PEER-TO-PEER MULTIPOINT VIDEO CONFERENCING USING LAYERED VIDEO

by

İstemi Ekin Akkuş

A Thesis Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Electrical and Computer Engineering

Koç University

August, 2007

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

İstemi Ekin Akkuş

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

Dr. M. Reha Civanlar (Advisor)

Assist. Prof. Öznur Özkasap (Advisor)

Prof. A. Murat Tekalp

Assist. Prof. F. Sibel Salman

Assist. Prof. Serdar Taşıran

Date:

To my family and friends

ABSTRACT

A new peer-to-peer architecture for multipoint video conferencing using layered video coding with two layers at the end hosts is presented. The system targets end points with low bandwidth network connections (single video in and out) and enables them to create a multipoint conference without any additional networking and computing resources than what is needed for a point-to-point conference. In contrast to prior work, it allows each conference participant to see any other participant at any given time under all multipoint configurations of any number of users, with a caveat that some participants may have to receive only the base layer video. Layered video encoding techniques usable within this architecture are described. A protocol for the peer-to-peer video transmission approach has been developed and simulated. Its performance is analyzed. A prototype has been implemented and successfully deployed. The effects of upload bandwidths that can support multiple video signals are investigated.

Furthermore, a multi-objective optimization approach to minimize the number of base layer receivers, to minimize the delay experienced by the peers and to maximize the granted additional requests to support peers having multiple input bandwidths has been developed. A technique assigning importance weights to each of the objectives is proposed and explained. The use of the proposed multi-objective optimization scheme is demonstrated through example scenarios. The system has been fully specified and verified. To ensure secure transmission of video, a scheme providing authentication, integrity and non-repudiation is presented and its effectiveness is shown through simulations.

ÖZETÇE

Uç noktalarda katmanlı video işleme yöntemi ile iki katman olarak işlenen video görüntüleri kullanan yeni bir eşler arası çok noktalı video konferans mimarisi sunulmaktadır. Bu mimari, düşük bant genişliğine (bir video alış ve gönderiş) sahip kullanıcılara, bir noktadan bir noktaya konferans için gerekli ağ ve işlemci gücünden daha fazlasına gerek duymadan çok noktalı konferans yetisi vermektedir. Daha önceki çalışmadan farklı olarak, bazı katılımcıların taban kalitede video izlemesi karşılığında, her katılımcının, istediği katılımcıyı herhangi bir zaman ve düzenleşimde izleyebilmesine olanak sağlamaktadır. Kullanılabilecek katmanlı video işleme yaklaşımları betimlenmiş, eşler arası video gönderimi için bir protokol geliştirilmiş ve benzetimler yapılmıştır. Protokolun başarımı incelenmiştir. Prototip bir uygulama geliştirilmiş ve başarılı bir şekilde gerçek hayata geçirilmiştir. Birden fazla gönderimi destekleyen bant genişliklerinin etkileri incelenmiştir.

Ayrıca, taban katman video izleyenlerin sayısını ve eşlerin gözlemlediği gecikmeyi azaltmayı ve birden fazla alış bant genişliğine sahip kullanıcıları desteklemek üzere, karşılanan ek video taleplerinin sayısını arttırmayı hedefleyen bir çoklu eniyileme yaklaşımı geliştirilmiştir. Bu hedeflerin her birine önem katsayıları atayan bir yöntem sunulmuş ve açıklanmıştır. Sunulan çoklu eniyileme yaklaşımının, sistem içindeki kullanımı, örnek senaryolarla betimlenmiştir. Sistem özellikleri tanımlanmış ve doğrulanmıştır. Videonun güvenli bir şekilde iletimi için, kimlik doğrulama, bütünlük kontrolü ve inkar önleme destekleyen bir çözüm sunulmuş ve etkinliği benzetimlerle gösterilmiştir.

ACKNOWLEDGMENTS

I would like to thank to my advisors Dr. Reha Civanlar and Prof. Öznur Özkasap for their guidance and time. I would also like to thank to Prof. Serdar Taşıran for supplying me another perspective on my work. I would also like to thank to my other committee members Prof. Sibel Salman and Prof. Murat Tekalp for their corrections and suggestions to improve my work.

Without my friends and colleagues, I would not have come this far: Canan Uçkun, Engin Kurutepe, Emre Atsan, Erkan Keremoğlu, Burak Görkemli, Emre Güney, Göktuğ Gürler, ”Metmet” Ali Yatbaz, Tayfun Elmas, Merve Kovan, Çiçek Güven, Berkin Abanoz, Engin Ural, Ferda Ofli, Erhan Baş, Ekin Tüzün, Nurcan Tunçbağ, Gözde Kar, Selen Pehlivan, Mustafa Kaymakçı, Onur Demir, Hazal Özden, Soner Yaldız, and many others at the Graduate School of Sciences and Engineering at Koç University.

Finally, I would like to thank my family, whose only intention is the *good of mine*, and who never rolled back (and I know, *would never will*) their spiritual and financial support for me, no matter what I do.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

ESM	End System Multicast
GVC	Global Video Conference
IM	Instant Messaging
ISP	Internet Service Provider
JSVM	Joint Scalable Video Model
LC	Layered Coding
LTL	Linear Temporal Logic
LVC	Local Video Conference
MCU	Multipoint Control Unit
MDC	Multiple Description Coding
MP	Multipoint
OSI	Open Systems Interconnection
P2P	Peer-to-Peer
PROMELA	Process Meta Language
RP	Rendezvous Point
RTT	Round Trip Time
TLA	Temporal Logic of Actions
TLC	Temporal Logic Checker
QoS	Quality of Service
VoD	Video on Demand
VoIP	Voice over IP

Chapter 1

INTRODUCTION

The Internet has revolutionized people's communication methods. It started replacing traditional pen-paper model letter with e-mail and continued with voice over IP (VoIP). With the increasing computing power and its decreasing cost, image and video coding has become more common. This created a place for video communications beneath the text and audio. Unfortunately, the increase in the access bandwidth to the Internet is not as steep as it is in the computing power of end hosts. Also, the cost does not become cheaper as speedily.

Instant messaging (IM) applications (e.g., ICQ^{TM}, Microsoft Messenger^{TM}) were originally designed for real-time text communications. VoIP applications (e.g., Skype^{TM}, VoipBuster^{TM}) targeted audio communications. Although these started providing point-to-point (i.e., between two end points) text and audio communications, they incorporated multipoint conferencing abilities as well and provided multi-user chat rooms and audio conferences where more than two end points could interact with each other. Video conferencing is slowly merging into people's lives as these applications try to include video communications as well. Unfortunately, the bandwidth demand of video communications prevents these applications to form a multipoint (MP) video conferencing environment and limits them to only point-to-point video communications. This is mostly because of the fact that low bandwidth connections (e.g., ordinary modem over a phone line or wireless GPRS) that are barely enough for point-to-point video communications make more than one video connection infeasible. Moreover, users tend to consume as much of the available bandwidth as possible to increase their video quality and, hence, a MP video system that increases the demand for bandwidth cannot be popular.

The bandwidth demand of a MP video system can be reduced using a special network-based equipment called Multipoint Control Unit (MCU) [1]. The MCU acts as a single-point recipient for each participant, thus needing a large bandwidth connection itself. It prepares a multipoint video representation that can fit into a smaller bandwidth and sends it to each participant. However, because of the complexity and cost of the operations of the MCUs, they are mostly used by large business applications that can afford such equipment. They also suffer from single-point-of-failures and hence are not failure transparent.

Multicasting is another approach to reduce bandwidth demands of MP video conferencing whenever the underlying network supports it. The additional advantage of a multicasting-based solution is the reduced operational complexity [2], but it requires the support of routers. Deploying multicast-supporting routers on the global Internet by the Internet Service Providers (ISPs) is costly and is not popular among the system administrators since they may increase security risks. Therefore, MP video conferencing using this approach is not applicable, due to the fact that native mode multicasting on the global Internet has not been realized.

Initiated for file sharing aims, peer-to-peer (P2P) systems became extremely popular in a short time. The distributed architecture where each peer could act as a server and a client at the same time, created a difference from the traditional client/server models. This has triggered P2P systems to find diverse applications. One of these applications is media streaming. Most of the recent techniques for P2P media streaming on the Internet utilize multicast model at the application layer. The main benefit of implementing application layer multicast is overcoming the lack of large-scale IP multicast deployment at the network layer.

In this thesis, we propose and develop extensions to the alternative approach to MP video conferencing presented in [3]. The system is based on the use of a distributed P2P architecture which does not need any special hardware or network infrastructure support. Its P2P architecture prevents single-point-of-failures and provides failure transparency (i.e., if one peer crashes, the system continues to function). There is no additional networking and computing resources needed at the end points more than that of a point-to-point video conference. In this system, each participant can see *one* other participant under *most* practical cases. However, the number of denied video requests increases as the participant count increases [3].

One of the extensions proposed in this thesis is the use of layered video. Utilizing two layers of video (i.e., base and enhancement layers), we overcome the problem of denying video requests and assure that each participant can view any other participant at any configuration. Although this may cause that some participants receive base layer quality video, we show that even with small bandwidths, the video is in acceptable quality. We also propose some optimizations to minimize the number of base layer receiving participants, since the existence of participants receiving base layer only becomes inevitable at some cases.

Simulation results show that the optimizations make the system scalable in terms of participant count. The use of layered video does not cause the system to crash as the participant count increases, nor does it hurt the average video quality viewed at the conference. The approach assumes that each peer has at least the computing and networking resources that are enough for a point-to-point video conference. That means, peers can send one video signal and receive one video signal at the same time. By the use of layered video, we *virtually* break the upload bandwidth into two. This way we guarantee that each peer's video request is granted.

We also present a multi-objective optimization framework to handle delays between the peers and their heterogeneous bandwidth connections. Since our approach makes use of a chain-based solution [4], this may cause that the peers at the ends of chains receive video after several peers have forwarded it. Furthermore, peers can have different bandwidth connections that can support more than one video download and upload. We define objective functions to minimize the number of base layer receivers, to minimize the maximum delay experienced in a chain and to maximize the number of additional requests granted. We formulate the multi-objective functions and describe how they can be employed within the system.

Other aspects of the system are also investigated. The system is modeled in detail using Lamport's specification language TLA+ based on Temporal Logic of Actions (TLA) [5]. TLA is a logic to specify and reason about concurrent and reactive systems. The system model is then verified with TLA+ Model Checker (TLC) [6]. Besides, since peers may act as a relay to forward the video of another peer, any intermediate peer *can* alter the original video. In order to prevent this, a security scheme is proposed. This scheme works on packet level and provides integrity, authentication and non-repudiation.

1.1 Contributions

Contributions of this study are the following:

1. We propose a P2P MP video conferencing system that makes use of layered video. The participants (i.e., the peers) are assumed to have *at least* the networking resources that are enough to be used in a point-to-point video conference, that is, they are able to send one video signal and receive one video signal at the same time. The use of layered video with two layers (i.e., base and enhancement layers) makes sure that each participant can view another participant under any configuration.

