all of its incoming flows. Determining the optimum rate does not require further message passing between a parent and a child.

A.3 Summary

In summary, the price update algorithms presented in this thesis require no messaging overhead since the bookkeeper can collect all the necessary information locally to compute the update. The flow rate update process needs one message from the receiver of the flow to its sender (i.e., from child peer to parent peer). Since such a message can be blended into existing traffic between parent and children peers, such as heartbeat message or acknowledgment message in transmission protocol (e.g., TCP or RTCP), the messaging overhead can be reduced significantly.

B Simulation

This section presents the simulation results. First, the simulation setup is described. Results are presented for both single and double pricing solutions on three key aspects: convergence speed, aggregate PSNR, and link capacity utilization.

B.1 Video Adaptation

Each relaying peer must be able to adapt the quality of the video to fit into the receiving rates of its child peers. In this simulation, each peer performs transcoding by adjusting the quantization value of the video. The transcoding technique chooses the highest quantized rate that is less than the receiving rate achieved by the rate allocation solution.

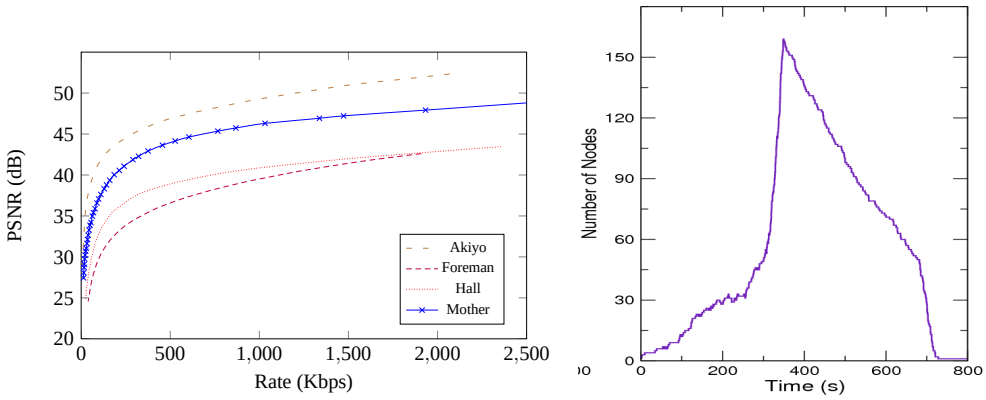
Definition 4. *Let x'_f be the optimal receiving rate for flow f calculated by the distributed algorithm denoted as $x_q = \{x_q \mid x_q < x_{q+1}, 1 \leq q \leq 51\}$ [71] as the video encoding rate with quantization value q. The actual relay rate will then be:*

$$x_f = \{x_q | x_q \leq x'_f < x_{q+1}\}$$

The open-source software x264 [71] is used to encode videos with different quantization values. The benchmark test sequences used for quality comparison of the transcoding simulation are the ITU-T test sequences [72] *foreman, akiyo, hall,* and *mother-daughter*, each having 300 frames with CIF resolution. The PSNR-rates for these videos are given in Fig. 13a.

B.2 Topology

A simple topology construction mechanism is used. However, the double-pricing algorithm is topology-independent and works with any topology construction mechanism. A real-time MSN video trace data [73] is used to construct the mesh. The traces provide



(a) Video transcoded using x264 [72]

(b) Number of nodes joining and leaving the network

Figure 13: ITU video sequences and peer churning in a P2P network

the join/leave time-stamp and the uplink bandwidth of each individual peers. The server bandwidth is assigned to 2 Mbps. A new update round begins every 0.1 second (i.e., members update their flow rate at every 0.1 second). For the sake of simplicity, multiple peers joining the network at any instant are queued and added at the beginning of each round.

Definition 5. *Let* capacity-coefficient *be the spare uplink capacity of a peer that can be used to allocate a new child peer and are calculated by each peer* $h \in \mathcal{H}$ *as:*

$$s_h = \left[z_h - \sum_{f \in \mathcal{F}_o(h)} x_f \right]_0^{x_h} \tag{42}$$

In the case of mesh topology, peers also calculate their link-ratio[3] *to decide if they should seek multiple parents. Formally, this ratio is defined as:*

$$r_h = \frac{x_h}{\sum_{f \in \mathcal{F}_o(h)} x_f} \tag{43}$$

A peer with $r_h < 1$ *implies that it dedicates more bandwidth to its children than it receives and vice-versa.*

[3]The server has a link-ratio of 0. It can be thought of as the seed in a torrent network

Implementation

The following procedures summarize the overlay construction process for both mesh and tree-based network:

- **Link Update**: At the end of each flow-rate update, a peer updates its link-ratio (for mesh topology) and capacity coefficient. Peers then send this information to their parents.

- **ID Peer**: If child peers exist, a parent peer decides the best candidate to be a parent based on its own coefficient and the coefficients of its children. It then sends this information to its own parents. This information eventually reaches the server.

- **Join**: A new peer sends a join request to the server. After receiving the prospective parent id, the new peer joins the network.

- **Request Parent**: In the case of mesh topology, a peer with $r_h < 1$ sends requests for parents with $r_h > 1$ (i.e., a peer that receives more than it gives). The server then replies with an appropriate parent ID. The parent ID is generated such that it does not create a cycle in the network. In the event a cycle exist, the requesting peer repeats the process.

During the simulation, the maximum number of incoming/outgoing flow is set to 4. However, the solution is independent of the number of flows a peer can have. During the course of the simulation, the dynamic joining process may lead to different overlay topology for the single pricing and double pricing solutions. In order to ensure fairness and that both solutions use the same overlay configuration, the overlay used to simulate the double-pricing solution is also used to simulate the single-price solution. Therefore, in the case of single-pricing solution, although the overlay is constructed gradually, the ID of the parent for a new peer joining the network is predetermined.

B.3 Single Pricing vs. Double Pricing Example

As mentioned at the end of Sec. A, a double-pricing solution involves both network and relay prices. Alternately, this problem can also be addressed by a single-pricing solution, which only considers the network constraint in Eq. 27. It first optimizes the rate allocation of all peers by treating them as independent flows competing for bandwidth and then imposes the relay constraint of Eq. 29 if the receiving rate of child peers exceed the receiving rate of its parent peer.

Example 5. *Consider the example in Fig. 12a. The uplink bandwidth of the hosts* h_1 *through* h_5 *is set to* 2.00, 1.72, 0.86, 1.60, *and* 1.62 *Mbps. Bandwidth is assigned based on the rate-distortion data points of the* foreman *video in the transcoding sequence. After running both double-pricing and single-pricing simulation, the final aggregate rates are* 3.54 *Mbps and* 3.26 *Mbps respectively.*

The reason double-pricing outperforms the single-pricing solution because it assigns more bandwidth to peer h_1 than h_2. This is because h_1 has two children, which raises its relay price, while the network price for both h_1 and h_2 stays the same. In single-pricing solution, the relay price is ignored. Therefore, the algorithm assigns the same rate for both h_1 and h_2 and causes h_1's children to suffer from low source rate. Furthermore, h_1's uplink bandwidth remains greatly under-utilized.

B.4 Results

This section compares results of single and double-pricing solutions and also provides analytical insight into convergence.

Convergence

The rate-convergence setup consists of a P2P network of 30 peers. The server peer has a maximum uplink capacity of 2 Mbps. Each peer joins the network after every 200000

iterations. For the single-pricing solution, data constraint is applied after every 100000 iterations. Following trial and error, the initial value of μ^{α} and μ^{β} is set to 0.5. The initial rate is set to 1 Mbps.

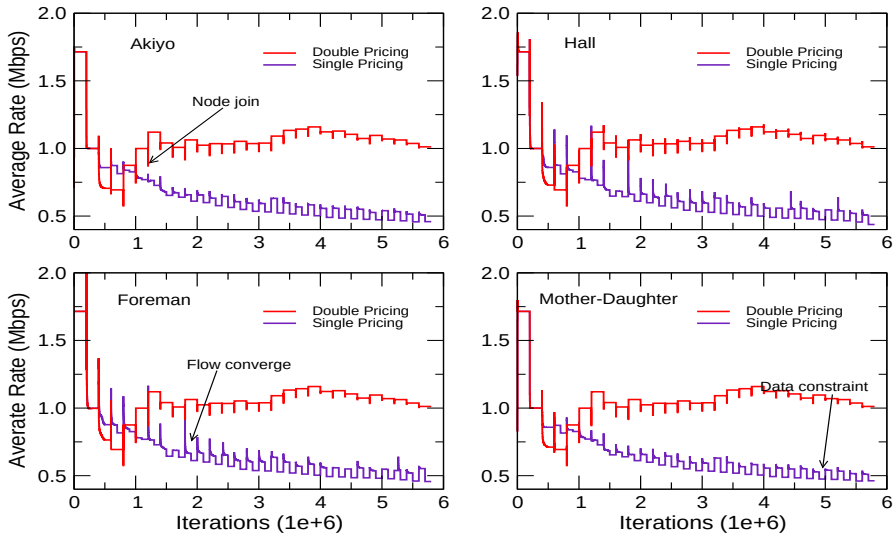


Figure 14: Rate convergence as peers join the multicast tree

Fig. 14 shows the rate convergence of the standard ITU video sequences used for benchmarking. At the beginning, it takes more iterations for the rates to converge. This is due to the fact that initially the price value of λ^{α} and λ^{β} are assigned to 0. However, as more peers join the network, the price value stabilizes to the optimum point and it takes less number of iterations for the price to move from old optimum point to a new optimum point.

The step size also influences the number of iterations required for rate convergence. Fig. 15 illustrates this width with different step sizes. An increase in step size from 0.0003 to 0.03 dramatically improves the number of iterations required for converge. Fig. 16 shows this improvement in terms of percent change during each iteration. After a few thousand iterations, the rate of change becomes insignificant. Fig. 17 shows the effect of changing the step size on average rate and PSNR value. The difference is negligible for

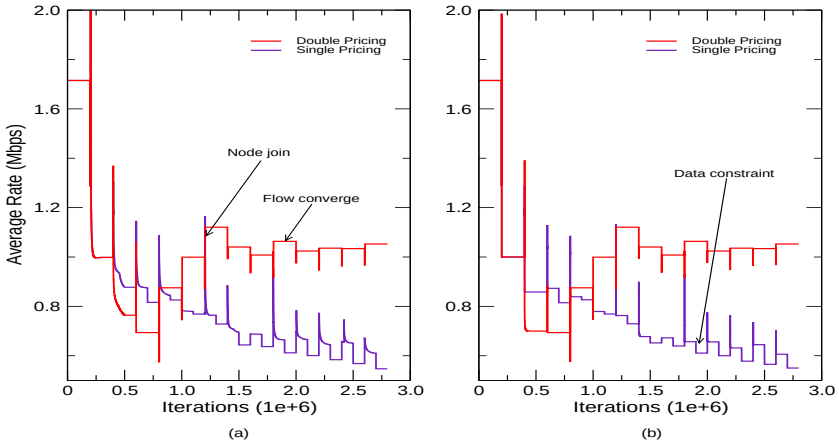


Figure 15: Effect of step size on rate convergence (a) step size = 0.0003 and (b) step size = 0.03

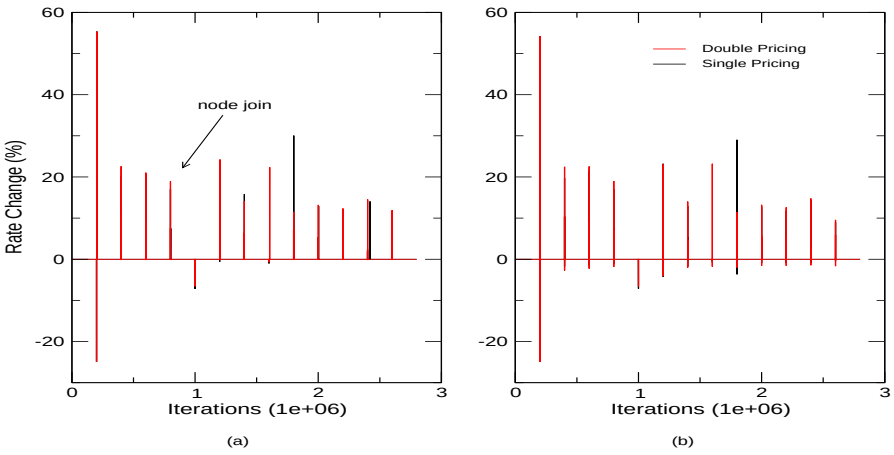


Figure 16: Rate change during iterations for step size (a) 0.0003 and (b) 0.03

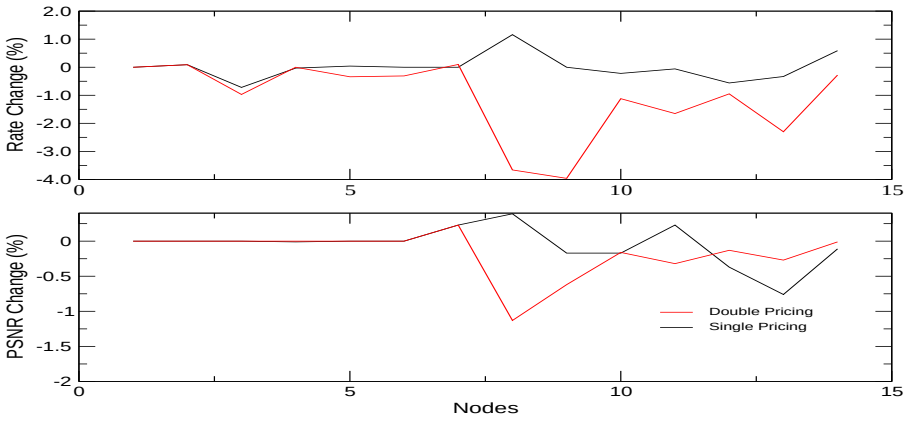


Figure 17: Performance difference for step size 0.03 with respect to step size 0.0003 with 100000/peer join event

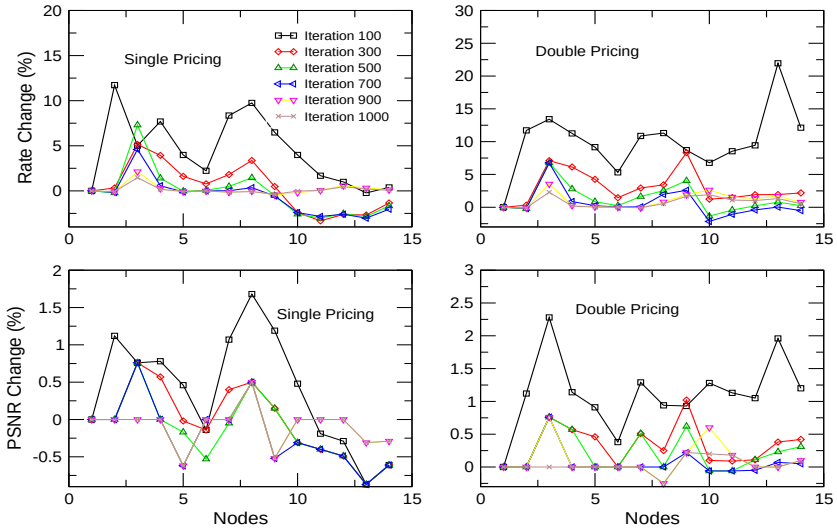


Figure 18: Rate and PSNR value difference for various number of iterations and 0.03 step size with respect to 0.0003 step size and 100000 iterations

step size 0.03 with respect to the step size 0.0003. The average rate change is less than 4% and the average PSNR change is within 1% of the optimal value when used with an step size of 0.03. In Fig. 18 shows the effect of the number of iterations used. It compares the deviation in average rate and PSNR for step size 0.03 with step size 0.0003. Even though it takes almost 100000 number of iterations to reach an optimal rate, 99% of the optimal rate is reached within 1000 iterations. Furthermore, for both solutions, the average rate value reaches 95% of the the optimal rate within 700 iterations. For the same number of iterations, the PSNR value reaches 99% of the optimal PSNR.

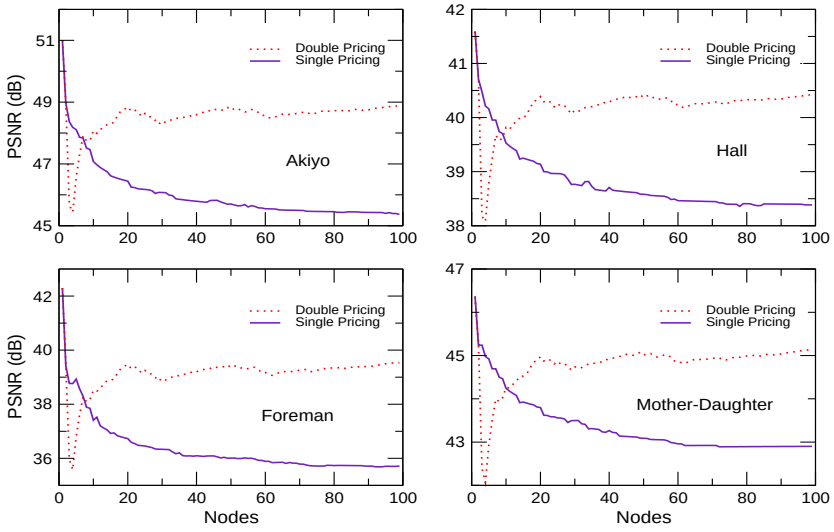


Figure 19: Double pricing solution vs. single pricing solution for transcoded video

Tree-based P2P

Fig. 19 shows the average PSNR value for the transcoded videos for a network of up to 100 peers. The average PSNR gain for the double-pricing solution over all the transcoded videos is 2.03 dB. It also maximizes the uplink bandwidth utilization of peers as shown in both Fig. 20. For all of the transcoded videos, the average link utilization over all the peers is 95% for the double-pricing solution compared to 76% for single-pricing solution.

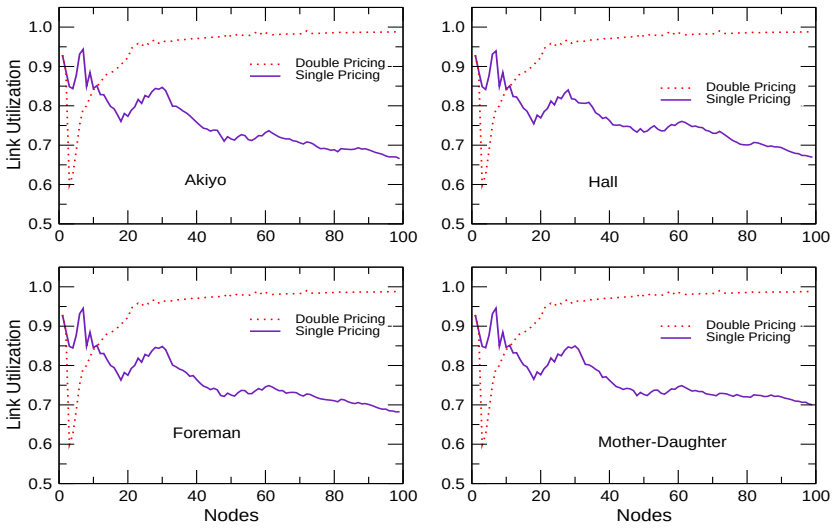


Figure 20: Link utilization comparison for transcoded video: utilization varies from 0 to 1 (equivalent to 0 from 100)

Mesh-based P2P

For mesh-based algorithm, peer churning is simulated based on the peer joining/leaving event in Fig. 13b. Instead of simulating an event at every 0.1 seconds, the real-time peer joining/leaving time-stamp is used to help construct the mesh topology. Fig. 13b shows the number of nodes in the network at any point in time over an 800 seconds interval. New peers stop joining the network at approximately 400 seconds and eventually all the peers leave the network. Fig. 21 shows the average PSNR value for the transcoded video sequences during the interval. Fig. 22 shows the average rate over all the flows in the network. The rate and the PSNR value drops to 0 at around 700 seconds, as the number of peers (excluding the server) in the network become 0. The results show that the double-pricing solution consistently performs better than the single-pricing solution.

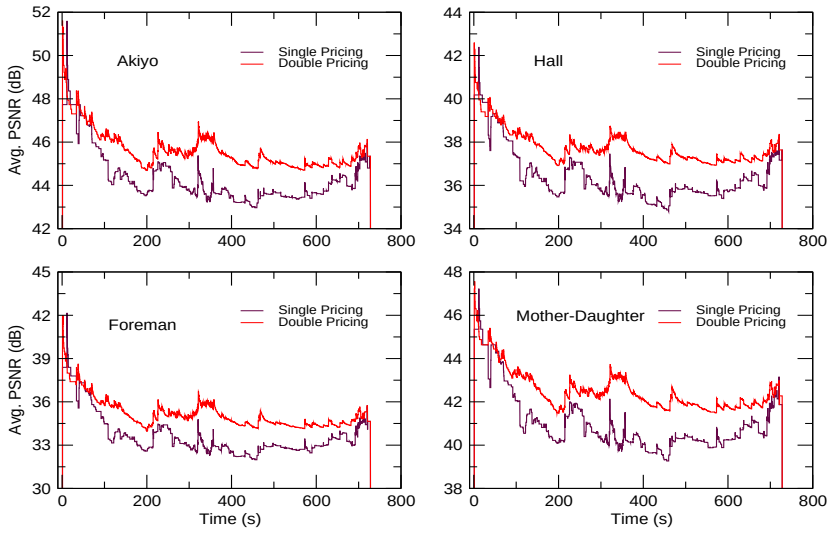


Figure 21: PSNR value comparison of the double-pricing and single-pricing solutions for transcoded video

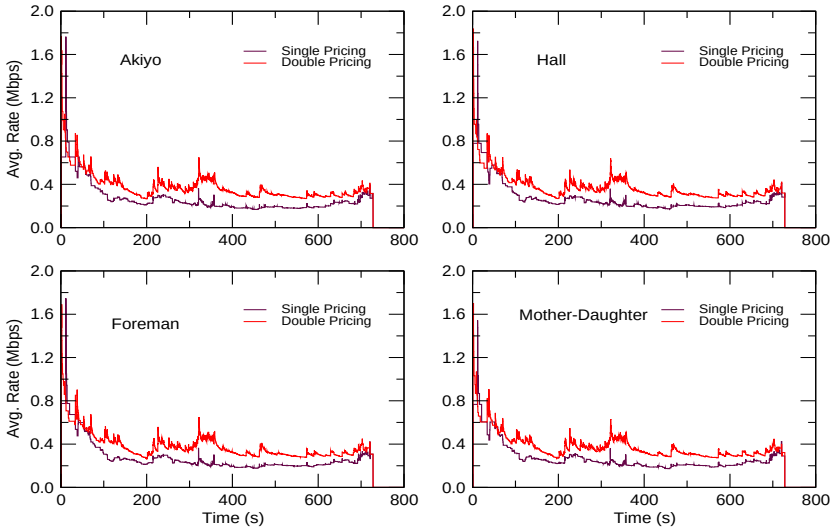


Figure 22: Rate comparison of the double-pricing and single-pricing solution

CHAPTER V

SCALABLE STREAM: HEURISTIC OPTIMIZATION

In this chapter, a heuristic-based optimization algorithm for scalable video is presented [74]. For a given topology, it attempts to optimally allocate layers among all peers. The algorithm focuses on two key metrics to achieve close to optimal layer allocation. Specifically, the heuristic algorithm focuses on the *load-balancing* and *weighted-layer allocation* techniques to achieve optimum allocation among peers in the network. In load balancing, a child peer evenly distributes the layer requests among all available parents. This allows a parent peer to serve multiple child peers. In weighted layer allocation, a parent prioritizes layer allocation to its children based on the cumulative number of descendants of a child.

A Distributed Algorithm

A high level description of the algorithm is now given followed by a detailed discussion of various aspects of the algorithm. A child initially inquires about the number of layers available from a parent peer. It then generates a valid combination of layers, ranks the layer combinations, and sends them to the respective parents. After receiving a layer allocation request from its child, a parent allocates a combination of layers that maximizes its uplink bandwidth utilization. The layer allocation is based on the ranking information provided by its children.

A.1 Layer Combination Generation

Each peer generates unique layer combinations for its parent based on the available layers information received. For a peer h_j with parents h_i having γ_i number of layers, there exist a total of 2^{γ_i} layer combinations h_j can request from h_i. These combination vectors are collected in a set $M^{h_i} = \{\delta_i^m \,|\, m = 1,2,\ldots,2^{\gamma_i}; \delta = [\lambda_{ij}^1, \lambda_{ij}^2, \ldots, \lambda_{ij}^l, \ldots, \lambda_{ij}^{\lambda_i}]\}$. The vector

Algorithm 2 Layer-Gen: Valid Layer Generation Algorithm

1: **For each end host** $h_j \in \mathcal{H}$
2: $\mathcal{P} = \emptyset$
3:
4: **for all** $\omega \in \Omega_i$ **do**
5: $\Omega_i \leftarrow \Omega_i - \omega$
6: matched = 1, totalLayers = 0
7: **for all** $\delta_i \in \omega$ **do**
8: **if** $\sum_{l=1}^{\gamma_i} x_i \lambda_{ij}^{l} \leq c_i$ **then**
9: $\omega \leftarrow \omega - \delta_i$
10: **end if**
11: **end for**
12: $l = 0$
13: **while** (**true**) **do**
14: numParents = 0, layers = 0
15: **for all** $\delta_i \in \omega$, **where** $\delta_i \in M^{h_i},\ h_i \in \mathcal{H}(h_j)$ **do**
16: **if** $l <$ γ_i **then**
17: layers += λ_{ij}^{l}
18: numParent ++
19: **end if**
20: **end for**
21: **if** layers > 1 **or** l >= totalLayers + 1 **then**
22: matched = 0
23: **break**
24: **end if**
25: **if** numParents == 0 **then**
26: **break**
27: **end if**
28: totalLayers += layers
29: l ++
30: **end while**
31: **if** matched == 1 **then**
32: $\mathcal{P} \leftarrow \mathcal{P}$ + ω
33: **end if**
34: **end for**

δ is a binary value vector that determines the presence of a layer between a parent and a child as mentioned in Eq. 8. For each peer h_j, let $p = |\mathcal{H}_i(h_j)|$ be the number of parents, where $h_i \in \mathcal{H}_i(h_j)$. Define $\Omega_j = (M^{h_1} \times M^{h_2} \times \ldots \times M^{h_i} \times \ldots \times M^{h_n})$ be an p-tuple, where each element set $\omega \in \Omega = \{\delta^1 \in M^{h_1}, \delta^2 \in M^{h_2}, \ldots, \delta^n \in M^{h_n}\}$ is a set of possible layer combination vectors from its parents. Algorithm 2 generates valid layer combinations for a peer h_j based on its parent set $\mathcal{H}_i(h_j)$. The **for loop** in line 7 eliminates any layer combinations that violate the capacity constraints of a parent peer mentioned in Eq. 10. The condition in line 23 eliminates discontinuity of Eq. 12 and duplicity of Eq. 11 in the layer combinations generated. The **for loop** in line 15 ensures that a layer received by a child must be present among its parents as mentioned in Eq. 13.

The resulting set $\mathcal{P}$ consists of all the valid layer combinations that satisfy the constraints from Eq. 10 - Eq. 11. Even though each parent h_i with γ_i number of layers generates 2^{γ_i} possible combinations, the total number of valid combinations of layers in set $\mathcal{P}$ is significantly smaller after satisfying all the constraints.

A.2 Receiver-Side Optimization

After generating all the valid layer combinations in $\mathcal{P}$, a child peer h_j then uses *rate maximization* and *load balancing* methodology to rank the layer combinations.

Definition 6. *A rank r is assigned to each* ω*. Each layer combination* $\delta \in \omega$ *also carries this rank value. The set of empty layer combination vectors is* $\Theta \subseteq \omega$*, where* $\omega \in \mathcal{P}$ *and* $\Theta = \{\delta \mid \delta = \emptyset\}$.

The set $\mathcal{P} = \{\omega_1, \omega_2, \ldots\}$ is ranked $\{r_1, r_2, \ldots\}$ based on rate maximization and load balancing.

Rate Maximization

In order to ensure that each peer receives maximum number of layers possible, a child peer ranks the elements of $\mathcal{P}$ such that the combination that delivers higher number of layers

has lower ranking compared to a combination that delivers lower number of layers.

Definition 7. *The rate-maximization condition can be expressed as:*

$$\sum_{\delta_i \in \omega_1} \sum_{l=1}^{\gamma_i} \lambda_{ij}^{l} \geq \sum_{\delta_i \in \omega_2} \sum_{l=1}^{\gamma_i} \lambda_{ij}^{l} \tag{44}$$

Load Balancing

A child peer contributes to the overall improvement of the average layer delivery by ensuring that it does not request all layers from a single parent. A balanced request by a child ensures that a parent does not carry the load of serving all the layers to a particular child.

Definition 8. *A balanced request is defined as the request with the minimum standard deviation of rates for a set of layer combinations among all the possible valid sets of combinations. Given* $\sum_{l \in \mathcal{L}(f_{ij})} \lambda_{ij}^{l} x_l$ *as a layer combination from a parent* h_i *to child* h_j*, the standard deviation of a valid set of layer combinations from one or more parents to a child is defined as:*

$$\sqrt{\frac{1}{|\mathcal{F}_i(h_j)|} \sum_{f_{ij} \in \mathcal{F}_i(h_j)} \left(\sum_{l \in \mathcal{L}(f)} \lambda_{ij}^{l} \cdot x_l - x_j^{\mu} \right)^2} \tag{45}$$

where x_j^{μ} *is the average rate for a particular set of valid layer combinations from all the layers received from all of the parents. Here* x_j^{μ} *is defined as:*

$$x_j^{\mu} = \frac{1}{|\mathcal{F}_i(h_j)|} \sum_{f_{ij} \in \mathcal{F}_i(h_j)} \sum_{l \in \mathcal{L}(f)} \lambda_{ij}^{l} \cdot x_l \tag{46}$$

Preference Matching

For a child peer, if all the layers received from parents have the same rank *r*, then there will be no duplicate layers. The parents initially attempt to accommodate the layers with the lowest rank to its children. A child receiving layers with higher rank implies that the parent is capacity limited and/or has other children with lower link ratio i.e., receives

higher priority in layer allocation decision.

In this case, a child determines a layer combination rank that satisfies the existing layer-rate allocation constraints by its parents. A parent only allocates a requested layer combination if the bandwidth required to service all the layers in the combination is less than the bandwidth allocated for that particular child.

A.3 Sender-Side Optimization

Maximizing the total number of layers received by peers across the network depends on the proper allocation of layers by each parent such that a child with fewer children receives fewer layers compared to another child. However, a child connected to fewer parent should receive more layers compared to a child with higher number of parents. In a mesh topology, a child can always receive more layers from any of its parents. Therefore, in order ensure fairness, a parent peer must consider both the incoming and outgoing degree of a child peer when allocating layers.

Weighted Layer Allocation

A parent peer uses cumulative weight of *link-ration* of its child to allocate layers.

Definition 9. *For a peer h_j, define the cumulative incoming and outgoing links as α_j and β_j:*

$$\alpha_j = |\mathcal{H}_i| + \sum_{h_k \in \mathcal{H}_o} \alpha_k \tag{47}$$

$$\beta_j = |\mathcal{H}_o| + \sum_{h_k \in \mathcal{H}_o} \beta_k \tag{48}$$

Therefore, the link ratio *for each peer h_j is:*

$$\ell_j = \eta \cdot \frac{\alpha_j - \beta_j}{\alpha_j + \beta_j + 1} \tag{49}$$

where η *is a proportionality constant that can be adjusted to assign priorities to children.*

A child peer with lower ℓ has higher priority in receiving its preferred layers from its parents. A parent then distributes its available uplink bandwidth based on this ratio. Fairness is ensured in this process because a peer with fewer child will receive fewer layers than a peer with more child.