2. A fully distributed algorithm and architecture of the P2P MP video conferencing system is developed and presented. Although all participants can receive one other participant's video signal of their choice, some peers only receive base layer quality video. Optimizations to minimize the number of such receivers are proposed and presented along with simulation results.

3. We present layered video techniques that can be employed within the proposed system. They are explained and compared.

4. A multi-objective optimization framework has been developed to consider the end-to-end delays between peers and their heterogeneous bandwidth connections to the Internet. Objective functions to minimize the number of base layer receivers, to minimize the maximum delay experienced and to maximize the number of additional requests granted are defined and derived. Formulations to achieve these objectives are developed and explained with example scenarios.

5. The formal specification of the system has been made using TLA+. The system is modeled in detail; describing the protocol run by the peers, the format of the messages they exchange, the syntax and semantics of these messages and the actions when these are received. The model has been verified with TLA+ Modelchecker (TLC). An abstraction map between the real-life application and the model is presented.

6. A simulation study to integrate security into the proposed system has been carried out. The proposed scheme integrates integrity, authentication and non-repudiation into the system via generating digests per packet and digitally signing them.

1.2 Organization

Next chapter gives a literature survey on video streaming applications and video conferencing tools. It also covers related work for the verification process and the security scheme. Chapter 3 explains the theory of our P2P approach using layered video. It also describes layered video schemes and presents optimizations to minimize the number of base layer quality video receivers in a configuration along with simulation results. Chapter 4 presents the details of our system prototype. We present the Multi-objective Optimization framework in Chapter 5. Other aspects of the proposed system such as formal specification of the system model, formal verification and security scheme are described in Chapter 6. Finally, Chapter 7 includes the concluding remarks and gives future directions.

Chapter 2

RELATED WORK

In this chapter, existing studies on P2P video streaming applications and video conferencing tools will be analyzed and investigated. Their targets, advantages and shortcomings will be discussed. Also, other related work on formally specifying and verifying distributed systems and securing a streamed media is presented.

2.1 Application Layer Multicast

Since replacing routers to support multicast is not an easy task to be realized in the global Internet, another method has been proposed. In the application layer multicast -also known as End System Multicast (ESM), P2P Multicast, Overlay Multicast- end hosts are used to relay data instead of routers in native multicast.

Narada [7] is a self-organizing and self-improving protocol for conferencing applications and adapts to network dynamics. After maintaining a connected graph called *mesh* among the peers, Narada constructs a shortest path spanning tree on the mesh whenever a source wants to transmit content to a set of receivers. The mesh is improved by adding links or removing them in an incremental fashion, considering *stress*, number of identical copies of a packet carried over a physical link, *relative delay penalty*, ratio of the delay between two members in the mesh to the unicast delay between them, and *resource usage*. Results indicate that ESM is a promising approach for conferencing applications in dynamic and heterogeneous Internet settings. The study shows the necessity for self-organizing protocols to adapt to latency and bandwidth metrics.

The implementation of video conferencing through ESM is reported in [8, 9]. In practice, ESM is widely and successfully used for single-source video streaming. The assumption is that in a typical conferencing application, there is a source transmitting at any point in time. Although its applicability to multi-source video conferencing is mentioned, its practical usage in this context is not that common and related performance results are not reported. Besides, the system assumes that participants have large upstream bandwidths. Also, maintaining the mesh becomes costly as the group size is increased, since the links are added by probing.

NICE [10] is another application layer multicast protocol which aims larger receiver sets than targeted in Narada [7]. NICE clusters the peers into a hierarchical structure. The hierarchy of clusters is useful for the scalability of the system, since NICE requires peers to maintain state of some other peers. With the clustering, the size of this state table is bounded with $O(logN)$. The hierarchical structure implicitly defines the multicast overlay paths, where each cluster head forwards data to other peers in the same cluster. The cluster heads form another cluster in the higher level and receive the data from their heads. Clustering also brings the advantage of faster recovery on leaves or failures of peers, since it is localized. The increase in the control overhead is logarithmic as the group size increases, which makes NICE scale better than Narada for large receiver sets. The performance may be increased by using randomized forwarding when there is high packet losses and host failures [11].

Zigzag [12] is another P2P system developed for single-source media streaming for large receiver groups like NICE. Zigzag and NICE look similar; however, they differ in their multicast tree construction and maintenance mechanisms. Although ZIGZAG also possesses a hierarchical cluster structure for peers, the cluster members are not used to forward the content to the peers. Instead, the so-called *associate* heads from the upper layer are used. This gives the ability of recovering fast from failures. Zigzag also aims that the height of the multicast tree is logarithmic and thus, to minimize the end-to-end delay from the server to the peers.

2.2 P2P Video Streaming

Nearly all P2P video streaming applications found in the literature make use of application layer multicast. Application layer multicast overcomes the lack of IP multicast by using the end hosts. However, there is a performance penalty since some packets may have to pass some links more than once. Generally, P2P video streaming applications aim to minimize this penalty and enhance performance of the application layer multicast by making use of video encoding techniques and using clever techniques in their construction of the overlay network.

One can classify P2P video streaming applications according to

- their target group size
- whether they are distributing video on demand (VoD) or live video
- the number of sources
- whether they use layers in their video encoding (i.e., single-stream, Multiple Description Coding (MDC) or Layered Coding (LC))
- their construction of data networks.

However, this classification does not prevent that these groups would intersect. Related work under this classification can be viewed at Tables 2.1, 2.2 and 2.3. In the context of P2P video streaming applications, we will focus on their video encoding approaches and the way they make use of video encoding structures. We will also take a closer look on how the data network is constructed when it comes to distributing live content.

Table 2.1: Classification of video streaming applications according to their target group size

Small	MP Video Conferencing Applications [30, 32, 3]
Medium	Narada [7, 8, 9]
Large	NICE [10, 11], Zigzag [12], P2Cast [14], SplitStream [16], CoopNet [20, 21], PeerCast & SpreadIt [26, 27], P2VoD [15], Centralized P2P applications [22, 23, 24], DONet & CoolStreaming [28], AnySee [29],

Table 2.2: Classification of video streaming applications according to their distributed content

Video on Demand	P2Cast [14], SplitStream [16], CoopNet [20], P2VoD [15], Centralized P2P applications [22, 23, 24]
Live Video	DONet & CoolStreaming [28], PeerCast & SpreadIt [26, 27], AnySee [29], CoopNet [21], MP video conferencing applications [30, 32, 3]

Table 2.3: Classification of video streaming applications according to the number of sources

Single source	Nice [10, 11], Zigzag [12], SplitStream [16], CoopNet [20, 21], P2Cast [14], P2VoD [15] PeerCast & SpreadIt [26, 27], AnySee [29], DONet [28]
Multiple sources	Narada [7, 8, 9], MP video conferencing applications [30, 32, 3]

Table 2.4: Classification of video streaming applications according to video encoding approaches

Single-stream	GnuStream [13], P2Cast [14], P2VoD [15], MP Video Conferencing Applications [30, 32, 3]
MDC	SplitStream [16], CoopNet [20, 21]
LC	Centralized P2P applications [22, 23, 24]

Table 2.5: Classification of video streaming applications according to their construction of data networks

Receiver-driven	GnuStream [13]
Source-driven	Narada [7, 8, 9], NICE [10, 11], Zigzag [12], PeerCast & SpreadIt [26, 27]
Data-driven	DONet & CoolStreaming [28], AnySee [29]

2.2.1 Video Encoding: Single-stream

GnuStream [13] is built on top of Gnutella and is a receiver-driven media streaming system. Instead of one fixed sender, a dynamic set of peers is used. This way, the streaming bandwidth is aggregated from multiple senders, so that the streaming load is distributed for the senders, and the receivers can react to the departure of the clients in a fast manner. This is achieved by keeping a list of standby senders, which become active if a sender disconnects. The Streaming Control Layer performs bandwidth aggregation, data collection and status change of the peers. The buffer control mechanism is the key challenge because of the P2P network dynamics. Data Collection Buffer collects data from peers and feeds the Playback buffer. The Playback buffer feeds the Decoder buffer, which are coordinated by the Control Buffer. The Control Buffer is responsible for the synchronization of the data collected from the peers. This data is then fed back to the Decoder Buffer, which feeds the Display Buffer. A double buffer for the display uses the player efficiently and minimizes the switching overhead. Experiments show that the buffer management mechanism works well, in case a peer supplying media is disconnected. There is no interruption even during the recovery by changing the sender to include the standby sender.

In P2Cast [14], the aim is to construct an application overlay appropriate for streaming and providing continuous stream playback without glitches even when some clients depart. A patching technique relying on unicast connections among peers is proposed. The peers that have contacted the source to receive the stream within a certain threshold form a session and cooperatively stream the video content. The peers in a session supply patches to later joining peers to the same session. The streaming tree is constructed regarding to the bandwidth availability of the connections and only using local information (i.e., about parents and children). Continuous playback is supported by using *shifted forwarding.* In this technique, a peer re-joining the base tree after its parent departed requests the missing parts from the new parent. Instead of current video data, the parent forwards the lost parts. Continuous playback occurs if sufficient time is waited. The threshold of the sessions can be adjusted to balance between scalability and quality. Base stream quality is higher, when the threshold is small (i.e., small sessions and small patches). If the threshold is large, more clients can be supported by forming large sessions, but the quality is decreased because of large patches.

P2VoD [15] is a system for Video-on-Demand that is similar to the P2Cast. In P2VoD, peers cache the most recent parts of the video and failures are handled locally using generations of peers. A generation is a group of clients, who always have the same smallest numbered retrieval block in their cache. If a failure occurs, the children of the failed peer are adopted by the peers that are in the same generation of the failed client. The performance comparison with P2Cast shows that P2VoD is better in criteria such as client rejection probability, server workload, failure probability and failure overhead.

2.2.2 Video Encoding: Multiple Description Coding (MDC)

SplitStream [16] aims to distribute the load of the overlay multicast evenly to all the peers. This is achieved by striping the content into multiple parts and building multiple distribution trees. To distribute the load among the interior nodes, these tree are built disjoint, so that one peer in the interior of one distribution tree would be a leaf in the others. Employing MDC, this forest of interior-node-disjoint multicast trees is made to be resilient to failures. It is built using Pastry [17], which is a scalable, self-organizing structured P2P overlay network similar to Chord [18] and Scribe [19]. They showed that SplitStream distributes load more evenly on the links than application-level multicast systems using a single distribution tree.