Definition 10. *The total allocated bandwidth by a peer* h_j *having uplink capacity* z_j *to a child* h_k *is:*

$$x_{f_{jk}} = z_j \cdot \frac{\ell_k}{\sum_{h_i \in \mathcal{H}_o(h_j)} \ell_i} \tag{50}$$

However, this process could lead to starvation by parents when child peers receive no layers.

Definition 11. *The* link-ratio band κ *is defined for as a peer* h_j *having two child peer* h_i *and* h_k *with rank* r_i *and* r_k *having the following property:*

$$\lceil (\ell_i - \ell_k)/\kappa \rceil \geq r_i - r_k \tag{51}$$

where $\ell_i > \ell_k$.

Example 6. *If* $\lceil (\ell_i - \ell_k)/\kappa \rceil$ *is 2 and* r_k *is 1, then peer* h_i *can expect a layer combination assignment having rank at least 3. If* h_j *were to assign layer combination having rank 4 to* h_i *then* r_k *must be lowered to rank 2.*

A.4 Server-Side Optimization

Preemptive join is used to ensure that a peer with low uplink bandwidth does not prevent a peer with high uplink bandwidth from joining. Furthermore, preemptive join ensures that a peer with high uplink bandwidth stays at the top of the mesh.

Preemptive Join

A peer initially sends a *Join* request to the server. The requesting peer also sends its uplink capacity along with the request. The server acts as a prospective parent and checks whether the number of uplink connection has reached a maximum limit. If not, it accepts the new peer as child. If the limit has been reached, the parent further checks whether the offered uplink capacity is higher than the existing uplink capacity of any of its children. If the check is positive, the parent then preempts the lowest capacity child to accommodate the new peer. If the capacity is not higher, the parent then delegates the request to its children, whom, as prospective parent, performs the same bandwidth check and accommodates the new peer if the bandwidth is higher than the bandwidth of its existing children. The parents send a *Decline* message if the maximum number of connection has been reached, and the offered bandwidth cannot be accommodated.

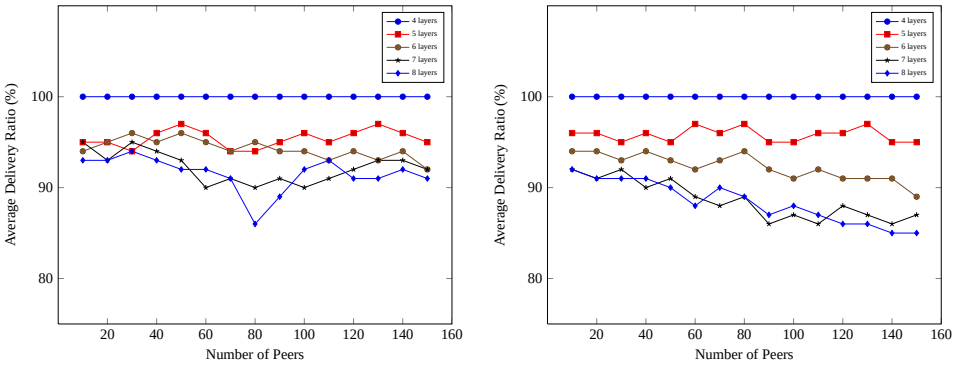
This recursive preemptive join ensures that a peer with lower uplink bandwidth does not bottleneck at the top of the mesh. The notations used in this chapter are collected in Tab. 5.

B Results

A streaming server is simulated with 8 layers, each having equal rate of 250 Kbps. The peer bandwidth is randomly assigned between 2 and 3 Mbps. The *in-degree/out-degree* ratio and the value of η for each peer is kept to 1. Fig. 23 compares the weighted layer allocation technique with simple proportional layer allocation. Fig. 24 compares the average delivery ratio for weighted and proportional layer allocation for various number of layers with 150 peers in the network. As the number of layers increase, the weighted layer allocation performs better than the simple proportional allocation. Fig. 25a shows the average layer delivery ratio without preemption. Fig. 25b compares the layer delivery with and without preemptive join. The performance is significantly better when the preemptive

Notation	Definition
γ_i	Number of layers peer h_i has
δ	Binary layer combination vector
M^{h_i}	Set of layer combination vector δ for peer h_i
Ω_i	n-tuple
$\omega \in \Omega$	Each element in n-tuple with layer combination from parents
$\mathcal{P}$	Set of valid layer combinations
$\Theta_i \subseteq \omega_i$	Set containing empty layer combinations
r	Ranks assigned to layer combination set
α, β	Cumulative incoming, outgoing links
ℓ, η	Link-ratio, proportionality constant
κ	Link-ratio band
s_h^c	Spare-capacity coefficient
s_h^b	Bandwidth coefficient

Table 5: Distributed Algorithm & Simulation Notations



(a) Weighted layer allocation scheme

(b) Proportional layer allocation scheme

Figure 23: Average layer delivery ratio

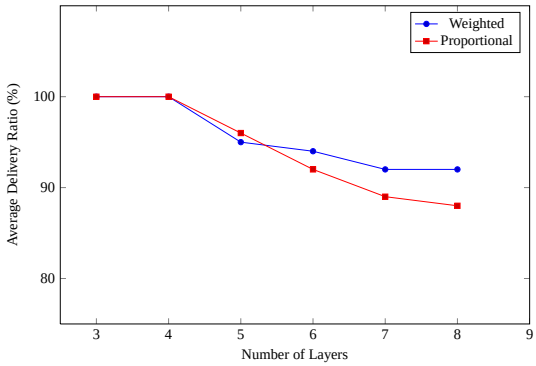
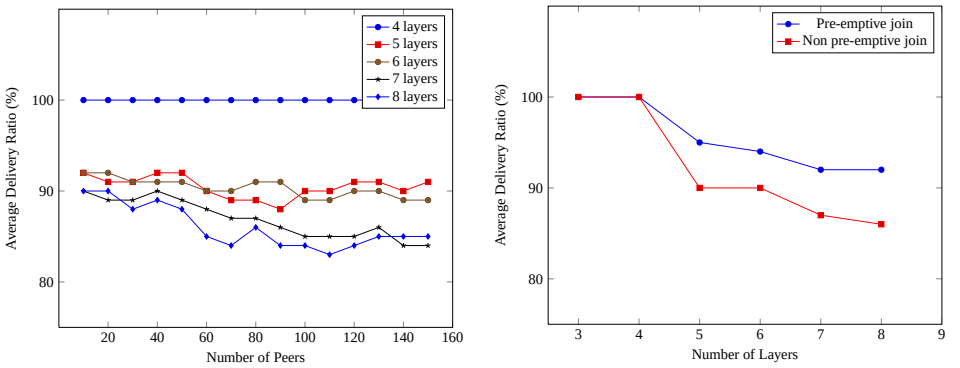


Figure 24: Average layer delivery ratio for different number of layers with 150 peers in the network

join mechanism is applied for topology construction.



(a) Layer allocation without preemption (b) Layer allocation with and without preemption

Figure 25: Average layer delivery ratio

CHAPTER VI

SCALABLE STREAM: MESSAGE-BASED OPTIMIZATION FRAMEWORK

In this chapter, message-based optimization framework for scalable video stream is introduced. Specifically, a code-based framework for optimization is presented that uses the *sum-product algorithm* as the basis for iterative update of messages and layer-allocation decision.

The framework uses *network codes* [61], which is defined as a state-space realization of network behavior that describes a set of connections in a network. This thesis extends the definition of network codes to capture the constraints of layered video in a P2P network. The rest of the chapter is organized as follows. First a brief history of network codes are presented followed by description of the state-space and normal realization of network codes and an introduction to generic sum-product based message-passing algorithm. Next, a description of message-passing optimization algorithm is presented for both single-layer and multi-layer video streams. After presenting the complexity analysis for the multi-layer algorithm, simulation results are presented. A thorough treatment of the state-space and normal realization on a graph can be found in the seminal work by Forney [75].

A Factor Graphs and State-space Realization

A factor graph is a bipartite graphical representation of the structure of a global function factored into a product of several local functions, which themselves depend on a subset of the global variables.

Example 7. *Let $w(x_1,x_2,x_3)$ be a global function of three variables that can be factored into two local functions $\mathcal{C}_1$ and $\mathcal{C}_2$. Fig. 26 shows a factorized representation of w. Based*

on this factor graph, w can be mathematically written as a product of the local functions:

$$w(x_1,x_2,x_3) = \mathcal{C}_1(x_1,x_2)\,\mathcal{C}_2(x_2,x_3)$$

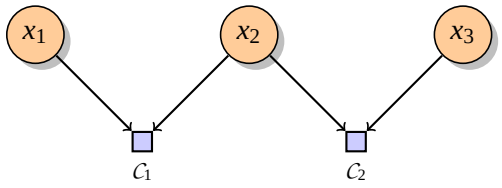


Figure 26: Example of factor graph representation of $w(x_1,x_2,x_3)$

Therefore, a factor graph has a variable node for each variable and a factor node for each local functions. An edge connects a variable node to a factor node if and only if the variable is an argument of the function. Factorization can be generalized for a global function w with a set of local variables $x = \{x_i,\, i \in I_n\}$ factored into a set of local functions $\mathcal{C} = \{\mathcal{C}_j(x_j),\, j \in I_m,\, x_j \subseteq x\}$, where each local function depends on a subset of the local variables:

$$w(x) = \prod_{j \in I_m} \mathcal{C}_j(x_j) \tag{52}$$

A.1 Marginal Function

Let $x_1,x_2,\ldots,x_n$ be a set of local variables such that for each $i \in I_n$, x_i takes values from a domain $\mathcal{A}_i$. Let $w(x_1,x_2,\ldots,x_n)$ be a global function of the variables such that the domain of w is a Cartesian product of the domain of each variable:

$$\mathcal{A} = \prod_{i \in I_n} \mathcal{A}_i \tag{53}$$

This is called the *configuration-space* of w. Assuming that the co-domain of w is well defined, there exist n marginal functions $w_i(x_i)$ associated with w. A marginal function

associated with variable x_i can be evaluated for value $a \in \mathcal{A}_i$ by summing w over all configuration of the variables with $x_i = a$:

$$w_i(a) = \sum_{\substack{x_j \\ j \in \{I_n \setminus i\}}} \sum_{\substack{x_j = a_j \\ a_j \in \mathcal{A}_j}} w(x_1, \ldots, x_{i-1}, a, x_{i+1}, \ldots, x_n) \tag{54}$$

In the example of Fig. 26, the marginal associated with $x_2 = a$ is:

$$w_2(a) = \sum_{x_1 \in A_1} \sum_{x_3 \in A_3} w(x_1, a, x_3)$$

A.2 Probability Marginal Function

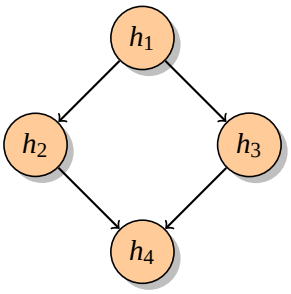
Factor graphs can also be used to represent joint probability mass function. Consider a set of independent input observations λ_i that are made on a set of variables $x = \{x_i,\, i \in I_n\}$, resulting in a set of output observations $y = \{y_i, i \in I_n\}$ and a likelihood vector $p(y \mid \lambda)$. The marginal product for each variable x_i is the component-wise product of the likelihood vector:

$$w_i(x) = \prod_{x_j \in \{x \setminus x_i\}} p(y_j \mid \lambda_j) \tag{55}$$

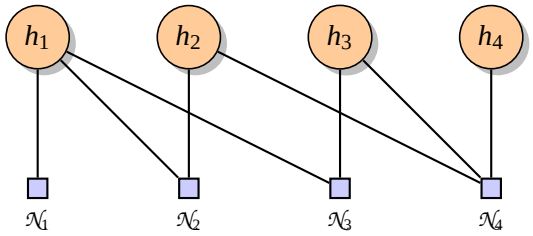
B Factor-Graph in Network Resource Allocation

Factor graphs can be applied to communication networks to represent the network connections. Consider a communication network with a set of peers $\mathcal{H} = \{h_i, i \in H\}$, each having link $n_i \in \mathcal{N}$. Each link in the network has a maximum capacity of z_n. The communication among the peers in a network states that a peer h_i is connected to its link n_i, which in turn is connected to a subset of peers that h_i wishes to communicate with.

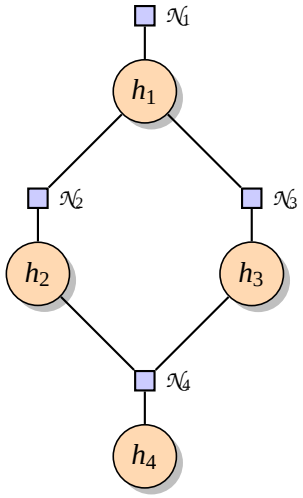
Example 8. *Fig. 27 illustrates the factor graph representation of a set of peers communicating with each other. Fig. 27b shows the bipartite factor graph representation of the*



(a) Peer communication example



(b) Factor graph representation



(c) Network representation of factor graph

Figure 27: Factor graph representation of a set of communicating peers

example in Fig. 27a and Fig. 27c shows the same factor graph in the form of a communication network.

B.1 State-space Realization: Variables and Codes

Variables have already been introduced in the context of factor graphs. A variable can be of type symbol and state. A symbol variable A_i takes values $a_i \in \mathcal{A}_i$ in a symbol alphabet $\mathcal{A}_h$. Therefore, a *symbol-configuration space* $\mathcal{A}$ is a Cartesian product:

$$\mathcal{A} = \prod_{i \in I_n} \mathcal{A}_i$$

where I_n is discrete index set of symbol alphabets. The elements of $\mathcal{A}$ are denoted by $a = \{A_i, i \in I_n\} \in \mathcal{A}$. Similarly, a state variable S_f takes values $s_f \in \mathcal{S}_f$ in a state alphabet $\mathcal{S}_f$. Therefore, the *state-configuration space* $\mathcal{S}$ is a Cartesian product of the state alphabets:

$$\mathcal{S} = \prod_{f \in I_m} \mathcal{S}_f$$

The elements of $\mathcal{S}$ are denoted by $s = \{S_f, j \in I_m\} \in \mathcal{S}$. The difference between a symbol and a state variable is that symbol variables are used when transmitting over a channel, whereas state variables are used internally by the constraints and remain hidden.

A code $\mathcal{C}$ can now be defined as a subset $\mathcal{C} \subseteq \mathcal{A} \times \mathcal{S}$ in the symbol-state configuration space. It may be characterized as a set of configurations that satisfy a certain set of local constraints (i.e., local functions in the context of a factor graph), $\mathcal{C}(h)$ and therefore, defines a subset in the symbol-state configuration space:

$$\mathcal{C}(h) \subseteq \mathcal{A}(h) \times \mathcal{S}(h) = \left(\prod_{i \in \mathcal{A}(h)} \mathcal{A}_i \right) \times \left(\prod_{j \in \mathcal{S}(h)} \mathcal{S}_j \right)$$

The elements of a code $c \in \mathcal{C} = \{C_j, j \in I_C\}$ are called codewords consisting of state and symbol variables represented by C_j. Therefore, a code represents a local constraint associated with a set of variables. A code consists of a set of codewords, where each codeword is constructed from a Cartesian product of the associated state and symbol variables. The variables take values from a discrete index set.

Example 9. *Let code $\mathcal{C}$ represents a local constraint and is connected to a set of variables $A = \{A_1, A_2, A_3\}$. Each variable A_i takes values from the same alphabet $\mathcal{A}_i = \{0, 1\}$. Therefore, the configuration-space is of length 3 and the number of possible codewords are 8: $\mathcal{C} = \{000, 001, 010, \ldots, 111\}$*

The term *local constraint* and *code* are used interchangeably throughout this chapter.

This realization of code consisting of a set of local constraints that are applied to symbol, and state variables is called *generalized state-space realization* [75], where each local code $\mathcal{C}(h)$ is a subspace of the direct product of the symbol-state vector space. However, a generalized state realization is not suitable to properly represent a communication network.

B.2 Normal Realization: Codes on Network

In a generalized state-space realization, the graph must be bipartite, constraints and variables must be represented by vertices, edges are not labeled, and do not carry values. A proper communication network representation requires that edges are directed, and carry values. Hence, edges in a communication network must be represented by variables. Furthermore, a communication network must not necessarily be bipartite. Forney [75] proposed a normal realization of factor graphs that can be applied to communication networks to represent nodes and directed edges. Such conversion without the loss of functionalities associated with factor graphs allows us to apply the properties of codes, functions and marginals to a operations on a communication network. Graphs with normal realization are called normal graphs.

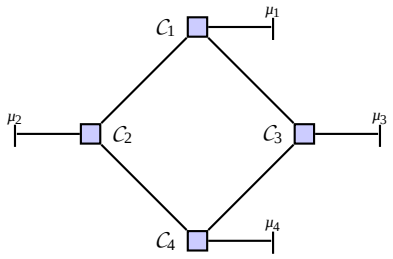


Figure 28: Normal realization of the example in Fig. 27a

In normal realization, each local constraint $\mathcal{C}_h$ is represented by a vertex. Each state variable S_f is represented by an edge between two constraints. Each symbol variable A_i is connected to one local constraint vertex, represented by a leaf-edge [75]. For a communication network, incident edges to a vertex represent inputs, while the remaining edges represent outputs. Fig. 28 illustrates the normal realization of the network presented in

Notation	Definition
w	Global function
$w(x)$	Cartesian product of local functions with $x_i \in x$
$w_i(x_i)$	Marginal over variable x_i
$w_i(x_i)$	Sum of all marginals over x_i
$\mathcal{A}$, $\mathcal{S}$	Symbol and state configuration space
$C_j \in c \in \mathcal{C}$	A code $\mathcal{C}$ consisting of set of codewords c consisting of a set of variables C_j
α_i, β_i	Incoming and outgoing message along edge i

Table 6: Notations used in factor graphs and marginal functions

Fig. 27a. In this case, each peer is converted to a constraint represented by vertices. The vertices are attached to a set of state variables that represent the incoming and outgoing edges associated with each peer. According to the definition of normal graph, each vertex is also attached to a symbol variable. The degree of each constraint c depends on the sum, $|\mathcal{A}(h)| + |\mathcal{S}(h)|$ of the symbol and state variables involved. Compared to a generalized state representation, state variables in a normal graph carry values.

The key notations used Sec. A, B, and C are collected in Tab. 6

C Sum-Product Algorithm

Sum-product is a powerful iterative decoding algorithm that operates by *message-passing* in a graphical model. The algorithm works by sending and receiving messages between a peer and its neighbors. Since all computation for the algorithm are done on the local constraint, it can be used in distributed computing. The sum-product name is derived from the fact that outgoing messages along each edge is a *sum* of the *marginal product* of all the incoming messages along the remaining edges. The sum-product algorithm is now described in the context of a normal graph.

C.1 Sum-Product of Messages

Consider a code $\mathcal{C}$ consisting of a set of codewords $c \in \mathcal{C}$. Each codeword consists of a set of variables $C_j \in c$. Each variable takes a set of values $c_j \in I_{C_j}$ from a discrete index set I. If codes $\mathcal{C}_x$ and $\mathcal{C}_y$ are connected by an edge, the outgoing message β_y from $\mathcal{C}_x$ is considered an incoming message α_x by $\mathcal{C}_y$.

Messages are sent and received along the edges for each value $c_j \in I_{C_j}$. For each variable C_j, let the incoming and outgoing messages for value c_j be defined as $\alpha_j(c_j)$ and $\beta_j(c_j)$ respectively. Therefore, the outgoing message along an edge is computed as:

$$\beta_j(c_j) = \sum_{c \in \mathcal{C}_j(c_j)} \prod_{C_i \in \{c \setminus C_j\}} \alpha_i(c_i) \tag{56}$$

where $\mathcal{C}_j(c_j)$ is the set of codewords consistent with c_j (i.e., variable C_j assumes the value c_j).

C.2 Sum-Product for Probability Decoding

The derivation of marginals for joint probability mass function has been discussed in Sec. A.2. The marginal in Eq. 55 can be used to compute probabilities in a sum-product algorithm. Following the description of codes in Sec. C.1, consider a set of input observations (i.e., codewords) $c = \{c_j \in I_{C_j}\}$ are made on all variables, resulting in a set of output observations $y = \{y_j \in I_{C_j}\}$ and a likelihood vector of $p(y_j \mid c_j)$. If all codewords are equiprobable, Bayes' theorem states that the *a posteriori probability* (APP) of $p(c \mid y)$ of a codeword $c \in \mathcal{C}$ is proportional to the likelihood vector:

$$p(c \mid y) = \frac{p(y \mid c)\, p(c)}{p(y)} \propto p(y \mid c), \qquad c \in \mathcal{C} \tag{57}$$

Therefore, the APP vector for a variable $p(C_j = c_j \,|\, y)$ is given by:

$$w_j(c_j) = p(C_j = c_j \,|\, y) = \sum_{c \in C_j(c_j)} \prod_{C_i \in \{c \setminus C_j\}} p(y_i \,|\, c_i) \tag{58}$$

$$\beta_j(c_j) = \sum_{c \in C_j(c_j)} \prod_{C_i \in \{c \setminus C_j\}} \alpha_i(c_i) \tag{59}$$

In this case, the output observation of the likelihood vector (i.e., equivalent to the set of incoming messages defined in Sec. C.1) are used to calculate the APP (i.e., equivalent to outgoing message along an edge) for each variable. Therefore, the sum-product algorithm for joint probability mass function involves computing the APP vector for every variables C_j over all values in the alphabet I_{C_j}.

D Codes for Scalable Video in P2P Overlay

This thesis proposes an extension of Forney's normal realization [75] of graph, to accommodate the relaying properties of peers in a P2P overlay and constraints of layered video.

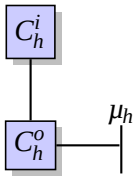


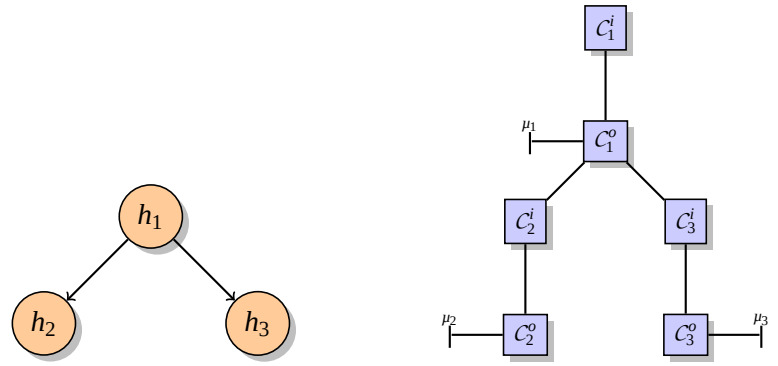
Figure 29: Normal definition of a peer

Definition 12. *Let peer h be a node in an overlay network having incoming and outgoing flows $\mathcal{F}_i(h)$ and $\mathcal{F}_o(h)$ respectively, where $\mathcal{F}(h) = \mathcal{F}_i(h) \cup \mathcal{F}_o(h)$. Since a state variable represents an edge in a normal graph, a flow connecting two peers will be represented by a state variable. The set of state variables representing incoming and outgoing flows are defined as $\mathcal{S}_h^i$ and $\mathcal{S}_h^o$ respectively, where $\mathcal{S}_h = \{S_f,\ f \in \mathcal{F}(h)\}$. A peer h will contain two codes (i.e., local constraints): incoming code $\mathcal{C}_h^i$ and outgoing code $\mathcal{C}_h^o$. The incoming code is connected to all the incoming flows and the outgoing code is connected to all the*

outgoing flows. A set symbol variables $\mu_h \in \mathcal{S}_h^{\mu}$ connected to the outgoing code $\mathcal{C}_h^o$ are used to represent the finite outgoing capacity of a peer in a overlay. The input and the output codes are connected by an internal edges. The internal edges are represented by a set $\mathcal{S}_h^l$ of internal state variables. The internal state variables here represent the relay constraints unique to a P2P network. With respect to the incoming code, the internal state variables represent the receipt of layers from parent peers, while for outgoing code, they represent the precondition that layers must be present before they can be allocated to child peers.

Fig. 29 shows the resulting normal realization of a peer in the context of a P2P overlay. The length of a codeword is directly related to the number of edges and the number of layers.

Example 10. *Fig. 30 shows an overlay network and its normal graph representation for scalable video.*



(a) Simple P2P overlay example

(b) Normal realization of P2P nodes

Figure 30: P2P overlay example and its normal realization

The key notations used throughout the rest of the chapter are collected in Tab. 7.

E Single-Layer Optimization

The single-layer optimization algorithm is presented here. First, a simplified optimization algorithm is presented, where each peer has a maximum of *unit uplink capacity*. The

Notation	Definition
$c^i \in \mathcal{C}^i$, $c^o \in \mathcal{C}^o$	Codewords associated with incoming and outgoing codes
$C_\mu^k \in C_\mu$	Variable vector associated with capacity
$C_l \in C_l$	Variable vector associated with received layers
$C_f^l \in C_f^l \in C_f$	Variable associated with the allocation of set of layers l in a flow f that part of a flow vector f
$\lambda_f^l = \{1,0\}$	Binary values assumed by variable C_f^l
$\gamma^l = \{1,0\}$	Binary values assumed by variable C_l
$\mu^k = \{1,0\}$	Binary values assumed by variable C_μ^k
σ	Capacity unit

Table 7: Notations used in sum-product algorithm optimization

algorithm is then be modified to consider the situation where peers can have uplink capacity that are multiple of the unit capacity.

Definition 13. *Let x_l be the rate required to deliver layer l and z_h be the uplink capacity of peer h. Since a scalable video stream requires the delivery of a complete layer for successful decoding, a **unit capacity** requires the ratio σ_h between the uplink capacity and the layer rate be* 1*:*

$$\sigma_h = \frac{z}{x_l} = 1$$

*In the case of **multi-unit capacity**, the ratio is defined as $\sigma_h > 1$. Throughout this chapter, it is assumed that the rate required for all layers are equal (i.e., $x_l = x_{l+1}$, $\forall\, l \in \mathcal{L}$) and constant. Therefore, uplink capacity of a peer h is defined to take value from the set of natural number, $\sigma_h \in \mathbb{N}_0$*

E.1 Algorithm: Unit Outgoing Capacity

Let $\mathcal{C}_h^i$ and $\mathcal{C}_h^o$ be the incoming and outgoing code for peer h. A codeword is generated by the Cartesian product of the variables associated the edges related to the code.

Definition 14. *The codewords associated with the incoming code is represented as the Cartesian product of state variables:*

$$\boldsymbol{c}^i = C_{\boldsymbol{f}} \times C_l = \left[\prod_{f \in \mathcal{F}_i} C_f \right] \times C_l \tag{60}$$

where $C_{\boldsymbol{f}} \in \mathcal{S}^i$ represents the set of variables associated with the incoming flows and $C_l \in \mathcal{S}^l$, $l \in \mathcal{L}$ represents the layer present on the internal edge between the incoming and the outgoing code. Similarly, the codewords associated with the outgoing code is represented as:

$$\boldsymbol{c}^o = C_l \times C_{\boldsymbol{f}} \times C_\mu = C_l \times \left[\prod_{f \in \mathcal{F}_o} C_f \right] \times C_\mu \tag{61}$$

where $C_{\boldsymbol{f}} \in \mathcal{S}^o$ represents the set of variables associated with the outgoing flows and C_μ is associated with the symbol variable that represents the outgoing capacity.

Since there is a single layer, the variables C_f, C_l, and C_μ can be represented by binary values.

Definition 15. *Let $\lambda \in \{0,1\}$ be the binary value assumed by C_f. Therefore, for peer h with having neighbor h_j connected by flow f:*

$$\lambda_f^l(h) = \begin{cases} 1 & \text{if layer } l \text{ is present in } f \\ 0 & \text{otherwise} \end{cases} \tag{62}$$

For incoming code $\mathcal{C}_h^i$, $\lambda_f^l(h) = 1$ implies that peer h receives layer l from parent peer such that $h(f) = h$. Similarly, for outgoing code $\mathcal{C}_h^o$, $\lambda_f^l(h) = 1$ implies that peer h allocates layer l to its child peer h_j such that $h \xrightarrow{f} h_j$.

Definition 16. *Let $\gamma \in \{0,1\}$ be a binary value taken by C_l:*

$$\gamma_h^l = \begin{cases} 1 & \text{if layer } l \text{ has been received by peer } h \\ 0 & \text{otherwise} \end{cases} \tag{63}$$

For incoming code C_h^i, $\gamma_h^l = 1$ implies that layer l is present in at least one of the incoming edges. For outgoing code C_h^o, $\gamma_h^l = 1$ serves as the precondition for allocating layers to outgoing edges.

Definition 17. *Let $\mu \in \{0,1\}$ be the binary value C_μ takes:*

$$\mu_h = \begin{cases} 1 & \text{if } \sum \lambda = 1 \\ 0 & \text{otherwise} \end{cases} \tag{64}$$

For outgoing code C_h^o, $\mu_h = 1$ implies that exactly 1 edge has been allocated a layer.

Definition 18. ***Admissible codeword*** *is defined as a codeword that satisfies the network properties and the scalable video properties. The network properties are:*

- ***Capacity Constraint****: Total allocated rate by a parent to a set of child does not violate the capacity (from Eq. 10).*
- ***Duplicity Constraint****: A layer is received from at most 1 parent (from Eq. 11). This increases the overall probability of receiving more layers by a peer by removing potential bandwidth waste.*

The layered video properties are:

- ***Relay Constraint****: A parent can only allocate a layer if it has received it (from Eq. 13).*
- ***Continuity Constraint****: Received layers must be consecutive (from Eq. 12). The properties of scalable video requires that a successful decoding of layer l depends on receiving all previous layers $1,2,\ldots,l-1$. If a peer receives layer 1, 2, and 4, it can only decode up to layer 2 because layer 3 is missing. Layer 4 in this case becomes useless.*

Definition 19. *Satisfying the capacity constraint requires the following condition to be true when generating a codeword for the outgoing code $\mathcal{C}^o$:*

$$\sum_{f \in \mathcal{F}_o} \lambda_f \leq \mu \tag{65}$$

In order to satisfy the duplicity constraint, admissible codewords for the incoming code $\mathcal{C}^i$ must satisfy the following condition:

$$\sum_{f \in \mathcal{F}_i} \lambda_f \leq 1 \tag{66}$$

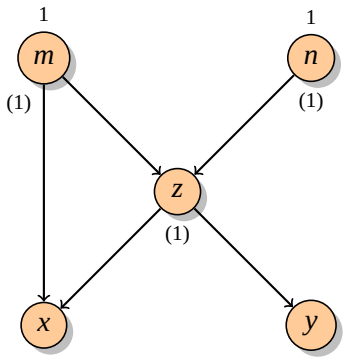
The relay constraint for the outgoing code $\mathcal{C}^o$ is satisfied by the following inequality:

$$\lambda_f \leq \gamma \tag{67}$$

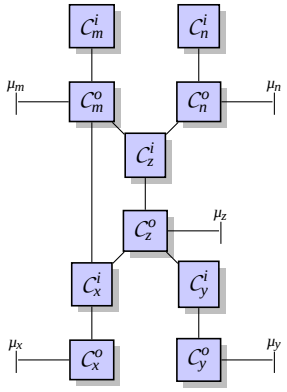
Since there is only 1 layer, the continuity constraint in Eq. 12 is not used. The following example illustrates the properties an admissible codeword must satisfy.