CoopNet [20, 21] can support both, VoD and live media streaming. In [20], the authors propose CoopNet, as a hybrid approach integrating the client-server and P2P models for both on-demand and live streaming media. Their aim is to complement the client/server architecture to decrease the load on the server and distribute it among the peers. Clients make use of caching content they viewed recently for on-demand media. For live media, they form a distribution tree rooted at the source (i.e., the server). To prevent disruptions because of peers departing, multiple disjoint distribution trees are formed. Using MDC, failure resiliency is increased. Effectiveness of the system is shown by simulation using the trace of a flash crowd event, for both on-demand and live streaming.

In the extension study [21], the authors aim to solve the distribution of ”live” media via a scalable centralized tree management algorithm, in which creating short, diverse and efficient trees along with quick joins and leaves are the targets. MDC is tuned adaptively by a scalable client feedback, in which the packet loss information is passed up the distribution trees, so that the server is not overwhelmed by all clients, but their direct children (i.e., the number of such children is equal to the number of distribution trees). The quality of the video increases with the number of distribution trees, since the loss resilience increases. The comparison with Forward Error Correction (FEC) shows that MDC is more suitable when all peers experience different loss rates.

2.2.3 Video Encoding: Layered Coding (LC)

Another VoD system is described in [22]. Video content is encoded into hierarchical layers and stored on different peers. A client request for a video is satisfied by streaming different layers from different peers. The number of copies of the layers differ according to the importance of the layers. Different number allocation schemes are proposed and simulated. When the network is heavily loaded, more importance is given to the lower layers, so that each peer can receive it. On the other hand, with no congestion in the network, higher layers are given more importance and the probability of a peer receiving all layers increases. The replacement time if a peer disconnects, in the highly loaded network is longer, so that MDC with FEC has better quality, because of MDC's loss resilience. In lightly congested network, where the replacement time is not that high, layered encoded video has better efficiency and performance.

The authors in [23] present a centralized P2P algorithm. A layered video coding technique based on 3D Discrete Cosine Transform is used along with the proposed P2P Streaming Protocol to guarantee Perceived Quality of Service. The hybrid approach works like a client/server architecture where the base and enhancement layers of video are sent to all clients from the server. In congestion, only the base layer is sent to the clients, whereas the enhancement layers are obtained from peers that already received them.

In [24], a layered P2P streaming mechanism for on-demand media distribution is proposed. This work points out the asynchrony of user requests and heterogeneity of peer network bandwidth. As the solution, cache-and-relay and layer-encoded streaming techniques are proposed. The solution has been shown to be efficient at utilizing peers' bandwidth, scalable at saving server bandwidth consumption, and optimal at maximizing streaming quality of peers.

2.2.4 Construction of the Data Network

A classification of the application layer multicast protocols according to their construction of the data network is as follows [25]:

- Receiver-driven: The receiver actively decides, which peers should be the senders of the desired video. Multiple sources contribute parts of the content and the receiver puts them together.

- Source-driven: There is a single source of the content and peers cooperatively stream the content from the source to the other receivers.

- Data-driven: No clear direction for the data flow is defined.

In the receiver-driven approach, protocols make the data plan as a spanning tree with the receiver being the root, so that afterwards they can organize the senders. During this, encoding of the media content (e.g., MDC or LC) is necessary, since each sender can send a different part of the stream. One example for the receiver-driven approach is GnuStream [13] which is built on top of Gnutella. Although the authors of GnuStream do not state explicitly the use of MDC or LC, their buffer mechanism is responsible for collecting data from the peers (i.e., the senders), aggregation and synchronization.

Source-driven approach: PeerCast

PeerCast described in [26] and [27] is a tree-based overlay network, that aims to distribute live content over peers which are very dynamic and unreliable in terms of the duration of their participation in the multicast scheme. They point out that current performance metrics, such as link stress and relative delay penalty, do not reflect end-system performance. Time-to-first-packet is one such metric. Peering layer between the transport level and application level of the TCP/IP stack is introduced, in order to hide the details and results of the overlay topology changes from the application level, which is just responsible for streaming the video. Peering layers of different peers establish data transfer sessions to manage the transience of peers (e.g., joining, leaving, failing). The peering layer takes care of these events by using a redirect primitive in which the peers are redirected to the parent, children or to the source by the correspondent peer. It is shown through simulation that the redirect primitive can handle a join request much faster than the traditional client-server architecture, in which the server has to queue all the requests and reply them one-by-one. Thus, the time-to-first-packet is decreased. Since the application sessions are established between the peering layer and the end application, PeerCast has a flexible structure and thus, can work with any of the existing streaming applications.

Data-driven approach: DONet & AnySee

In contrast to receiver-driven and source-driven approaches, a data-driven overlay network for P2P live media streaming is presented in DONet [28]. In a data-driven approach, there is no specific direction (i.e., the connections are used in both directions) of the flow of the content. The basic idea is that each peer exchanges information with its neighbor about data availability. If it misses data, it retrieves it from the peer. If the neighbor misses data, it sends it to the neighbor. The design is simple since no global structure is maintained; efficient since data transfer depends on the dynamical availability information; robust and resilient, as the neighbor switching can be done quickly when a node fails. The membership management algorithm depends on gossiping, so that each peer exchanges information with a random partial group of peers, which also does the same. Simulations show that a group of 4 neighbors gives quite good streaming continuity and rate. Comparisons with a tree overlay structure shows that DONet is more robust to client departures/failures, thus giving better streaming continuity. An application is implemented and downloaded by about 30,000 people. Statistical usage and feedback to the authors show that the larger the overlay size, the better the streaming quality. This is expected, since larger overlay gives peers the chance to look for better neighbors (in terms of bandwidth and delay) and increases the probability of availability of the content.

AnySee [29] is a P2P live streaming system that has been in use since 2004 to distribute TV shows, movies and even academic conferences. Previous P2P streaming applications focus on intra-overlay optimizations, so that they can efficiently build a control and data structure in the overlay network. However, these optimizations can achieve well up to a certain point on performance metrics, such as startup delay, source-to-end delay and playback continuity. The authors propose the inter-overlay, so that peers can join multiple overlays. The target is to improve global resource utilization, distribute traffic to all physical links evenly, assign resources based on locality and delay and balance the load among group members. Using the nearest peers, receivers are guaranteed good service quality. Simulations show that inter-overlay optimizations outperform CoolStreaming (i.e., the commercial software for DONet [28]) in resource utilization. One of the interesting findings is that the peers tend to have great patience for large delays, if the streamed content is desired.

Source-driven or Data-driven?

In the source-driven approach (e.g., Narada [7, 8, 9], Nice [10, 11], Zigzag [12], PeerCast [26, 27]), the system builds a control plan among the peers [25]. The data plan is always a spanning tree using the control plan with its root at the sender. In the receiver-driven approach, spanning trees are built at the receivers so that multiple senders can be organized to send data to them. The data-driven approach does not clearly separate both plans; the peers exchange information about the availability of the pieces of the media.

The authors present simulation results of the source-driven approach using PeerCast and data-driven approach using DONet for different metrics, like average time to first packet, distribution of clients in the structure and data packet loss rates with the clients dynamically leaving the system. In each of these approaches, DONet outperforms the other variations of PeerCast (i.e., different cluster sizes). Since peers randomly exchange information with other peers in DONet, the dynamic movement of the peers does not severely affect the data loss rate. Therefore, the authors conclude that data-driven approach is more suitable for designing live P2P streaming applications.

2.3 Video Conferencing Tools

2.3.1 OCTOPUS - A Global Multiparty Video Conferencing System

In [30], a global multiparty video conferencing system is presented. It is a two-tier scheme: Local Video Conference (LVC) and Global Video Conference (GVC). LVCs consist of participants that are geographically close to each other (a few kilometers) and their architectures can be distributed or centralized. GVC does not require or assume that LVCs use a fixed architecture, thus allowing flexibility. OCTOPUS is a video conferencing application for closed participant groups that connects local video conferences to make a globally connected one. GVCs make use of a group communication protocol running on the IP multicast. Each LVC selects a group coordinator, which handles the connections between the other LVC group coordinators. Group coordinators are responsible for setting up a global conference, connecting the LVC to a global conference, managing the QoS, exchanging information with other group coordinators about participants who join or leave, receiving video streams from other LVCs and supplying floor control, to queue the participants who request to speak.

Bandwidth restrictions of the links between LVCs are determined through measurements by and among LVC group coordinators. When bandwidth is not sufficient to stream video signals of all participants to all other participants, the *visibility problem* emerges. Depending on the bandwidth, a limit on the number of the participants who can be shown is imposed. A voting scheme is applied to decide which participants are mostly desired to be viewed [31]. The speaker is always shown with a good quality (i.e., 12 frames per second). The rest of the bandwidth is divided between the other participants who collected the most votes. (i.e., up to a certain number of participants are shown). Group coordinators prioritize streams of the speaker and the participants who are given importance levels through voting in order to match with the bandwidth restrictions of the links.

Since the system is based on the connections established among the group coordinators (i.e., the LVCs are connected through group coordinators), it suffers from single-point-of-failures. Although detection and recovery is possible, it may take some time before the LVC (whose group coordinator had failed) and its participants join the global conference again. The flexibility of allowing LVCs having distributed or centralized architectures seems to be weaken by this. Also, the group communication protocol run between the group coordinators depends on the IP multicast, whose deployment is not that wide.

2.3.2 3D Video Conferencing using Application Level Multicast

A 3D video conferencing application using an ESM is reported in [32]. The authors point out that the current applications for P2P media streaming are focusing on distributing a video content, be it on-demand or live, from a single source to a large group of receivers. However, a video conferencing application targets a smaller group, mostly from 4 to 10 participants, where each of these participants can be the source of a video signal.

They propose an awareness-driven video model where a participant is seeing other participants whenever they are in its field of view. They prefer a centralized algorithm for tree construction for the recipients since it can react faster to changes during join or leave events. Rendezvous Points (RP) decide how the multicast trees for different sources are built and which peer will be acting as a parent for another peer. The algorithm also considers the fan-out of the peers, so that the system does not allow the total number of streams requested to exceed the total number of fan-outs of all peers (e.g., in a 4 participant conference, there can be a total of 12 video requests; as each participant at the maximum can see every other participant in the conference). In a 4-participant conference experiment, they show that the total number of video requests in the system at any time is mostly under 6, hinting that a video conference participant at most desire to view two other participants.

During the construction of the trees, RPs can decide that some participants act as *reflector* points, in the situation, that the corresponding participant does not want to stream a video signal, but has to forward it to another participant. This makes use of the idle fan-out that the reflector possesses. Despite this, some video requests may still be rejected.

Although using a centralized approach decreases reaction time to changes and makes good use of the idle bandwidths, it suffers from single-point-of-failures. Controlling multiple multicast trees also increases operational complexity. Another disadvantage is that some participants' fan-out bandwidths are used without their consent.

2.4 Other Aspects

2.4.1 Formal Specification & Verification

A formal description technique is LOTOS which is a standard for the design of distributed systems in particular for OSI services and protocols [33]. While other description techniques are based on state representations of the systems, LOTOS defines the temporal relations between externally viewable events. LOTOS allows concurrency, non-determinism, synchronous and asynchronous communications descriptions. For specification, verification and code generation support, toolsets called EUCALYPTUS [34] and APERO exist as well [35].