Example 11. *Consider the mesh network in Fig. 31a. The peers m and n are server peers having single layers and z is the relay peer. Both servers and the relay peer have unit uplink capacity Fig. 31b shows the normal-graph realization of this mesh network example. Fig. 32 shows the codewords for all the constraints associated with each peers.*

Fig. 32a shows the admissible codewords associated with the outgoing constraint $\mathcal{C}_m^o$ for peer m. The codewords are $\{C_l C_x C_z C_\mu\} = \{0000, 1101, 1011\}$. Here the codeword 0000 *represents the situation where peer m does not allocate layer to peer x or z. The codeword* 1101 *represents the situation where peer m allocates layer to peer x (i.e., $C_x C_z =$* 10*). Similarly, codeword* 1011 *represents the situation where peer m allocates layer to peer z (i.e., $C_x C_z =$* 01*). Since a layer must be present in order for m to allocate it, C_l must be* 1 *to satisfy the relay constrain in Eq. 13. Due to capacity constraint, allocation of a layer completely occupies the bandwidth of m. Therefore, C_μ must be* 1*. Note that* 1111 *is*



(a) Topology and initial layer allocation



(b) Normal graph representation

Figure 31: Example used for single layer optimization

not a valid codewords because providing layer to both children would violate the capacity constraint in Eq. 10.

Fig. 32c shows codewords for the incoming constraint C_z^i *of peer z. The admissible codewords are* $\{C_m C_n C_l\} = \{000, 011, 101\}$. *The codeword* 000 *means that peer z does not receive layer from neither m nor peer n. The codeword* 101 *implies that peer z receives layer from peer m but not from n. The value* $C_l = 1$ *in this codeword represents the fact that a layer has been received. Notice that* 111 *is not a valid codeword because it violates the duplicity constraint in Eq. 11. The continuity constraint in Eq. 12 is not addressed in the single layer case.*

Therefore, with the use of codes, it is possible to embed the network and the video constraints in the codeword-based symbol-state representation.

Admissible Codeword Generation

For a single layer, if an incoming code is connected to $F_i = |\mathcal{F}_i|$ number of flows, there are 2^{F_i} number of possible codewords. Similarly, for an outgoing code connected to $F_o = |\mathcal{F}_o|$ number of outgoing flows, there are 2^{F_o} number of possible codewords. Let A be an $M_A \times F_i$ and B be an $M_B \times F_o$ binary-value matrix that contains all possible binary combinations of

C_l	C_x	C_z	C_μ
0	0	0	0
1	1	0	1
1	0	1	1

(a) Outgoing constraint in $\mathcal{C}_m^o$

C_l	C_z	C_μ
0	0	0
1	1	1

(b) Outgoing constraint in $\mathcal{C}_n^o$

C_m	C_n	C_l
0	0	0
0	1	1
1	0	1

(c) Incoming constraint in $\mathcal{C}_z^i$

C_l	C_x	C_y	C_μ
0	0	0	0
1	1	0	1
1	0	1	1

(d) Outgoing constraint in $\mathcal{C}_z^o$

C_m	C_z	C_l
0	0	0
0	1	1
1	0	1

(e) Incoming constraint in $\mathcal{C}_x^i$

C_z	C_l
0	0
1	1

(f) Incoming constraint in $\mathcal{C}_y^i$

Figure 32: Creation of codewords for various constraints associated with each peer

length F_i and F_o respectively. The rows of the matrix are represented by A_m or B_m, while the columns are represented by A_f or B_f.

Algorithm 3 presents the admissible codeword generation process for incoming code. The **for loop** in line 8 counts the number of 1's in a row. The **if** condition at line 11 checks the duplicity constraint. If the sum ζ is greater than 1 from the loop in line 8, the row m is discarded as a potential codeword. C_l is set in line 14 according to ζ. Here $\zeta = 1$ implies that the peer has received the layer from at least 1 of its parents. Algorithm 4 presents the outgoing code generation process. The **if** condition in line 11 checks the capacity constraint. The outgoing capacity here is assumed to be 1. The value C_l in line 14 satisfies the relay constraint. A value of 1 implies that parent has allocated layer to at least 1 child and the parent must possess the allocated layer before it can relay it to a child. The value C_μ in line 15 refers to the uplink capacity used by a peer. A value 1 implies that the peer

Algorithm 3 Admissible Codeword Generation for Incoming Code

Initialize code: $\mathcal{C}^i = \emptyset$
Initialize matrix: $\boldsymbol{A}$
$\zeta = 0$

for each $m \in A_m$ **do**
  $\boldsymbol{c} = \emptyset$
  $\zeta = 0$
  for each $f \in A_f$ **do**
    ζ += a_{mf}
  end for
  if $\zeta > 1$ **then**
    continue
  end if
  $C_l = \zeta$, where $C_l \in c$
  for each $f \in A_f$ and $C_f \in C_{\boldsymbol{f}} \in \boldsymbol{c}$ **do**
    $C_f = a_{mf}$
  end for
  $\mathcal{C}^i \leftarrow \mathcal{C}^i + \boldsymbol{c}$
end for

has allocated layer to one of its outgoing flows.

Probability Update

The sum-product based probability decoding for layered video is now presented. The goal is to determine the probability with which a layer can be allocated by a parent and the probability with which a layer can be received by a child peer.

Definition 20. *The incoming message* α_j *for variable* $C_j \in \boldsymbol{c}$ *along an edge represents the probability of carrying a layer by that edge:*

$$\alpha_j = p\left(C_j = 1 \,|\, C_j = \lambda_j\right) \tag{68}$$

Since α *only carries the probability of* $\lambda_j = 1$*, the probability of variable* $C_j = 0$ *can be readily computed by* $1 - \alpha_j$*. Upon computing the marginal probability for each variable,*

Algorithm 4 Admissible Codeword Generation for Outgoing Code

Initialize code: $\mathcal{C}^o = \emptyset$
Initialize matrix: $\boldsymbol{B}$
$\zeta = 0$

for each $m \in B_m$ **do**
  $\boldsymbol{c} = \emptyset$
  $\zeta = 0$
  for each $f \in B_f$ **do**
    ζ += b_{mf}
  end for
  if $\zeta > 1$ **then**
    continue
  end if
  $C_l = \zeta$, where $C_l \in \boldsymbol{c}$
  $C_\mu = \zeta$, where $C_\mu \in \boldsymbol{c}$
  for each $f \in B_f$ and $C_f \in C_{\boldsymbol{f}} \in \boldsymbol{c}$ **do**
    $C_f = b_{mf}$
  end for
  $\mathcal{C}^o \leftarrow \mathcal{C}^o + \boldsymbol{c}$
end for

the outgoing message β_j *is normalized within a range of* 0 *to* 1*:*

$$\beta_j = \frac{p(C_j = 1)}{p(C_j = 1) + p(C_j = 0)} \tag{69}$$

Example 12. *Let us consider a code* $\mathcal{C}$ *consists of two codewords* $\mathcal{C} = \{101, 011\}$. *The incoming messages in this case are* α_1, α_2, α_3 *representing the probability* $p(C_1 = 1)$, $p(C_2 = 1)$, $p(C_3 = 1)$. *The probability of* $C_1 = \{1, 0\}$ *can be computer as:*

$$w(1) = p(C_1 = 1) = (1 - \alpha_2)\,\alpha_3$$

$$w(0) = p(C_1 = 0) = \alpha_2\,\alpha_3$$

Therefore, the outgoing message normalized within the range 0 to 1 is:

$$\beta_1 = \frac{p(C_1 = 1)}{p(C_1 = 1) + p(C_1 = 0)}$$

Similarly, the probability of $C_2 = \{1,0\}$ *is:*

$$w(1) = p(C_2 = 1) = (1 - \alpha_1)\,\alpha_3$$
$$w(0) = p(C_2 = 0) = \alpha_1\,\alpha_3$$

and the probability of $C_3 = \{1,0\}$ *can be calculated as:*

$$w(1) = p(C_3 = 1) = \alpha_1(1 - \alpha_2) + (1 - \alpha_1)\,\alpha_2 \qquad = 1$$
$$w(0) = p(C_3 = 0) \qquad = 0$$

The probability of $p(C_3 = 1)$ *and* $p(C_3 = 0)$ *are set to* 1 *and* 0 *respectively because there are no codeword representing* $C_3 = 0$*. Therefore, normalization of* β_3 *results in* 1.

Probability Initialization

At the beginning of the update algorithm, initial probability values are set along each edge.

Definition 21. *The initial probability message along the incoming and the outgoing edges related to the flows are set to 0.5. The probability assignment for the internal edge is set as:*

$$\alpha_l = p(C_l = 1) = \begin{cases} 1 & \text{if peer is a root and } l \text{ is present} \\ 0 & \text{if } l \text{ is not present for root peer} \\ 0.5 & \text{otherwise} \end{cases}$$

Here, $p(C_l = 1) = 1$ for a root peer indicates that the layer is definitely present. Sum-product update algorithm does not compute this probability for root peers since the probability is fixed. Similarly, if a root peer is known to not have a layer, the probability assigned in this case is $p(C_l = 1) = 0$.

Definition 22. *The probability value for variable C_μ is set as follows:*

$$\alpha_\mu = p(C_\mu = 1) = \begin{cases} 1 & \text{if } C_\mu = 1 \\ 0 & \text{otherwise} \end{cases}$$

Since the capacity value is assumed to be constant, the update algorithm does not update the probability values associated with C_μ.

Exit Condition

The exit condition determines the convergence point when the probability of carrying a layer from parent to child through an an edge reaches 1 or 0.

Definition 23. *Let ε be the threshold for convergence. Therefore, convergence on an edge with variable C_j is reached and the probability value is set to 1 or 0 if the following condition is satisfied:*

$$p(C_j = 1) = \begin{cases} 1 & \text{if } \quad 1 - p(C_j = 1) \quad \leq \varepsilon \\ 0 & \text{if } \quad p(C_j = 1) \qquad \leq \varepsilon \end{cases}$$

Algorithm

For each variable, the probability value α is determined by calculating the sum of the marginals and then normalizing it. This normalized probability is sent along the edges as β in response to each incoming α. Each peer waits to receive messages on all of its outgoing edges from its child peers and updates the probability on its internal edge. Based on the update on the internal edge, a peer then updates probability values on its incoming code and sends this to its parents. Similarly, upon receiving reply messages on all of its incoming edges, a peers recomputes the probability on its internal edge. A peer then updates the probability on all of its outgoing edges and sends this updated probability message to its children.

Algorithm 5 Sum-Product Algorithm on Outgoing Code

For each constraint $\mathcal{C}_h^o$

Input: t
Incoming Message Set $\mathcal{M}_o$

if t is odd round **then**
  for all $f \in \mathcal{F}_o$ **do**
    Wait for all in coming messages: $\mathcal{M}_o \leftarrow \alpha_f$
  end for
  if h.type != root **then**
    {update probability of the internal edge}
    Calculate $w_l(1)$
    Calculate $w_l(0)$
    Normalize β_l
    Send β_l to incoming code $\mathcal{C}_h^i$
  end if
else
  {even round now}
  if h.type != root **then**
    Wait for update on C_l
  end if
  for all $f \in \mathcal{F}_o$ **do**
    Calculate $w_f(1)$
    Calculate $w_f(0)$
    Normalize β_f
    Send β_f to all outgoing edges
  end for
end if

Definition 24. *The algorithm works on rounds $t = 1, 2, \ldots$. During the odd rounds, messages travel upstream from child to parent peers, passing through the internal edge. During the even round, messages travel downstream from parent to child peers, similarly passing through the internal edge.*

Algorithm 5 and Algorithm 6 summarizes the single-layer sum-product update algorithm for each round for outgoing and incoming codes respectively.

Example 13. *The following is an example of the sum-product message-passing algorithm based on the topology and constraints shown in Example 11. Fig. 33 and Fig 34 shows*

Algorithm 6 Sum-Product Algorithm on Incoming Code

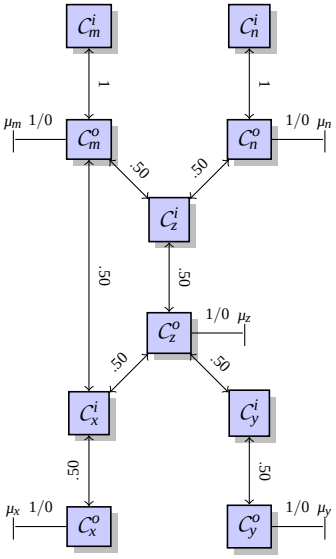
For each constraint C_h^i

Input: t
Incoming Message Set $\mathcal{M}_i$

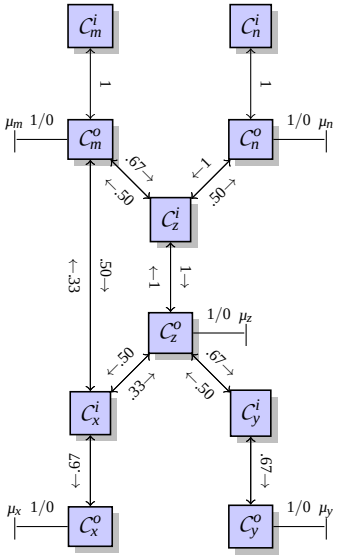
{odd round}
if t is odd round **then**
  if h.type != root **then**
    Wait for update on C_l
  end if
  for all $f \in \mathcal{F}_i$ **do**
    Calculate $w_f(1)$
    Calculate $w_f(0)$
    Normalize β_f
    Send β_f along all incoming edges
  end for
else
  {even round now}
  for all $f \in \mathcal{F}_i$ **do**
    Wait for all in coming messages: $\mathcal{M}_i \leftarrow \alpha_f$
  end for
  if h.type != root **then**
    {update probability of the internal edge}
    Calculate $w_l(1)$
    Calculate $w_l(0)$
    Normalize β_l
    Send β_l to outgoing code C_h^o
  end if
end if

the the rounds 1 *to* 8*. During the odd rounds, the messages travel upstream from child to parent, while messages travel downstream from parent to child during the even rounds. The messages converge after* 8 *rounds. In optimality, peer m gives layer to x, while peer n gives layer to peer z. Since x has already received layer from parent m, peer z relays the received layer to peer y.*

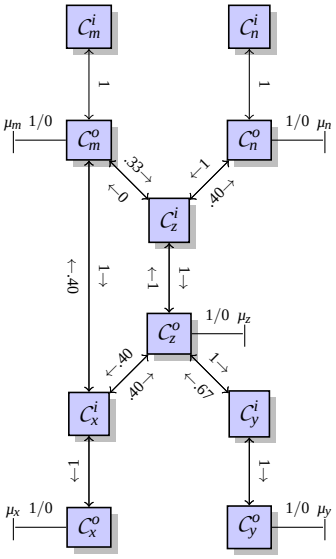
Example 14. *Fig. 35a shows another single layer optimization example. Fig. 35b and Fig. 36 shows message-passing for rounds* 1 *to* 4*. After round* 4*, the layer allocation converges to optimality. In optimal allocation, m does not allocate layer to x. Allocating*



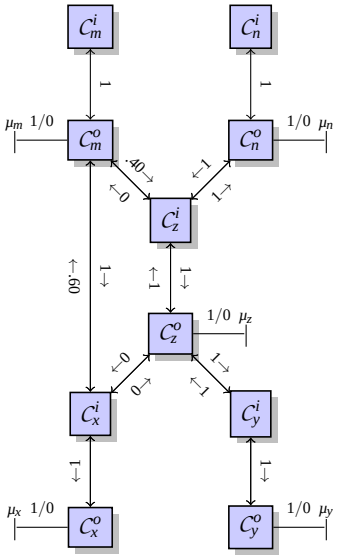
(a) initial probability



(b) Round 1 and 2: leaf to root and root to leaf

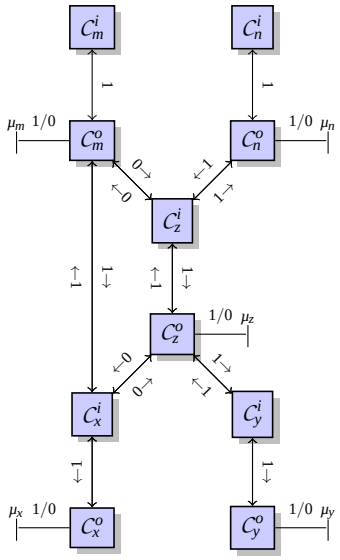


(c) Round 3 and 4



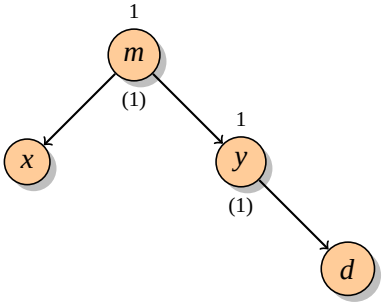
(d) Round 5 and 6

Figure 33: Message passing algorithm for single layer allocation: Topology example 1