Several tools for model checking exist. One of the most popular is SPIN model checker [36]. It is an automata-based model checker in which the system is written in Process Meta Language (PROMELA) and uses Linear Temporal Logic (LTL) for properties to be verified. With options such as partial order reduction [37] or bit-state hashing [38], the model check process can be accelerated and done more efficiently. SPIN is a suitable model checker for asynchronous distributed algorithms and non-deterministic automata, and thus used to model check for various systems and communication protocols [39, 40].

Temporal Logic Checker (TLC) is a model checker that verifies TLA+ specifications. TLA+ is based on TLA, Temporal Logic of Actions, which is fully described by Lamport in [5] and adds constructs that can be used in the formal specification process. In TLA, the systems as well as their properties are represented in the same logic. TLC can be used for concurrent and reactive systems described in TLA+ specifications [6]. TLA+ and TLC can be used in different types of systems [41, 42, 43].

As a comparison, the main advantage of SPIN model checker is that the systems can be defined and described with traditional C program-like input files. It is designed for verification of software systems. Since the code is written like in a programming language environment, expressing formulae is hard. On the other hand, TLA+ can provide formulae in an easier manner. Sets, functions and temporal operators are embedded in the language which makes it more complex yet powerful, but in the mean time harder to understand and use. The generated specifications then can be used in TLC to be verified. Therefore, TLA+ is chosen to specify our system and TLC to verify it.

2.4.2 Security

Shifting the multicasting burden from the routers to the end systems is beneficial in terms of scalability; however, since every end system forwards received data to other hierarchically lower systems, security concerns may be introduced. The intermediate peers may alter the received data and forward them so that the receivers in the lower levels of the hierarchy may encounter modified data. One way to prevent this would be using encryption. The video would be encrypted by the sender and decrypted by the receivers. This is a challenging task, since this needs to be achieved in a limited time to meet the real-time streaming demands.

To enhance performance that is brought by encryption of the entire data packet, Spanos et al. [44] and Li et al. [45] propose encrypting only the I-frame that are used as reference frames for other frames. However, it was shown that the performance enhancement is not considerable and the video is recoverable using only B and P-frames [46].

The Adaptive Rich Media Streaming System (ARMS) uses Advanced Encryption System (AES) to stream MPEG-4 video [47]. End-to-end security is established while content is adapted to the network conditions and streamed over untrusted servers. Because of its efficiency and low overhead, AES is also pointed out in RFC3711 [48] as standard encryption and decryption schemes that could be used in RTP [49] are described.

To meet the real-time demands, compression and encryption may be done at the same [50, 51]. Compression-independent algorithms [52] may also achieve fast performance. All these techniques make use of a shared secret key that enhances performance. However, the same key is used both for encryption and decryption. In an application layer multicast session where the sender and receivers share the same secret key, the authentication of the sender is not possible.

Timed Efficient Stream Loss-tolerant Authentication (TESLA) allows receivers in a multicast group to authenticate the sender [53]. It requires that the sender and receivers are loosely time synchronized meaning that the receivers know an upper bound of the sender's clock. Although encryption is done with symmetric keys, they are not known by the receivers until the sender discloses it. So the receivers need to buffer incoming packets until the key is disclosed. Non-repudiation is not provided.

In [54], chaining techniques for signing and verifying multiple packets (i.e., a block) using a single signing and verification operation are proposed. This way, the overhead is reduced. The signature-based technique proposed does not depend on the reliable delivery of packets, but uses caching of the packet digests in order to verify the other packets in the block efficiently.

In our scheme, no caching is required and verification is done per packet basis. Besides the authentication and integrity of the sender, it also provides non-repudiation. Simulations show that this can be done fast enough even for large key sizes.

2.5 Remarks

P2P streaming research can be classified according to the network architectures and video coding techniques, like done in [55]. The architectures differ in the number of their senders, namely multiple sources and single source. In the multiple source architecture, there may be intermediate peers involved between the source and the destination; however, their role is limited to the transfer of the received packet. On the other hand, in a single source architecture, the intermediate peers are also the destination of the packets and they cache some part of the packets in their internal buffers, so that requesting peers can obtain the content from an intermediate peer directly as well as from the source. This helps distributing the load among the peers. Two video coding techniques are generally used. MDC is more loss resilient than the LC; however, LC has better coding efficiency, so that network resources are used more efficiently. Besides a brief literature review of the existing applications, they point out that a successful P2P streaming application should have a method for deciding the appropriate coding scheme, the peer dynamicity and heterogeneity, efficient overlay construction, best peer selection, adapting the network conditions and to take measures to encourage participation of peers.

The system presented in this thesis gives the flexibility of using two different video coding techniques. As long as their bandwidth requirements are the same, either MDC or LC can be used within the application to generate two layers of video. Since a small group of participants is considered, the assumption is that the peers are static most of the time, so that the reconstruction of the chain is not done very frequently. Their heterogeneous bandwidth distribution is considered in Section 3.3 and in Chapter 5 of this thesis. Optimization techniques are given to construct the overlay optimally for different metrics. They aim to:

- minimize the number of base layer receivers
- minimize the maximum delay experienced in a chain
- maximize the number of extra requests

The overlay construction and best peer selection, in the sense that the objective functions of the optimizations are achieved, are done in a distributed manner. Adapting to network conditions, such as bandwidth availability, is not considered, since peers are assumed to have fixed low-bandwidth connections. Peers are taken as cooperative (i.e., they agree to forward another's video), but security measures to assure integrity, authentication and non-repudiation are taken and explained in Section 6.2.

In [56, 57], the authors point out that the assumption in all of the application layer multicast protocols is that the peers have symmetric bandwidths. In real world, the asymmetry is more common and because of this assumption the type and quality of the streamed content is limited. They propose a technique in which the peers supply as their upstream bandwidths allow them. The video quality is not degraded because of the output restriction, rather output bandwidths of different peers are aggregated at the receiver peer to keep the quality of the video as intended. They propose mesh optimizations to limit the length of the mesh, the network cost and make use of client buffers and strategic nodes (i.e., stable nodes in higher levels). Simulations show that the service is available for about 95% and is increasing as the latency increases.

Although the assumption throughout this thesis is that peers possess bandwidth that is enough for sending and receiving only one video signal at a time, this may not be generally true in real-life. The authors in [56, 57] use the asymmetry of the peers' bandwidths to increase the quality of the video streamed, which in turn increases user satisfaction. In this thesis, the bandwidths of the peers will be spoken in an input/output basis. (i.e., if the video quality is assumed to be 64kbps, and a peer has 128kbps downstream and 64kbps upstream, then the peer is said to have 2 inputs and 1 output.) A peer having two inputs is handled so that it can make two requests. A peer having multiple outputs will help other peers receive full quality video. In general, multiple inputs/outputs are used to increase user satisfaction. Three or more inputs having peers, however, will not receive the third request. This restriction is to guarantee that every request can be granted in the entire conference. Also, like argued in [32], trying to view more than two users simultaneously is most probably not realistic and is not considered. Multiple inputs/outputs cases are discussed in Section 3.3 and in Chapter 5.

Chapter 3

PEER-TO-PEER MULTIPOINT VIDEO CONFERENCING USING LAYERED VIDEO

The assumption of the P2P approach for multipoint video conferencing described in [3] is that the participants (i.e., the peers) could be the source of one video signal. Also, they could only receive and send one video signal at a time. In other words, they have the computational power to produce one video signal and their bandwidth is limited and only enough for streaming one video signal upstream and downstream. This way, the networking and computing resources of an MP video conference do not exceed the needs of a point-to-point video conference.

This requires that some of the peers may have to forward the video packets they receive to another participant. In previous work [4], the following definitions were made:

- Chain: The ordered group of peers that receive the same video signal.
- Head of a chain: A peer sending its own video signal.
- Relay: A peer that is forwarding the video signal it is receiving, to another peer.

The restriction that the upstream bandwidth is only enough to send one video signal may cause a request to be denied. This is because a peer cannot send its own video signal if it is relaying another peer's video signal and a peer cannot relay if it is sending its own video signal. In [4], it was shown that this is not the case most of the time, and in fact each participant could view another participant's video under *most* practical cases. However, the number of the cases in which a request is denied

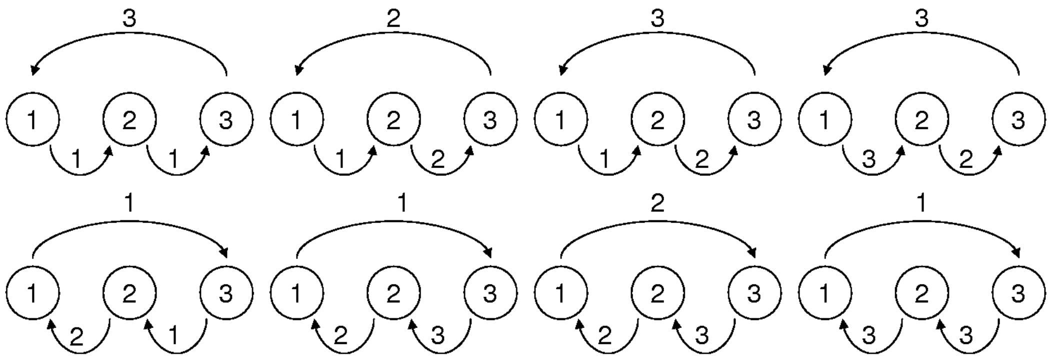


Figure 3.1: Configurations for a three-participant conference (Numbers on arrows indicate which video signal is used in that upstream). Each participant can see any other one.

increases as the number of the participants increases. This causes the system to be unscalable.

The cases that could emerge on a video conferencing with three participants are depicted in Figure 3.1. As can be seen, there is always a solution to grant every participant's every request. However, when a fourth user requests the video of an intermediate peer, like shown in Figure 3.2 (a), the request cannot be met since the upstream bandwidth of the intermediate peer (i.e., peer 2) is already in use.

Our approach in this thesis study makes use of two layers in a video signal, and allows that each participant can view any other participant's video at anytime for any number of participants. This is accomplished by allowing a peer to be a relay and a head of a chain at the same time. One of the video layers is the *base* and the other is the *enhancement*. The two layers have almost equal bandwidths, so that sending a base layer and an enhancement layer, is not different from sending two different base layers. A participant receiving a base layer and the corresponding enhancement layer is able to view the video signal in full quality whereas one receiving only the base layer can view the video in reduced quality. This allows to achieve a solution under *all* cases without having to reject any video request from any participant at anytime.