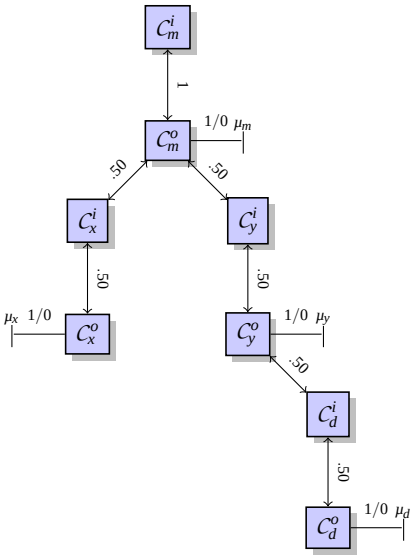


(a) Round 7 and 8: Converged

Figure 34: Message passing algorithm for single layer allocation: Topology example 1

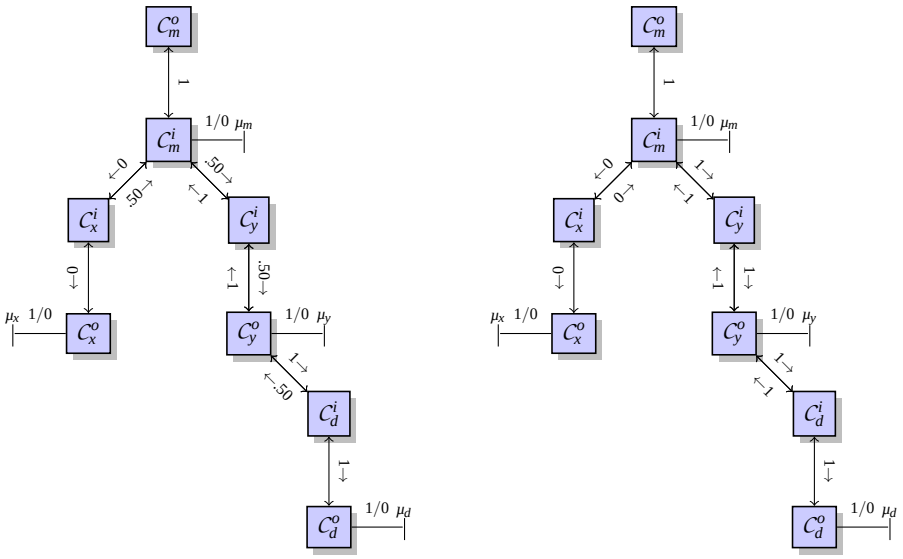


(a) Network example topology



(b) Normal graph representation and initial probability

Figure 35: Message passing algorithm for single layer allocation: Topology example 2



(a) Round 1 and 2 (b) Round 3 and 4: Converged

Figure 36: Message passing algorithm for single layer allocation: Topology example 2

layer to y allows peer y to relay the layer to child peer d. This improves the network-wide QoS. The allocation is influenced by messages traveled from d.

E.2 Generalized Single-Layer Optimization Algorithm

The previous algorithm for single-layer with unit capacity is now expanded to include peers with multi-unit capacity. Updating the algorithm involves adjusting the variable C_μ associated with the outgoing code $\mathcal{C}^o$.

Definition 25. *Let $\boldsymbol{c}^o$ be the codeword associated with outgoing code:*

$$\boldsymbol{c}^o = C_l \times C_{\boldsymbol{f}} \times C_{\boldsymbol{\mu}} = C_l \times \left[\prod_{f \in \mathcal{F}_o} C_f \right] \times \left[\prod_{k=1}^{\sigma} C_\mu^k \right] \tag{70}$$

where $C_{\boldsymbol{\mu}}$ is the Cartesian product of the symbol variables C_μ^k that represents the uplink

capacity. The variable C_μ^k can be assigned binary value μ^k based on the following:

$$\mu^k = \begin{cases} 1 & \text{if } k \leq \sum \lambda \\ 0 & \text{otherwise} \end{cases} \tag{71}$$

Admissible Code Generation

The admissible code generation process for incoming code remains unchanged in Algorithm 3. The outgoing code generation follows Algorithm 4 with the following modifications:

- Layers can be allocated up to σ
- The value of σ is determined by the uplink capacity of the peer and the total number of layers requested by child peers.

The outgoing code generation process begins by parent peers receiving layer requests from all children and summing up the number of layer requests. The modified outgoing code generation process is given in Algorithm 7.

Algorithm

As previously mentioned, the sum-product update algorithm is independent of the underlying constraints. The generalized algorithm for incoming and outgoing code follows Algorithm 5 and Algorithm 6.

Example 15. *The codeword construction technique for single-layer algorithm with multi-unit capacity is given here for Example 14 based on Fig. 35a. In this example, assume that peer m has a capacity of* 2 *instead of* 1. *The new codewords associated with the outgoing code $\mathcal{C}_m^o$ for peer m is given in Fig. 37. In optimal configuration, peer m allocates layer to both child x and y. This is possible because peer m now has a capacity of* 2.

Algorithm 7 Codeword for Outgoing Code: Single-Layer Multi-Unit Capacity

1: **Initialize code:** $\mathcal{C}^o = \emptyset$
2: **Initialize matrix:** $\boldsymbol{B}$
3: $\zeta = 0$
4: $\vartheta = 0$
5:
6: **for each flow** $f \in \mathcal{F}_o$ **do**
7: **if** layer_request$_f$ == **true then**
8: ϑ += 1
9: **end if**
10: **end for**
11: $\sigma = \min(\sigma_h, \vartheta)$
12: **for each** $m \in B_m$ **do**
13: $\boldsymbol{c} = \emptyset$
14: $\zeta = 0$
15: **for each** $f \in B_f$ **do**
16: ζ += b_{mf}
17: **end for**
18: **if** $\zeta > \sigma$ **then**
19: **continue**
20: **end if**
21: $C_l = (\zeta \geq 1)? 1 : 0$ where $C_l \in \boldsymbol{c}$
22: **for** $k = 1$ **to** σ **do**
23: $C_\mu^k = (k \leq \zeta)? 1 : 0$ where $C_\mu \in \boldsymbol{C_\mu} \in \boldsymbol{c}$
24: **end for**
25: **for each** $f \in B_f$ and $C_f \in \boldsymbol{C_f} \in c$ **do**
26: $C_f = b_{mf}$
27: **end for**
28: $\mathcal{C}^o \leftarrow \mathcal{C}^o + \boldsymbol{c}$
29: **end for**

F Multi–Layer Optimization

The single-layer optimization algorithm is now extended for the multi-layers. Here the root peers can have multiple layers. Furthermore, peers can have multi-unit capacity and can deliver multiple layers to its children.

Definition 26. *Let $C_f^{\boldsymbol{l}} \in \boldsymbol{C_f}$ be a new variable vector associated with each flow $f \in \mathcal{F}$. Let $C_f^l \in C_f^{\boldsymbol{l}}$ be the variable representing the presence or absence of layer l in flow f.*

C_l	C_x	C_y	$C_\mu^1 C_\mu^2$
0	0	0	0 0
1	1	0	1 0
1	0	1	1 0
1	1	1	1 1

(a) Outgoing constraint in $\mathcal{C}_m^o$

Figure 37: Codewords for outgoing code C_m^o for peer m in Fig. 35a with uplink capacity 2

Furthermore, let $C_{\boldsymbol{l}}$ be the new variable vector associated with internal edge. Let $C_l \in C_{\boldsymbol{l}}$ be the variable representing the presence of layer l. The codewords $\boldsymbol{c}^i$ associated with the incoming code $\mathcal{C}^i$ for multi-layered video stream is defined as:

$$\boldsymbol{c}^i = C_{\boldsymbol{f}} \times C_{\boldsymbol{l}} = \left[\prod_{f \in \mathcal{F}_i} C_f^{\boldsymbol{l}}\right] \times \left[\prod_{l \in \mathcal{L}} C_l\right] \tag{72}$$

The codewords $\boldsymbol{c}^o$ associated with the outgoing code $\mathcal{C}^o$ is:

$$\boldsymbol{c}^o = C_{\boldsymbol{l}} \times C_{\boldsymbol{f}} \times C_{\boldsymbol{\mu}} = \left[\prod_{l \in \mathcal{L}} C_l\right] \times \left[\prod_{f \in \mathcal{F}_o} C_f^{\boldsymbol{l}}\right] \times \left[\prod_{k=1}^{\sigma} C_\mu^k\right] \tag{73}$$

where $C_f^{\boldsymbol{l}}$ is a Cartesian product representing the presence of layers in flow f:

$$C_f^{\boldsymbol{l}} = \prod_{l \in \mathcal{L}} C_f^l \tag{74}$$

Definition 27. *Let $\lambda_f^l = \{0,1\}$ be the value taken by C_f^l. It is defined based on Eq. 62. Let Let γ^l be the binary value taken by C_l. For the outgoing code $\mathcal{C}^o$, it is redefined as:*

$$\gamma^l = \begin{cases} 1 & \text{if } \sum_{f \in \mathcal{F}_o} \lambda_f^l \geq 1 \text{ or } \sum_{f \in \mathcal{F}_o} \lambda_f^{l+1} \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{75}$$

For the incoming code $\mathcal{C}^i$, it is redefined as:

$$\gamma^l = \begin{cases} 1 & \text{if } \sum_{f \in \mathcal{F}_i} \lambda_f^l \geq 1 \\ 0 & \text{otherwise} \end{cases} \tag{76}$$

Admissible codewords must satisfy the network and layered video properties defined in Definition 18.

Definition 28. *In a multi-layer environment, the admissible codewords associated with the outgoing code $\mathcal{C}^o$ must satisfy the* **capacity constraint***:*

$$\sum_{f \in \mathcal{F}_o} \sum_{l \in \mathcal{L}} \lambda_f^l \leq \sigma \tag{77}$$

Satisfying the **duplicity constraint** *for incoming code requires the following condition:*

$$\sum_{f \in \mathcal{F}} \lambda_f^l \leq 1 \qquad \forall\, l \in \mathcal{L} \tag{78}$$

The **continuity constraint** *on the incoming can be embedded by satisfying the following condition when choosing a codeword:*

$$\sum_{f \in \mathcal{F}_i} \lambda_f^l \geq \sum_{f \in \mathcal{F}_i} \lambda_f^{l+1} \qquad \forall\, l \in \mathcal{L} \tag{79}$$

Admissible Codewords Generation

The generation of codewords for multi-layer video is now considered. For L number of layers, let B be a $M \times L$ binary value matrix consisting of all possible layer combinations that can be delivered to flow f. Here $M = 2^L$. The rows of the matrix are represented by B_m, while the columns are represented by B_l.

Admissible codeword generation associated with the incoming code must satisfy the video constraints for the binary matrix B. Algorithm 8 determines admissible codewords

for incoming codes.

Algorithm 8 Check Video Constraint

```
1: Input: vector C_l,  type : int
2: Output: type : int
3:
4: for i = 0; i < L; i++ do
5:     if C_l > 1 then
6:         return −1
7:     end if
8: end for
9: for l = L − 1; l > 0; l -- do
10:    if C_l < C_{l−1} then
11:        return −2
12:    end if
13: end for
14: return 1
```

Ensuring the capacity and relay constraints for the outgoing code is done based on Algorithm 9 and Algorithm 10.

Algorithm 9 Check Capacity Constraint

```
1: Input: vector C_f^l,  type : int
2: Input Capacity: z_f,  type : int
3: Output: type : int
4:
5: set ∑ = 0
6: for l = 0 to L do
7:     ∑ += C_f^l
8: end for
9: if ∑ > z_f then
10:    return 0
11: end if
12: return 1
```

Complexity Optimization for Outgoing Code

If there are F_o number of child peers, there are a possible M^{F_o} number of codewords to consider before selecting the admissible codewords. Here $M = 2^L$ for L number of layers.

Algorithm 10 Create Code Layers

1: **Initialize Matrix:** $\boldsymbol{B} = M \times L,\ M = 2^L$
2: **Input:** F_i, type : int
3: **Input:** vector $P_{\boldsymbol{f}}$, type : int
4: **Output:** vector $C_{\boldsymbol{l}}$, type : int
5:
6: set $C_{\boldsymbol{l}} = 0$
7:
8: **for** $f = 0$ **to** F_i **do**
9: $m = P_f$
10: **for** $l = 0$ **to** L **do**
11: C_l += B_{ml}
12: **end for**
13: **end for**

For a large number of layers and a large number of outgoing peer, computing sum-product update on so many codewords is computationally intensive.

Example 16. *If there are $L = 5$ layers and $F = 5$ outgoing peers, the number of possible codewords are $(2^5)^5 = 33554432$. Furthermore, computing the probability on each codeword requires $F \cdot L = 25$ multiplication operations.*

During the optimized codeword generation process, Admissible codewords associated with the outgoing code requires that if $C_\mu^k = 0$ and $k \leq \sigma$, then $p\,(C_\mu^k = 0) = 0$. This is because a parent peer always attempts to allocate the maximum number of layers. Therefore, marginals of any codeword that under utilizes the uplink capacity will result in 0. Therefore, during the code generation process, the only valid codewords are codewords that that maximizes the bandwidth utilization based on the valid codeword requests received.

Codeword generation algorithm must consider the computational overhead required to handle peer churning. Generating outgoing codewords every time a peer leaves/joins will requires intensive computation. However, the following lemma shows that the number of admissible codewords are significantly less than the default $(2^L)^F$, for L layers and F child peers.

Definition 29. *For an outgoing code having L probable layers to allocate, there are 2^L*

$(C_x^1\, C_x^2$	$C_y^1\, C_y^2)$	$(C_x^1\, C_x^2$	$C_y^1\, C_y^2)$
(0 0	0 0)	(~~1 0~~	~~0 0~~)
(0 0	0 1)	(~~1 0~~	~~0 1~~)
(0 0	1 0)	(1 0	1 0)
(0 0	1 1)	(1 0	1 1)
(~~0 1~~	~~0 0~~)	(~~1 1~~	~~0 0~~)
(0 1	0 1)	(~~1 1~~	~~0 1~~)
(0 1	1 0)	(~~1 1~~	~~1 0~~)
(0 1	1 1)	(1 1	1 1)

(a) Outgoing constraint in $\mathcal{C}_m^o$

Figure 38: Optimized codewords generation for outgoing code C_m^o for peer m

codewords:

$$\sum_{l=1}^{2^L} 1 \tag{80}$$

Since the codewords used to calculate the outgoing probability along each outgoing edge is the same, a peer only needs to calculate codewords with respect to one peer. Therefore, the number of codewords for n number of peers are:

$$\sum_{l_1=1}^{2^L} \sum_{l_2=l_1}^{2^L} \cdots \sum_{l_F=l_{F-1}}^{2^L} 1$$

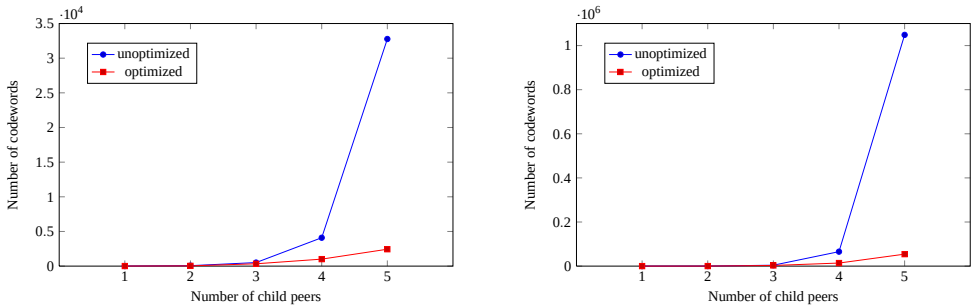
The optimized codewords generation associated with the outgoing code are given in Algorithm 11.

Example 17. *Let peer m be a parent with* 2 *child x and y. Assume that peer m has* 2 *layers. Codewords used to generate outgoing probability to peer x is the same as peer y. Fig. 38 shows the optimized process used to generate admissible codewords. Due to optimized codeword generation, the codeword combinations* 00 *and* 01 *are interchangeable. Therefore, codeword* 0100 *has not been used because codeword* 0001 *has already been generated. Similarly,* 1000, 1001 1100, 1101, *and* 1110 *has not been used because the reverse combinations* 0010, 0110, 0011, 0111 *and* 1011 *has already been generated.*

Example 18. *Fig. 39 shows a comparison for the number of codewords vs. number peers*

Algorithm 11 Optimized Codeword Generation for Outgoing Code: Multi-Layer

1: **Initialize code:** $\mathcal{C}^o = \emptyset$
2: **Initialize matrix:** B
3: **Input:** Number of Output Flows = F_o
4: **Initialize:** count[F_o] = 0
5: $\zeta = 0$
6:
7: **while** (**true**) **do**
8: {get the current codeword}
9: $c = \emptyset$
10: $\zeta = 0$
11: **for** $l = 0$ **to** L - 1 **do**
12: **for** $f = 0$ **to** F_o - 1 **do**
13: m = count[f]
14: ζ += b_{ml}
15: $C_f^l = b_{ml}$, where $C_f^l \in C_f^l$
16: C_l += b_{ml}, where $C_l \in C_l$
17: **end for**
18: **if** $C_l == 1$ **then**
19: $C_{l-1} = 1$, for $l > 1$
20: **end if**
21: $c \leftarrow c + C_l + C_f^l$
22: **end for**
23: **if** $\zeta == \sigma$ **then**
24: {This is a valid codeword}
25: $\mathcal{C}^o \leftarrow \mathcal{C}^o + c$
26: **end if**
27: {Search for the next optimized codeword}
28: **for** $f = F_o$ - 1 **to** 0 **do**
29: **if** count[f] < 2^L - 1 **then**
30: count[f] ++
31: **for** $j = f + 1$ **to** F_o - 1 **do**
32: count[j] = count[f]
33: **end for**
34: **break**
35: **end if**
36: **end for**
37: **if** count[0] $\geq 2^L$ **then**
38: **break**
39: **end if**
40: **end while**



(a) With 3 layers (b) With 4 layers

Figure 39: Number of codewords associated with the outgoing code vs. the number of child peer and the number of layers

for 3 and 4 layers. For a scalable stream with 3 *layers and* 5 *child peers, the unoptimized codewords generation process considers* 32768 *number of codewords to generate admissible codewords, while the optimized process considers only* 2436 *codewords to generate the list of admissible codewords (Fig. 39a). Similarly, with* 4 *layers and* 5 *child peers, the number codewords considered during the unoptimized code generation process is* 1048576*, while the number of codewords considered in the optimized code generation process is* 54264 *(Fig. 39b).*

Complexity Optimization for Incoming Code

Similar to the complexity associated with the generation of admissible codewords for outgoing code, optimized codewords generation is also necessary for incoming code. Let B be a $M \times L$ matrix where $M = 2^L$, containing all possible binary combination of layers that a child peer can receive from its parent. If there are F_i number of parents, the possible codeword combinations are $(2^L)^{F_i}$. Therefore, codeword optimization is necessary to reduce the computational complexity. Algorithm 12 presents the permutation based optimized codeword generation for incoming codes.

The algorithm takes a list of parents $\boldsymbol{p}$ and a binary array $\boldsymbol{r}[M]$ that corresponds to all the codeword combinations for L number of layers, where $M = 2^L$. The function is

Algorithm 12 Optimized Codeword Generation for Incoming Code

```
1: Initialize Matrix: A = M × L, M = 2^L
2: Input: F_i                {number of incoming flows}
3: Input: ℓ
4: Input: int p[F_i] {list is parents}
5: Input: int r[M]
6: Input: int list[F_i]
7: Input: int j
8: for i = j + 1 to M - 1 do
9:    r[i] = 1
10:   res = valid_codewords(r) {check duplicity and continuity constrain }
11:   if res == 1 then
12:      {This is valid codeword vector}
13:      sort_codeword_vector(list, r, |F_i|)
14:      repeat
15:         if valid_codeword_capacity(list, |F_i|, p[F_i]) == 1 then
16:            compute_probability_vector()
17:         end if
18:      until permute(list, |F_i|)
19:   else if res == -1 then
20:      r[i] = 0
21:   end if
22:   if ℓ < |F_i| − 1 then
23:      recursive_function_call(ℓ = ℓ + 1, j = i, r)
24:   end if
25:   r[i] = 0
26: end for
```

initially invoked with parameters $\ell = 0$ and $j = 0$. The valid_codewords function call in line 10 refers to Algorithm 8 that checks the continuity and duplicity constraints associated with incoming code. For a given set of codewords, the sort_codeword$_{\text{v}}$ector sorts the index of the vectors used from matrix $\boldsymbol{A}$ in ascending order and puts it in ***list*** array. If the number of codewords used (i.e., $\boldsymbol{r}[i] = 1$) is less than the number of parents, the sort function fills the remaining positions with 0s. The computation of the probability vectors are given in the next section.

Example 19. *If* $L = 3$ *and there are a total of* 4 *parents, the number of possible codeword vectors is* 7*. A combination of* 001 *(i.e., index* $\boldsymbol{r}[i] = 1$*) and* 010 *(i.e., index* $\boldsymbol{r}[i] = 2$*)*

Algorithm 13 Compute Next Permutation of an Ordered List

```
1:
2: Initialize Input: list
3: Initialize Input: len
4:
5: key = len − 1
6: newkey = len − 1
7: while key > 0 and list[key] ≤ list[key − 1] do
8:    key - -
9: end while
10: key - -
11: {If key ¡ 0 the data is in reverse sorted order, which is the last permutation}
12: if key < 0 then
13:    return 0
14: end if
15: newkey = len − 1
16: while newkey > key and list[newkey] ≤ list[key] do
17:    newkey - -
18: end while
19: swap(list, key, newkey)
20: len - -
21: key++
22: {The tail must end in sorted order to produce the next permutation.}
23: while len > key do
24:    swap(list, len, key)
25:    key++
26:    len - -
27: end while
28: return 1
```

constitutes a valid codeword combination. However, there are 4 parents. Therefore, ***sort*** *function returns the sorted indexes* $[0, 0, 1, 2]$ *in the* ***list*** *array.*

The permutation of all the vectors in the ordered ***list*** is generated by Algorithm 13 [76]. The permutation here refers the possible number of ways a child peer can receive a a binary row from its parents.For L number of layers, the computational complexity of finding the next permutation $\boldsymbol{O}(L)$. Therefore, if there are a total N permutations, the the final computational complexity for finding all the codewords is $\boldsymbol{O}(NL)$.

Example 20. *Fig. 40 presents an example of the application of Algorithm 12 to gener-*

Parent 1 cap: 1	Parent 2 cap: 3	Parent 3 cap: 2	Parent 4 cap: 1
000	000	000	000
001	001	001	001
010	010	010	010
100	011	011	100
	100	100	
	101	101	
	110	110	
	111		

(a) Parent Capacity

Level 1	Level 2	Level 3	Number of Codewords
001			4 P 1 = 4
001	010		4 P 2 = 12
001	010	100	4 P 3 = 24
001	110		$2 \cdot 3$ P 1 = 6
010			
010	101		$2 \cdot 3$ P 1 = 6
011			$2 \cdot 3$ P 0 = 2
011	100		$2 \cdot 3$ P 1 = 6
111			$1 \cdot 3$ P 0 = 1

(b) Codeword Construction

Figure 40: Possible binary combinations based on capacity of parents to construct the codewords

ate optimized codewords for incoming code. In this example, a child peer has 4 *parents. The capacity of the parents and the possible codeword representation of layers that the child can expect to receive from its parents are given in Fig. 40a. The optimized codeword generation process for this configuration is shown in Fig. 40b. The algorithm recursively generates possible codeword permutations. The algorithm starts with the codeword* 001 *. The codeword combinations* 001 000 000 000 *can be arranged* ${}^4P_1$ *possible ways to receive from 4 parent. Therefore, there are* ${}^4P_1 = 4$ *ways to receive layer 1. After successful recursive call, the possible codeword combinations* 001 010 000 000 *can be arranged in* ${}^4P_2 = 12$ *ways among 4 parents to receive 2 layers. After the next successful recursive call, the codeword combinations* 001 010 100 000 *can be*

arranged in ${}^4P_3 = 24$ *ways. At this point, the cursive call is at level 3 and no other codeword combination is valid. Therefore, the algorithm returns to level 2 in recursion. The next valid codeword combinations are* $0\,0\,1 \quad 1\,1\,0 \quad 0\,0\,0 \quad 0\,0\,0$. *From this codeword combination, a child can receive up to 3 layers in* ${}^4P_2 = 12$ *ways among* 4 *parents. However, only* 2 *parent can deliver the codeword* $1\,1\,0$. *Therefore, only* $2 \cdot {}^3P_1 = 6$ *possible ways a child can receive* 3 *layers in this codeword combination. Here the permutation* ${}^3P_1$ *is used because one level with codeword* $1\,1\,0$ *has been removed. Once* $1\,1\,0$ *is set, there is only 3 more places left for the permutation. The multiplication by* 2 *is used because there are* 2 *peers whose codewords can be set to* $1\,1\,0$. *At this point, the algorithm returns to level 1 in recursion. Since the codeword* $0\,1\,0$ *does not violate the duplicity constraint, a recursive call to level* 2 *ensures that this combined with codeword* $1\,0\,1$ *forms a valid combination. This codewords can be arranged in* $2 \cdot {}^3P_1 = 6$ *ways. The next valid codeword is* $0\,1\,1$. *Due to capacity constraint, only 2 peers can serve this codeword. Therefore, there are* $2 \cdot {}^3P_0 = 2$ *ways the codeword combinations* $0\,1\,1 \quad 0\,0\,0 \quad 0\,0\,0 \quad 0\,0\,0$ *can be arranged to receive* 2 *layers from parents. Finally, only 1 peer can server codeword* $1\,1\,1$. *Hence it can be arranged in only 1 way.*

Algorithm

The algorithm works on the codeword vector. The algorithm follows the same iterative method mentioned in Algorithm 5 and Algorithm 6 for outgoing and incoming codes respectively. For a given permutation of vectors probability is computed in Algorithms 14 and 15. Algorithm 14 computes the probability on the internal edge for the incoming code when messages go from parent to child and on the outgoing code when messages traverses from child to parent. Algorithm 15 computes the probability on the edges. For incoming code, this determines the probability when messages travel from child to parent. Similarly, Algorithm 15 also computes the probability on the edges connected to the outgoing code when parent send message to child.

Algorithm 14 Compute Probability: Edge to Layer Probability

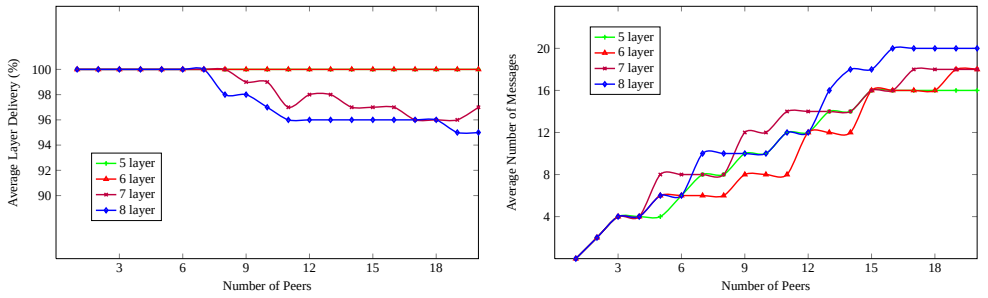
1: **Initialize Matrix:** $\boldsymbol{B} = M \times L,\ M = 2^L$
2: **Initialize:** prob = 1
3:
4: **Input:** F, type : int {edge set}
5: **Input:** vector $P_{\boldsymbol{f}}$, type : int
6: **Input:** vector $C_{\boldsymbol{l}}$, type : int
7: **Input:** edge[F][L][2], type : double
8: **Input:** pIn[L][2], type : double
9:
10: **Output:** pOut[L][2], type : double
11:
12: **for** $f = 0$ **to** F **do**
13: $m = P_f$
14: **for** $l = 0$ **to** L **do**
15: prob *= edge[f][l][B_{ml}]
16: **end for**
17: **end for**
18: **for** $l = 0$ **to** L **do**
19: prob *= pIn[l][C_l]
20: **end for**
21: **for** $l = 0$ **to** L **do**
22: pOut[l][C_l] = prob/pIn[l][C_l]
23: **end for**

Algorithm 15 Compute Probability: Layer to Edge Probability

1: **Initialize Matrix:** $\boldsymbol{B} = M \times L,\ M = 2^L$
2: **Initialize:** prob = 1
3:
4: **Input:** F, type : int {edge set}
5: **Input:** vector $P_{\boldsymbol{f}}$, type : int
6: **Input:** vector $C_{\boldsymbol{l}}$, type : int
7: **Input:** edgeIn[F][L][2], type : double
8: **Input:** pIn[L][2], type : double
9:
10: **Output:** edgeOut[F][L][2], type : double
11:
12: **for** $f = 0$ **to** F **do**
13: $m = P_f$
14: **for** $l = 0$ **to** L **do**
15: prob *= edgeIn$[f][l][B_{ml}]$
16: **end for**
17: **end for**
18: **for** $l = 0$ **to** L **do**
19: prob *= pIn$[l][C_l]$
20: **end for**
21: **for** $f = 0$ **to** F **do**
22: $m = P_f$
23: **for** $l = 0$ **to** L **do**
24: edgeOut$[f][l][B_{ml}]$ = prob/edgeIn$[f][l][B_{ml}]$
25: **end for**
26: **end for**

G Simulation

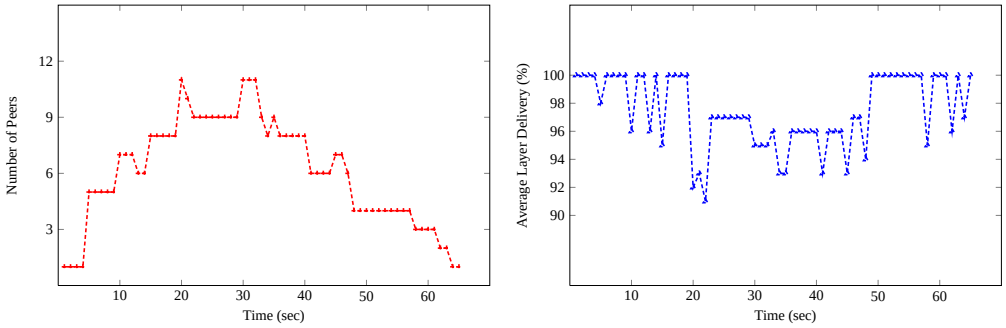
Preliminary simulation is performed with 20 peers in the network. The server peer is assigned a 3 Mbps bandwidth. A group of 25% of the peers, selected randomly, are assigned a bandwidth of 500 Kbps, 1 Mbps, 1.5 Mbps. Rest of the peers are assigned 2 Mbps bandwidth. Furthermore, the maximum number of incoming or outgoing connection is set to 3. We assume that each layer requires 150 Kbps bandwidth to deliver and there are up to a total of 8 layers available. In this simulation, each peer seeking a parent randomly selects a peer to be parent. However, this algorithm can be used together with any other parent-child



(a) Average layer delivery ratio as peers join the network

(b) Number of messages exchanged between parent and child peers to reach optimality

Figure 41: Average layer deliver ratio and message complexity as peers join network



(a) Number of peers with peer churning

(b) Layer delivery ratio for all peers in the network

Figure 42: Average layer delivery ratio of peers in the network in the presence of peer churning

selection methodology for the overlay construction. Fig. 41a shows the average layer delivery ratio for various layers as peers join the network. When there are up to 5 and 6 layers available, the algorithm achieves maximum layer allocation. As the number of available layers increase, average delivery ratio decreases. Up to layer 8, the algorithm achieves 95% average layer delivery. Fig. 41b shows the average number of messages exchanged between any two peers before the algorithm converges. Since the algorithm proceeds in rounds and convergence decision is reached after a cycle of parent-child-parent messages is completed, total number of messages is always a multiple of 2.