The employment of layered video and how it solves the described problem can

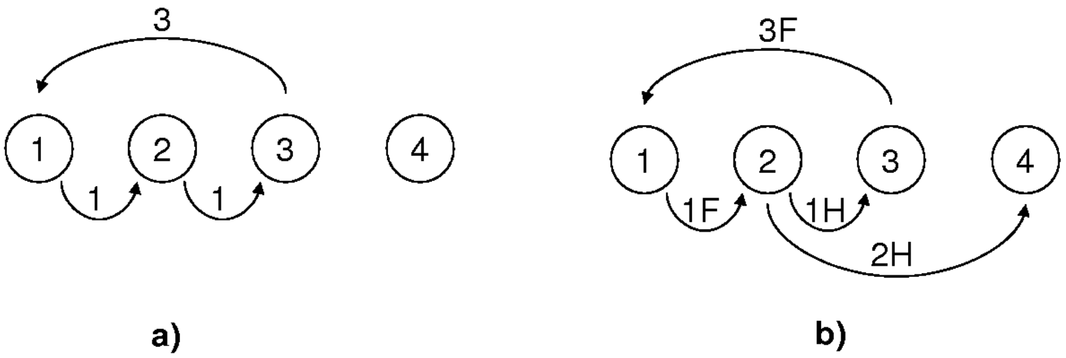
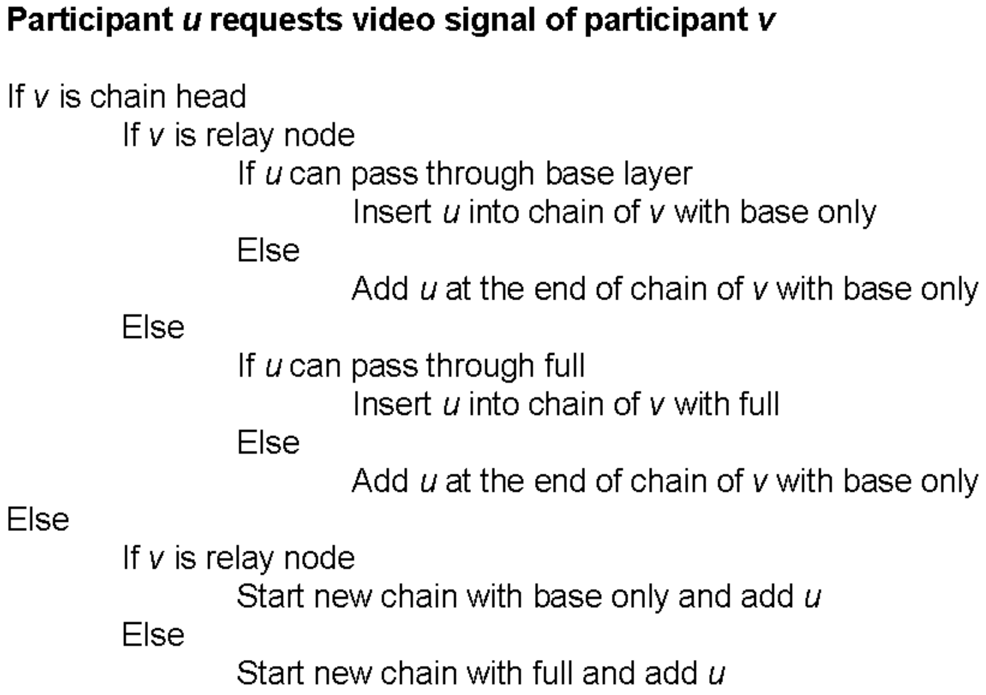


Figure 3.2: a) Participant 4 can not see participant 2. b) Participant 4's request can be granted. (F stands for full quality video (base & enhancement), H stands for half quality video (base))

be seen in the example of Figure 3.2 (b). The pseudo code describing the actions performed by a participant who receives a video request is given in Figure 3.3.

Since the system is based on a P2P paradigm and is software only, it does not need any particular server functionality or special hardware that may be prone to single-point-of-failures. It does not depend on network multicast infrastructure support which may or may not be present. Hence, the use of a distributed P2P architecture brings the advantage of enhanced fault tolerance. Another fault tolerance related issue is that peers can leave the system at anytime, either at will or unintentionally (i.e., by crashing). A peer may be sending its own video signal (i.e., the peer is a chainhead) when this happens. The members of the chain would immediately notice that something is wrong and would act accordingly (i.e., stop receiving the video signal of the peer that has just left the system and may request another peer's video signal). Another possibility is that this peer may be a relaying peer in a chain. The chainhead would notice and rearrange its chain accordingly when this is the case, since members of a chain periodically send keep-alive messages to the chainhead. If a peer does not coordinate on time with its chainhead, the chainhead removes the peer from its chain and makes the necessary changes (i.e., sends update messages to



Participant *u* requests video signal of participant *v*

```
If $v$ is chain head
    If $v$ is relay node
        If $u$ can pass through base layer
            Insert $u$ into chain of $v$ with base only
        Else
            Add $u$ at the end of chain of $v$ with base only
    Else
        If $u$ can pass through full
            Insert $u$ into chain of $v$ with full
        Else
            Add $u$ at the end of chain of $v$ with base only
Else
    If $v$ is relay node
        Start new chain with base only and add $u$
    Else
        Start new chain with full and add $u$
```

Figure 3.3: Algorithm for granting a video request.

appropriate peers). Messages that are used in our prototype system are explained in Chapter 4. A peer that is both a chainhead and a relaying peer would not cause a problem either when it leaves or crashes. The members of the leaving peer's chain would react by stopping to receive the video signal; the chainhead would rearrange its chain accordingly.

Next section gives details of layered video solutions and how the base and enhancement layers can be generated. Section 3.2 explains the optimizations that can be used a) to minimize the number of lower quality (i.e., only the base layer) receivers and b) to minimize the maximum delay experienced in a chain along with the number of base layer receivers at the same time. The effects of multiple output bandwidths that allow more than one video signal to be sent simultaneously, are investigated in Section 3.3.

3.1 Layered Video

The proposed algorithm requires that the video signal is coded in two layers, such that it consists of a base layer and an enhancement layer. The video is in full quality, if the base and the corresponding enhancement layers are received. If only the base layer is received, the video can be viewed in an acceptable quality.

The base layer is used whenever the intermediate peer, the relay, receives a video request from another participant. The relay stops forwarding the video signal it is receiving in full quality, that is, with the base and the enhancement layer, but continues to forward only the base layer. The rest of the bandwidth is used for its own video signal and sent to the participant that requested the relay's video signal. This way, any participant can see any other participant at any time and any configuration of requests.

This statement makes an assumption about the bandwidth of the layers: The bandwidth of the base layer and the bandwidth of the enhancement layer are equal. This allows a relaying participant to stop forwarding the enhancement layer of the video signal it is receiving and then sending its own video signal in base layer.

In this section, two approaches for layered video are described:

- Scalable video approach
- Multiple description coding approach

We will explain the details of these approaches and how they can be achieved with standard encoders, such as JSVM [58] and Nokia [59].

3.1.1 Scalable Video Approach

Scalable video coding techniques are gaining popularity with H264/SVC allowing encoding of the video in different quality layers so that according to the bandwidth restrictions corresponding layers can be transmitted [60]. Quality is increased by

Table 3.1: Average PSNR and bit rate values: Temporal scalable base and full quality layers.

Quality	Average PSNR(Y)	Average PSNR(U)	Average PSNR(V)	Average Bit Rate (kbps)
Single Layer	34.3292	38.7423	40.0163	63.7320
Base Layer	31.8586	37.4623	38.5050	31.5066
Base + Enhancement	33.6630	38.8204	40.1227	63.9414

using more layers. There are three aspects of scalability, namely spatial, temporal and SNR.

Spatial scalability is achieved by changing the resolution of the video. Since this is done in two dimensions, switching from one standard resolution (QCIF, 176x144) to another (CIF, 352x288) increases the bandwidth requirement much more. The bandwdith requirement of a CIF video is *approximately* four times the bandwidth requirement of a QCIF video. In our approach we need two layers with equal bandwidth requirements, and hence we do not employ spatial scalability.

Temporal and SNR scalability are explained next.

Temporal scalability

The bandwidth requirements of each layer should be equal. This is achieved simply by dividing the video stream in the temporal dimension. Base layer will have half the frame rate of the original video. Using JSVM [58], two temporal layers can be created. We are assuming a base layer of 7.5 fps. Full quality video would then have 15 fps frame rate. The results of the encoding of a standard sequence (i.e., Foreman) and the available layers of the resulting bitstream can be seen in Table 3.1 and 3.2, respectively. The last column (i.e., DTQ) stands for **D**ependent layer id, **T**emporal layer id, **Q**uality layer id).

Table 3.2: Bitstream contents of the full quality layer with temporal scalability.

Layer	**Resolution**	**Frame rate**	**Bit rate**	**Minimum Bit rate**	**DTQ**
0	176x144	0.9375	13.00	13.00	(0,0,0)
1	176x144	1.8750	17.52	17.52	(0,1,0)
2	176x144	3.75	23.52	23.52	(0,2,0)
3	**176x144**	**7.5**	**30.52**	**30.52**	**(0,3,0)**
4	176x144	0.9375	28.00	28.00	(1,0,0)
5	176x144	1.8750	36.12	36.12	(1,1,0)
6	176x144	3.75	45.12	45.12	(1,2,0)
7	176x144	7.5	55.12	55.12	(1,3,0)
8	**176x144**	**15**	**63.12**	**63.12**	**(1,4,0)**

SNR scalability

JSVM [58] also allows SNR scalability to be used so that one can fine-tune the quality of the video by using Fine Granular Scalability (FGS) layers. To achieve SNR scalability, JSVM uses progressive refinement (PR) slices whose coding symbols are ordered by their importance. PR slices can be truncated at any point, providing a rate interval rather than rate points [58]. So, one can create a bitstream with one FGS layer and extract the base layer when needed. The encoding results can be found in Table 3.3. Although the bit rate of the layers is not very high, the quality it provides is acceptable. The base layer can now be extracted to have a bit rate about the half of the full quality video, so that the frame rate is still the same and the bandwidth is equally shared between the base and the enhancement layers. This can be verified from Table 3.4.

3.1.2 Multiple Description Coding Approach

Multiple description coding (MDC) is another alternative for transmitting video [61]. In this approach, odd numbered frames and even numbered frames may be predicted

Table 3.3: Average PSNR and bit rate values: SNR scalable base and full quality layers.

	Average	**Average**	**Average**	**Average**
Quality	**PSNR(Y)**	**PSNR(U)**	**PSNR(V)**	**Bit Rate (kbps)**
Single Layer	34.3292	38.7423	40.0163	63.7320
Base Layer	30.3959	36.9938	37.7932	31.7730
Base + Enhancement	32.9896	38.4540	39.6059	64.1106

Table 3.4: Bitstream contents of the full quality layer with SNR scalability.

Layer	**Resolution**	**Frame rate**	**Bit rate**	**Minimum Bit rate**	**DTQ**
0	176x144	1.875	15.00	15.00	(0,0,0)
1	176x144	1.875	35.00		(0,0,1)
2	176x144	3.75	20.00	20.00	(0,1,0)
3	176x144	3.75	45.00		(0,1,1)
4	176x144	7.5	25.00	25.00	(0,2,0)
5	176x144	7.5	54.00		(0,2,1)
6	**176x144**	**15**	**31.00**	**31.00**	**(0,3,0)**
7	**176x144**	**15**	**64.00**		**(0,3,1)**

only from each other, creating two independently decodable threads. This approach may also be used to increase the system's loss resilience by allowing forwarding of the description experiencing smaller number of packet losses at the relay nodes. The quality is increased if more than one description is received. However, there is some redundancy in the layers because of the independent decodability feature. This redundancy may cause MDC approach to have an encoding that is not as efficient as the scalable coding. Therefore, the quality of base layer video or full video (i.e., both layers have arrived at the receiver) may not be as high as the scalable coded video when both layers are received.

The results of MDC approach can be seen in Table 3.5 and in Table 3.6. As one can see, JSVM has better results in terms of PSNR than Nokia Encoder, because in contrast to Nokia Encoder, JSVM uses two reference frames for encoding a frame. However, JSVM is not fast enough to meet the real-time requirements. On the other hand, Nokia works fast enough to encode frames at 15 fps.

Table 3.5: Average PSNR and bit rate values: MDC scalable base and full quality layers. (JSVM)

	Average	**Average**	**Average**	**Average**
Quality	**PSNR(Y)**	**PSNR(U)**	**PSNR(V)**	**Bit Rate (kbps)**
Single Layer	34.3292	38.7423	40.0163	63.7320
Base Layer (even frames)	32.5192	37.8689	38.9798	31.6908
Base Layer (odd frames)	32.4926	37.9254	38.9997	31.6272
Combined (even + odd frames)	32.5059	37.8972	38.9898	63.3180

Table 3.6: Average PSNR and bit rate values: MDC scalable base and full quality layers. (Nokia)

	Average	**Average**	**Average**	**Average**
Quality	**PSNR(Y)**	**PSNR(U)**	**PSNR(V)**	**Bit Rate (kbps)**
Single Layer	34.3292	38.7423	40.0163	63.7320
Base Layer (even frames)	30.2404	37.0603	37.8799	32.045
Base Layer (odd frames)	30.4107	37.0932	37.9618	32.167
Combined (even + odd frames)	30.32555	37.07675	37.92085	64.212

3.2 Optimizations - Minimizing the Number of Base Layer Receivers

Chain configuration optimizations can be performed in order to maximize the number of participants that receive full quality video in a particular configuration. In other words, let n be the number of participants and r be the set of video requests, defining

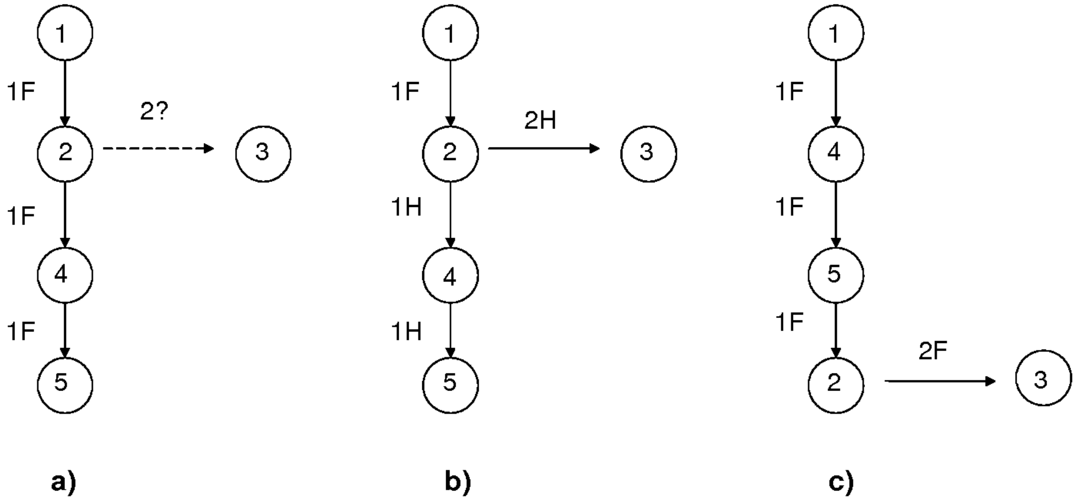


Figure 3.4: An example of chain optimization a) Participant 3 requests a relaying participant's video. b) Chain without optimization. c) Chain after Participant 2 is moved to end.

a particular configuration. The number of full quality receivers is $k = f(n, r)$. The aim of the optimizations is to maximize k, when the number of participants changes and/or the request of a participant changes, namely $k' = f(n', r')$.

For example, assume that a participant (e.g., 2) is relaying another participant's (e.g., 1's) video signal at full quality, i.e., base and enhancement layers. When yet another participant (e.g., 3) requests 2's video signal, 2 has to drop the relayed video signal (of 1) to base layer and forward only the base layer, so that it can send its own video to 3. This makes all participants receiving the relayed video from 2, and participant 3 to receive base quality video. Assuming that 2 is located right after 1 in a long chain, letting it relay base quality video would reduce the received video quality for a large number of participants. However, if 2 could be moved to the end of the chain, this large number of participants can continue to receive full quality video. Figure 3.4 shows this situation.

If in the configuration of Figure 3.4, participant 5 was also sending its own video

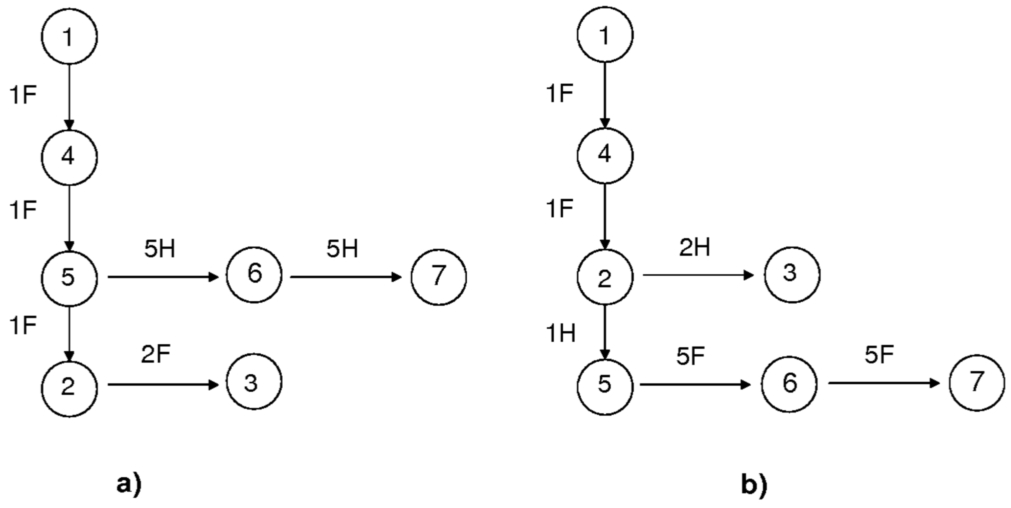


Figure 3.5: a) Chain after Participant 2 is moved to the end. b) Chain after Participant 2 is moved just before the end.

signal, then moving 2 to the end of the chain would cause all the participants in 5's chain to receive base quality video. In this case, chain lengths for 2 and 5 can be compared and the participant with a longer chain can be moved to (or left at) the end. For instance, if 5 has a chain length of two, then moving participant 2 to the end causes a total of three participants to receive base quality video (the chain members of 5, i.e., two participants, plus participant 2). On the other hand, moving participant 2 just before the end would cause only two participants to receive base quality video (the participant that requested video from 2, i.e., 3, and participant 5). This situation is depicted in Figure 3.5.

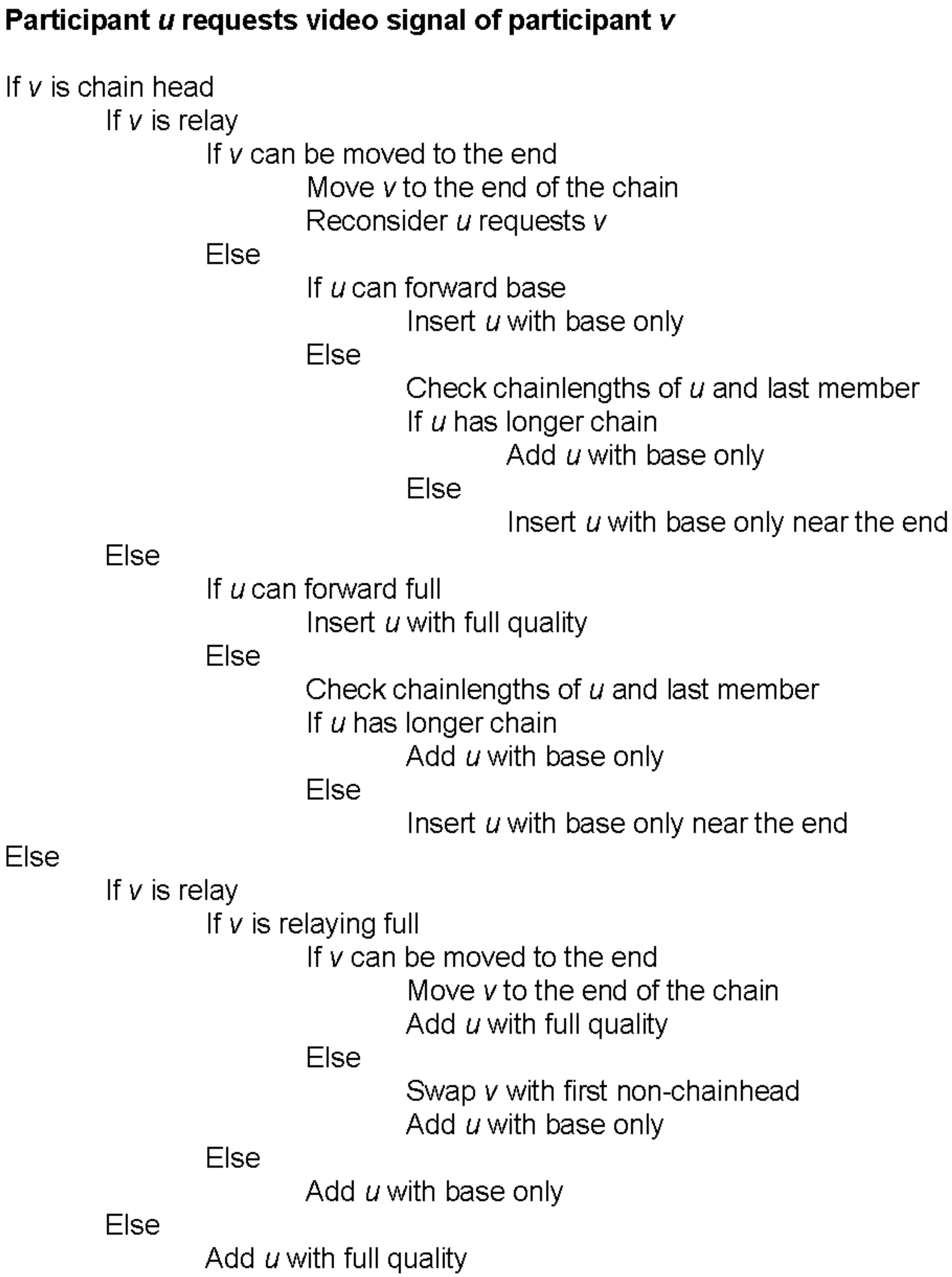
Similarly, when inserting participants into chains, the lengths of the involved chains should be taken into consideration. If a participant is the head of a chain sending full quality video, we should avoid using it as a relay node. As an example, assume that a participant, P, that is a chain head sending full quality video, requests participant T's video signal and T is already sending at base quality. In this case, P could drop its video to base and relay T's video signal. However, doing so would cause the chain members of P receive base quality video. A better solution would be adding

P to the end of the chain, so that it would continue to send its video at full quality. However, if the last member, L, of the chain is also a chain head sending its video, a comparison between P's and L's chain lengths should be carried out. The participant with longer chain would go to the end of the chain, minimizing the number of base quality video receivers. This greedy approach ensures that every time a participant requests video from another one, the configuration stays with maximum number of full quality receivers. The complete decision chart including the optimizations to minimize the number of base layers is given in Figure 3.6.