Fig. 42 shows the average delivery ratio in the presence of peer churning. Fig. 42a shows the total number of peers in the network. After every 5 seconds, between 1 to 4

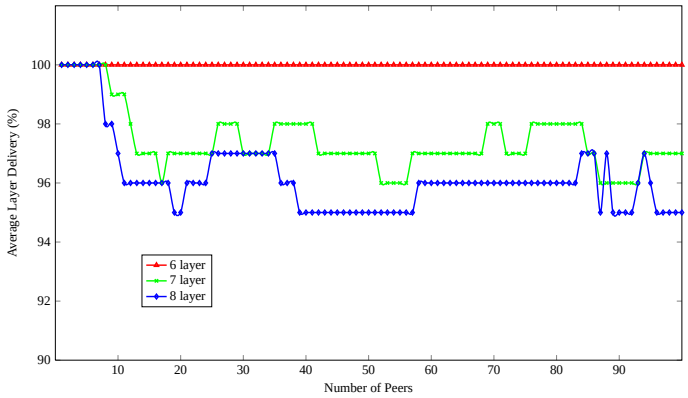


Figure 43: Average layer delivery ratio as peers join the network

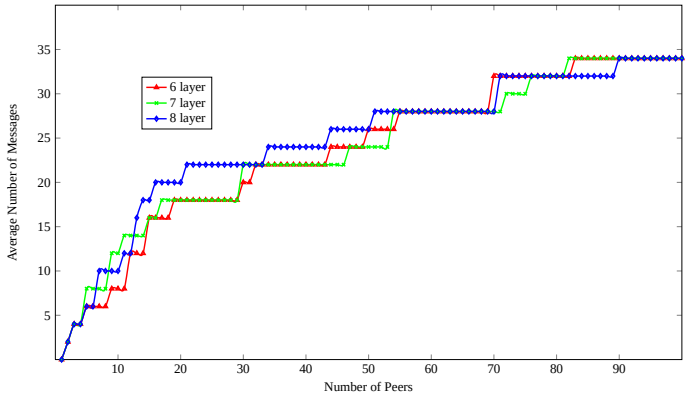


Figure 44: Number of messages exchanged between parent and child peers to reach convergence

peers join the network. They are randomly given a life-time between 5 to 30 seconds. The left axis show the average layer delivery when up to 8 layers are available. Simulation has also been performed with larger networks. Fig. 43 shows the average layer delivery ratio as large number of peers join the network. In addition, Fig. 44 shows the number of messages required for layer allocations to converge as peers join the network. Comparing Fig. 41b and Fig. 44, it is clear that as more peers join the network, the rate of the number of messages required slowly decreases.

CHAPTER VII

CONCLUSION

An optimal rate allocation solution for P2P applications is presented. For continuous-rate video streams, non-linear convex optimization framework has been used to minimize the aggregated distortion and thus to maximize the overall PSNR among all peers in a P2P network. The optimization process uses peer relaying price - unique in a P2P distribution scenario - along with the network price. Simulation shows that using this double pricing solution improves the aggregate rate distortion for all peers in the network and provides a better video experience compared to a solution that uses relay and network prices separately. A solution has also been developed for multi-path P2P networks.

For scalable video streams, a heuristic-based layer allocation algorithm for a P2P mesh network has been developed. The algorithm targets to achieve close to optimal rate among peers by considering load balancing and weight based layer allocation. This ensures that a child evenly distributes layer allocation request among all its parents and a parent allocates higher rates to a child that in turn has more children than other child. Simulation shows that for up to 7 layers, the algorithm achieves a layer delivery ratio of 90% more.

Finally, a simple sum-product based message-passing approach has been developed to solve the problem of scalable video optimization in the context of P2P mesh network. The simple but elegant nature of the algorithm results from the fact that the network and video properties are embedded in a set of codewords. Sum-product algorithm iteratively updates the probability along each connecting edge based on these set of codewords. Results show that peers achieve 95% or higher average layer delivery ratio with exchanging fewer than 20 messages between neighbors.

REFERENCES

[1] J. Liu, S. Rao, B. Li, and H. Zhang, "Opportunities and challenges of p2p internet broadcast," *Proc. IEEE*, 2008.

[2] V. Jacobson, "Congestion avoidance and control," *SIGCOMM*, vol. 18, 1988.

[3] D. Tan and A. Zakhor, "Real-time internet video using error resilient scalable compression and tcp-friendly transport protocol," *IEEE Transactions on Multimedia*, 1999.

[4] J. Mahdavi and S. Floyd, "Tcp-friendly unicast rate-based flow control," 1997, note sent to end2end-interest mailing list.

[5] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control," *IEEE Transactions on Networking*, 1999.

[6] "Osi network layer model." [Online]. Available: http://en.wikipedia.org/wiki/OSI_model

[7] Y. Chu, R. Rao, and H. Zhang, "A case for end system multicast," *In ACM SIGMETRICS*, 2000.

[8] N. Magharei, R. Rejaie, and Y. Guo, "Mesh or multiple-tree: A comparative study of live p2p streaming approaches," *IEE Infocom*, pp. 1424–1432, 2007.

[9] M. Chen, M. Ponec, S. Sengupta, J. Li, and P. A. Chou, "Utility maximization in peer-to-peer systems," in *SIGMETRICS '08: Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. ACM, 2008, pp. 169–180.

[10] T. Nguyen, , and A. Zakhor, "Distributed video streaming over internet," *MMCN*, 2002.

[11] ——, “Multiple sender distributed video streaming,” *IEEE Transactions on Multimedia*, 2004.

[12] A. Majumdad, R. Puri, and K. Ramchandran, “Distributed multimedia transmission from multiple servers,” *IEEE ICIP*, 2002.

[13] M. Hafeeda, A. Habib, D. Xu, B. Bhargava, and B. Botev, “Promise: peer-to-peer media straming collectcast,” *ACM IMC*, 2003.

[14] V. Padmanabhan, H. Wang, and P. Chou, “Supporting heterogeneity and congestion control in peer-to-peer multicast streaming,” *IPTPS*, 2004.

[15] V. Agarwal and R. Rajaie, “Adaptive multi-source streaming in heterogeneous peer-to-peer networks,” *MMCN*, 2005.

[16] S. McCanne, V. Jacobson, and M. Vetterli, “Receiver-driven layered multicast,” *ACM SIGCOMM*, vol. 26, pp. 117–130, 1996.

[17] J. Liu, B. Li, and Y. Zhang, “Adaptive video multicast over the internet,” *In IEEE Multimedia*, vol. 10, 2003.

[18] K. Stuhlmuller, N. Farber, M. Link, and B. Girod, “Analysis of video transmission over lossy channels,” *Selected Areas in Communications, IEEE Journal on*, vol. 18, no. 6, pp. 1012–1032, 2000.

[19] Z. Zhu, E. Setton, and B. Girod, “Congestion-distortion optimized video transmission over ad hoc networks,” *EURASIP Journal of Signal Processing: Image Communications*, 2005.

[20] D. Katabi, M. Handley, and C. Rohrs, “Internet congestion control for high bandwidth-delay product environment,” *ACM Siggcomm*, 2002.

[21] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *ACM SIGCOMM conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2000.

[22] R. Rejaie, M. Handley, and D. Estrin, "Rap: An end-to-end rate-based congestion control mechanism for realtime streams in the internet," in *IEEE Infocom*, 1999.

[23] D. Sisalem and H. Schulzrinne, "The loss-delay based adjustment algorithm: A tcp-friendly adapation scheme," in *International workshop on Network and operating systems support for digital audio and video (NOSSDAV)*, 1998.

[24] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "Rtp: A transport protocol for real-time applications," *RFC 1889*, 1996.

[25] I. Kim, Y. Kim, M. Kang, J. Mo, and D. Kwak, "Tcp-mr: Achieving end-to-end rate guarantee for real-time multimedia," in *2nd International Conference on Communications and Electronics (ICCE)*, 2008.

[26] J. Wagner and P. Frossard, "Media-friendly distributed rate allocation," *IEEE Transactions on Multimedia*, 2008.

[27] D. Bertsekas, *Nonlinear Programming*. Athena Scientific, 1999.

[28] F. Kelly, "Charging and rate control for elastic traffic," *European Transactions on Telecommunications*, vol. 8, no. 1, 1997.

[29] F. Kelly, A. Maullo, and D. Tan, "Rate control for communication networks: Shadow prices, proportional fairness and stability," *Journal of Operations Research Society*, vol. 49, no. 3, 1998.

[30] S. Low and D. Lapsley, "Optimization flow control, i: Basic algorithm and convergence," *IEEE/ACM Transactions on Networking*, vol. 7, no. 6, pp. 861–874, 1999.

[31] K. Kar, S. Sarkar, and L. Tassiulas, "Optimization based rate control for multirate multicast sessions," *IEEE Infocom*, 2001.

[32] D. Bertsekas and J. Tsittsiklis, *Parallel and Distributed Computation*. Prentice-Hall, 1989.

[33] H. Han, S. Shakkottai, C. Hollot, R. Srikant, and D. Towsley, "Multi-path tcp: A joint congestion control and routing scheme to exploit path diversity in the internet," *IEEE/ACM Trans. Networking*, 2006.

[34] D. Jurca and P. Frossard, "Media flow rate allocation in multipath networks," *IEEE Transactions on Multimedia*, vol. 9, no. 6, 2007.

[35] X. Zhu, T. Schierl, T. Wiegand, and B. Girod, "Video multicast over wireless mesh networks with scalable video coding(svc)," *Visual Communications and Image Processing*, January 2008.

[36] M. Chiang, S. Low, A. Calderbank, and J. Doyle, "Layering as optimization decomposition: a mathematical theory of network architectures," *Proc. of IEEE*, 2007.

[37] Y. Cui, Y. Xue, and K. Nahrstedt, "Optimal resource allocation in overlay multicast," *IEEE International Conference on Network Protocols*, 2003.

[38] M. Castro, P. Druschel, A.-M. Kermarrec, A. R. A. Nandi, and A. Singh, "Splitstream: High-bandwidth content distribution in a cooperative environment," in *ACM SOSP*, 2003.

[39] M. Dai and D. Loguinov, "Analysis of rate-distortion functions and congestion control in scalable internet video streaming," in *ACM NOSSDAV*, 2003.

[40] C. Hsu and M. Hefeeda, "On the accuracy and complexity of rate-distortion models for fgs-encoded video sequences," *ACM Transactions on Multimedia Computing, Communications, and Applications*, vol. 4, no. 2, 2008.

[41] X. Zhu, P. Agarwal, J. Singh, T. Alpcan, and B. Girod, "Rate allocation for multi-user video streaming over heterogeneous access networks," *ACM Multimedia*, 2007.

[42] K. Eger and U. Killat, "Fair resource allocation in peer-to-peer networks," *Elsevier Computer Communications*, 2007.

[43] X. Xiao, S. Yuanchun, Y. Gao, and Q. Zhang, "Layerp2p: A new data scheduling approach for layered streaming in heterogeneous networks," *IEEE Infocom*, 2009.

[44] Y. Cui and K. Nahrstedt, "Layered peer-to-peer streaming," *In Proc. NOSSDAV*, 2003.

[45] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: high bandwidth data dissemination using an overlay mesh," *ACM SOSP*, 2003.

[46] D. Tran, K. Hua, and T. Do, "Zigzag: An efficient peer-to-peer scheme for media streaming," *IEEE Infocom*, 2002.

[47] S. Banerjee and B. Bhattacharjee, "Scalable application layer multicast," *ACM SIGCOMM*, 2002.

[48] R. Rajaie and S. Stafford, "A framework for architecting peer-to-peer receiver driven overlays," *ACM NOSSDAV*, 2004.

[49] S. Banerjee, S. Lee, and B. Bhattacharjee, "Resilient multicast using overlays," *ACM SIGMERICS*, 2004.

[50] V. Venkatraman, K. Yoshida, and P. Francis, "Chunkyspread: heteroeneous unstructured end system multicast," *IEEE ICNP*, 2006.

[51] A. Ganesh and A. e. a. Kermarrec, "Peer-to-peer membership management for gossip-based protocols," *IEEE Transactions on Computers*, 2003.

[52] X. Xiao, Y. Shi, B. Zhang, and Y. Gao, "Ocals: A novel overlay construction approach for layered streaming," *In Proc. IEEE ICC*, 2008.

[53] H. Guo, K. Lo, Y. Qian, and J. Li, “Peer-to-peer live video distribution under heterogeneous bandwidth constraints,” *IEEE Trans. on Prallel and Distributed Systems*, 2009.

[54] J. Lee, R. Mazumdar, and N. Shroff, “Non-convex optimization and rate control for multi-class services in the internet,” *IEEE Trans. on Networking*, 2005.

[55] P. Hande, S. Zhang, and M. Chiang, “Distributed rate allocation for inelastic flows,” *IEEE Transactions on Networking*, 2007.

[56] R. Rajaie and A. Ortega, “Pals: Peer-to-peer adaptive layered streaming,” *NOSSDAV*, 2003.

[57] Z. Liu, Y. Shen, K. Ross, S. Panwar, and Y. Wang, “Layerp2p: Using layered video chunks in p2p live streaming,” *IEEE Transactions on Multimedia*, 2009.

[58] G. Gallager, “Low-density parity-check codes,” *IRE Transactions on Information Theory*, 1962.

[59] R. Tanner, “A recursive approach to lowcomplexity codes,” *IEEE Transactions on Information Theory*, 1981.

[60] N. Wiberg, H. Loeliger, and R. Køtter, “Codes and iterative decoding on general graphs,” *Euro Trans. Telecomm*, 1995.

[61] R. Ahlswede, N. Cai, and R. Li, S. Yeung, “Network information flow,” *IEEE Transactions on Information Theory*, 2000.

[62] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near shannon limit error-correcting coding and decoding,” *Proc. International Communications Conference*, 1993.

[63] E. Aurell, U. Gordon, and S. Kirkpatrick, “Comparing beliefs, surveys, and random walks,” *Advances in Neural Information Processing Systems*, 2005.

[64] C. Moallemi, “A message passing paradigm for optimization,” *PhD Dissertation*, 2007.

[65] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, “Tcp throughput: A simple model and its empirical validation,” in *ACM SIGCOMM Symposium on Communications Architectures and Protocols*, 1998.

[66] A. Renyi and L. Vekerdi, *Probability Theory*. North-Holland, 1970.

[67] D. Katabi and C. Blake, “Inferring congestion sharing and path characteristics from packet interarrival time,” 2001, mIT TR-828.

[68] T. Hossain, Y. Cui, and Y. Xue, “Minimizing rate distortion in peer-to-peer video streaming,” *IEEE CCNC*, 2009.

[69] ——, “Rate distortion optimization for mesh-based p2p video streaming,” *IEEE ICC*, 2009.

[70] W. Wang, M. Palaniswami, and S. Low, “Optimal flow control and routing in multi-path networks,” *ELSEVIER Performance Evaluation*, vol. 52, pp. 119–132, 2003.

[71] “Videolan.” [Online]. Available: http://www.videolan.org

[72] “Xiph.org.” [Online]. Available: http://media.xiph.org/video/derf/

[73] C. Huang, J. Li, and K. Ross, “Can internet video-on-demand be profitable?” *Proc. SIGCOMM*, 2007.

[74] T. Hossain, Y. Cui, and Y. Xue, “On the optimality of layered video streaming rate in a p2p mesh network,” *IEEE ICCCN*, 2009.

[75] G. Forney, “Codes on graphs: normal realizations,” *IEEE Transactions on Information Theory*, 2001.

[76] J. Johnson, “Sepa: A simple, efficient permutation algorithm.” [Online]. Available: http://www.freewebs.com/permute/soda_submit.html