While making these optimizations, however, some configuration messages need to be sent, because checking chain lengths or mobility of members in the chain require exchanging information between chain head and members. Although these can be done in one message, there may still be delay before a configuration is updated. Therefore, some optimizations may be skipped whenever low delay is more important than quality. The optimizations can be performed after the requested video is provided immediately using a suboptimal configuration.

Another method to prevent this delay is to let the chain head to keep a status table of the chain members, in which the states and chain lengths of the members are stored. This way, the chain head would have all the information it needs to decide whether a member can be moved to the end of the chain or where a new member should be in the newly configured chain. This method however, has a drawback: The members' status need to be updated periodically. This drawback can be overcome by exploiting the keep-alive messages that are sent periodically from the members to the head to let it know that it has not crashed. The user that has just made a video request uses the video request message to send her state and chain length as well. The piggybacked information is used to update the status table.

One other alternative is to assume that if no information update about the state and the chain length of a member is made, it has stayed the same. This time, the overhead that occurs by piggybacking the status information to each keep-alive message can be cut down to only a message that is sent once state change or chain



Participant *u* requests video signal of participant *v*

```
If v is chain head
    If v is relay
        If v can be moved to the end
            Move v to the end of the chain
            Reconsider u requests v
        Else
            If u can forward base
                Insert u with base only
            Else
                Check chainlengths of u and last member
                If u has longer chain
                    Add u with base only
                Else
                    Insert u with base only near the end
    Else
        If u can forward full
            Insert u with full quality
        Else
            Check chainlengths of u and last member
            If u has longer chain
                Add u with base only
            Else
                Insert u with base only near the end
Else
    If v is relay
        If v is relaying full
            If v can be moved to the end
                Move v to the end of the chain
                Add u with full quality
            Else
                Swap v with first non-chainhead
                Add u with base only
        Else
            Add u with base only
    Else
        Add u with full quality
```

Figure 3.6: Complete decision algorithm with optimizations to minimize the number of base layer receivers.

length update occurs.

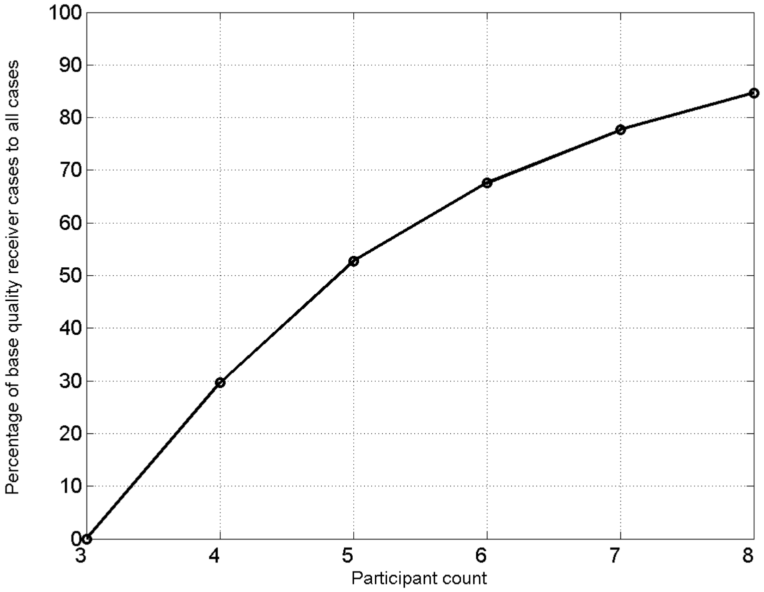


Figure 3.7: Percentage of configurations containing base quality receivers versus number of participants.

In the simulations, all possible cases for a given number of participants are generated. In each case, each participant requests a video signal of any other participant. Using the algorithm given in Figure 3.3 and employing optimizations given in Figure 3.6, the chains were obtained and analyzed.

Figure 3.7 shows that with increasing number of participants, the probability that some participant receives base quality video increases. In Figure 3.8, the percentage of the total number of base quality receivers in all receivers, under all possible cases is shown. Increasing the number of participants would increase the number of possible cases and thus, the total number of base quality receivers as well. However, this increase is asymptotic and the ratio of the average number of base quality receivers to the number of participants decays. This is shown in Figure 3.9.

Simulations show that with increasing number of participants, the use of layers and thus, base quality video receivers is inevitable. However, the ratio of the base

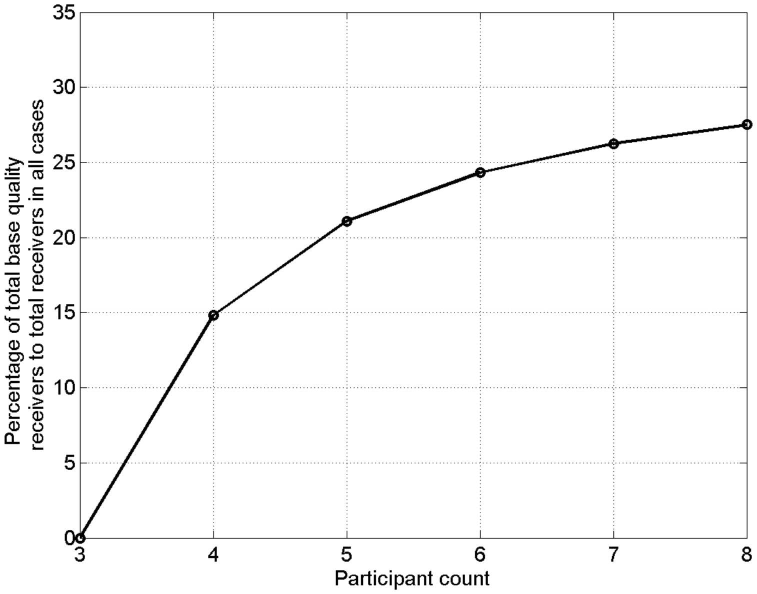


Figure 3.8: Percentage of base quality receivers to all receivers under all configurations versus number of participants.

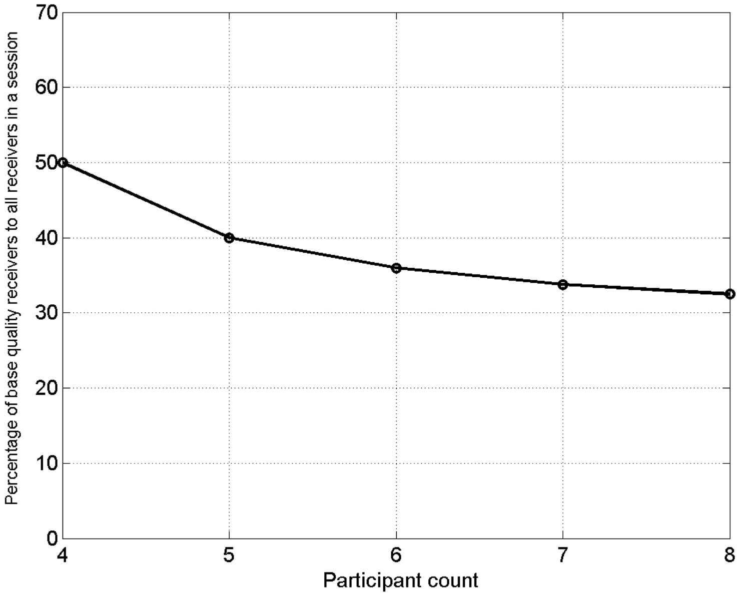


Figure 3.9: Average percentage of base quality receivers versus number of participants.

quality receivers to the total number of participants remains under 50%.

Besides this issue, the geographical location of the users and their heterogeneous connection bandwidths would also play a role on the ordering of the participants in a chain. In that case, the trade-off between maximizing the number of full quality receivers, minimizing the maximum delay any participant experiences and maximizing the number of granted additional requests still remains and can be left to the users' choice. This is investigated in Chapter 5.

3.3 The Effects of Multiple Outputs

The application presented in [3, 62] assumed that the end-hosts possessed an Internet connection that could be enough to send and receive only one video signal. However, the bandwidth diversity of the Internet connections plays an important role in MP video conferencing applications.

The extension presented in [62] showed that two output bandwidth is necessary to be able to grant each and every request at any time. The obstacle emerging from the assumption that the end-hosts do not have enough bandwidth was overcome by using two layers of video, so that virtually two output bandwidths were created. In real life, peers can have bandwidth that can support the case of two or more video signals are received and/or sent. In this section, we investigate the case of multiple video outputs. Effects of multiple inputs will be explained as we describe the multi-objective optimization framework in 5.

When peers have multiple outputs, a relay receiving a video request, does not have to drop the forwarded video quality to half; it just uses the other output bandwidth to send its own video signal. In order to support these peers, the availability of a spare output is checked, before the requester is added to the chain with base quality. If there is, the requester receives full quality. Figure 3.10 shows the algorithm incorporating both single and multiple output bandwidth cases.

Figure 3.11 shows that the increase of the percentage of cases, in which there are base layer receivers, in all cases is much smaller when the maximum possible output is

```
If v is chain head                                                //1
  If v is relay                                                   //2
    If v can be moved to the end of the chain                     //3
      Move v to the end of the chain, reconsider from the start
    Else                                                          //3
      If v has free output                                        //4
        Start new chain with full and add u
      Else                                                        //4
        If u can forward base                                     //5
          Insert u with base only
        Else                                                      //5
          Compare chain lengths of u and last peer of v's chain
          If u has longer chain                                   //6
            Add u to the end of the chain with base only
          Else                                                    //6
            Insert u into the chain with base only
  Else                                                            //2
    If u can forward full                                         //7
      Insert u into the chain with full
    Else                                                          //7
      If last peer of v's chain is chain head                     //8
        If v has free output                                      //9
          Start new chain with full and add u
        Else                                                      //9
          Compare chain lengths of u and last peer of v's chain
            If u has longer chain                                 //10
              Add u to the end of the chain with base only
            Else                                                  //10
              Insert u into the chain with base only
      Else                                                        //7
        Add u to the end of the chain with full
Else                                                              //1
  If v is relay                                                   //11
    If v is relaying full                                         //12
      If v can be moved to the end of the chain                   //13
        Move v to the end of the chain, reconsider from the start
      Else                                                        //13
        If v has free output                                      //14
          Start new chain with full and add u
        Else                                                      //14
          Find the first non-chain head in the chain, swap with v,
          start new chain with base only and add u
    Else                                                          //12
      If v has free output                                        //15
        Start new chain with full and add u
      Else                                                        //15
        Start new chain with base only and add u
  Else                                                            //11
    Start new chain with full and add u
```

Figure 3.10: Algorithm to grant a request. Works for single and multiple output bandwidths (The numbers of the lines are there to increase readability and indicate the corresponding [if] and [else] pairs.).

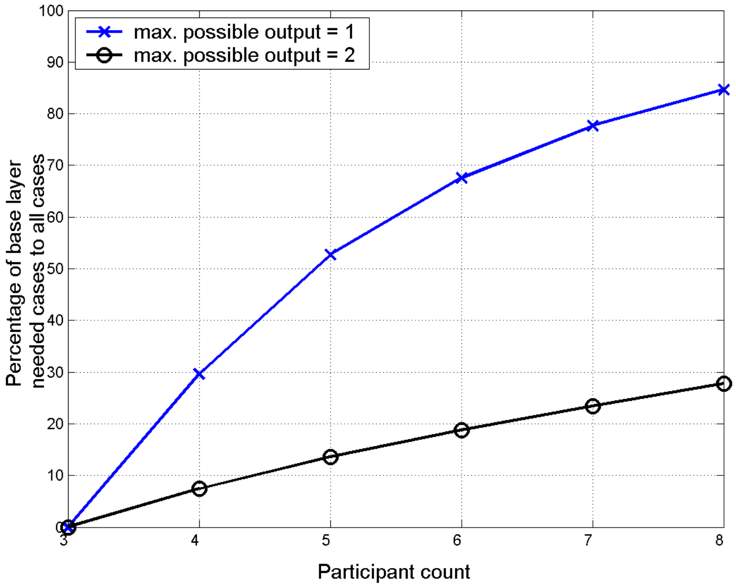


Figure 3.11: Percentage of base layer needed cases in all cases vs. participant count.

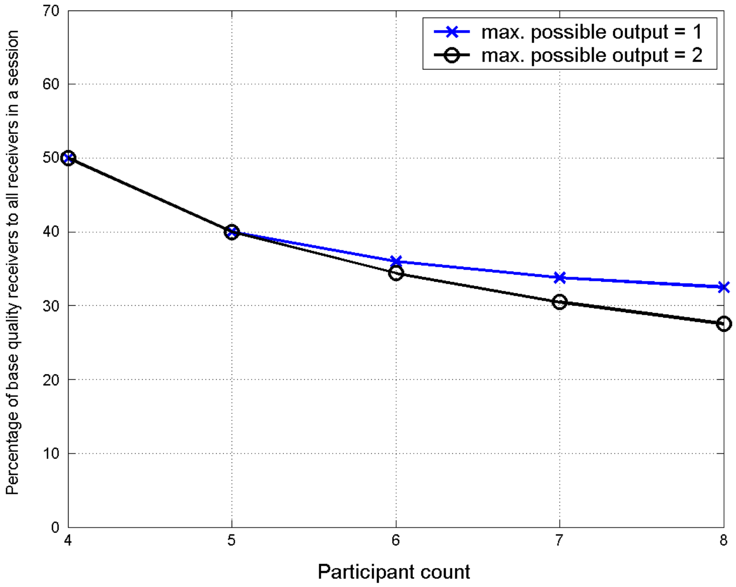
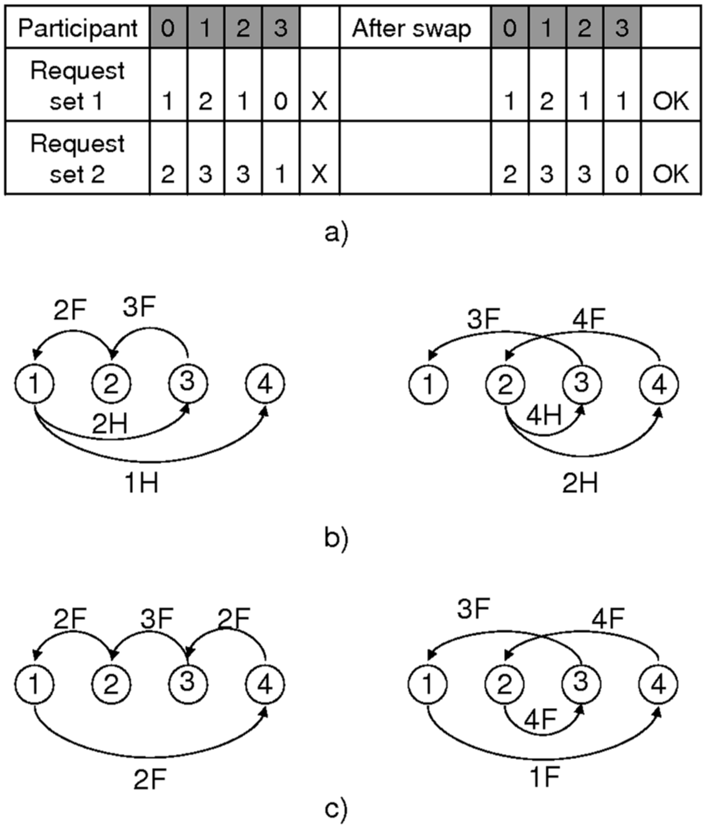


Figure 3.12: Percentage of average number of base layer receivers in a case vs. participant count.

2. This is because the number of the cases with base layer receivers becomes insignificant compared to the total number of configurations that can exist. In addition, the probability of a relay's dropping the forwarded video's quality to base layer decreases, since the spare output can be used for sending the relaying peer's own video signal whenever a request is received. This causes the percentage of average number of base layer receivers in a configuration to decrease as can be observed in Figure 3.12. This is expected, since if a peer has a spare output it can use it whenever a request is received, without the quality of the forwarded video is dropped to base layer.

3.3.1 Special Case: 4 participants, 2 requests, 2 layers



Participant	0	1	2	3		After swap	0	1	2	3	
Request set 1	1	2	1	0	X		1	2	1	1	OK
Request set 2	2	3	3	1	X		2	3	3	0	OK

Figure 3.13: a) Two requests before and after swap. X denotes that the request set needs base layer, shown in b). OK means there is no need for layers. b) Two request sets needing two layers c) Two request sets not needing any layers.

If each participant had two outputs and single input for video, then every request would be granted at any time in a conference. This fact can be generalized: If each participant had more than one input allowing them to request more than one video at the same time, each request set could be thought as a separate conference in which participants are making just one request. This means, if there are two layers of video for each extra request, the request can still be granted at any time, allowing participants to watch more than one participant at the same time. This sets an upper bound for the number of layers required to allow more than one request at the same time. Determining how many layers are actually needed can be considered as a future work.

However, one special case is a conference with four participants, each participant requesting two other participants' video signals. Assuming that each participant has bandwidth enough to send and receive two video signals, it can be shown with enumeration that without using any layers, each request can be granted at any time. In every possible case, a request can be found in a set to be swapped with another request in the other request set, so that both request sets are solvable without needing an extra layer. This way, every participant watches two other participants. An example can be viewed in Figure 3.13. As can be seen, in b) and c) part, the request sets are the same (i.e., the participants request the to view the same participants' video in both configurations). If the requests are taken into consideration like done in b) the use of base quality is required to grant each request. On the other hand in c), there is no such need.

Using two layers created two virtual outputs from one physical output bandwidth. It would be the same thing if each participant had one input and one output bandwidth and two layers of video were used. The only difference would be that the video quality is degraded to an acceptable quality to allow video conferencing even when there is scarce bandwidth available.

Chapter 4

REAL-LIFE APPLICATION

In this chapter, details of the MP video conferencing application described in Chapter 3 will be given. States of the participants, messages exchanged between them, application architecture and interfaces between modules will be explained.

The application presented in this thesis is a P2P MP video conferencing application that makes use of layered video and hence enables each participant to view another participant's video signal at any configuration. The video conferencing part is a P2P architecture, which runs over an IM application. This IM application is used only for the purpose of keeping a record of online peers. These can later create a conference or join a conference through an invitation. We will now describe each part of the implementation in detail.

4.1 Tracker Software

The tracker software's only responsibility is to keep the track of which peers are online. Each peer has an *id* and a *peername* (i.e., username of a peer). They have a contact list which consists of the id's and names of other peers. After the sign-in messages, peers periodically send check-in messages to the tracker. The contact list information is piggybacked at the end of the check-in message. As the tracker receives the check-in message, it updates the status of the peer to ”online”. It also returns a reply with the availability of the peers and their addresses if they are online. The tracker keeps track of each peer and whenever a peer does not check-in in a timely manner or sends a sign-out message, the its status is updated to ”offline”.

4.2 Peer Software

The peer software is responsible for conference management, join, leave, invite, text messaging, video request and video release operations. Furthermore, it handles video module processes and their configurations. It provides a graphical user interface (GUI) which allows the user to interact with the system. After signing in to the tracker, the peer can view the status of the peers in its contact list. It can send private text messages to online peers and invite them to a conference. A conference is established after a peer accepts another peer's invitation. Peers that join a conference are called participants.

Participants can make video requests to one other participant or can release their video sources if they had been already receiving a participant's video signal. Conference Manager module in the recipient's software handles the request and sends the appropriate configuration messages to the corresponding participants. These may be members of the chain of the requested participant, or the participant who has just made the video request. Conference Manager module also updates the participant's state variables. Video module processes are created or destroyed according to the type of the request message and the current state of the participant. Configuration updates on these processes are also made by the Conference Manager module.

4.3 Peer Software Modules

There are two modules in the peer software. The first module, the Conference Manager, is responsible for handling user interactions through a graphical user interaction and conference related interactions with other peers. The second module dealing with video related operations consists of four processes: Capturer, Encoder, Decoder and Displayer. The modules and interfaces are depicted in Figure 4.1.

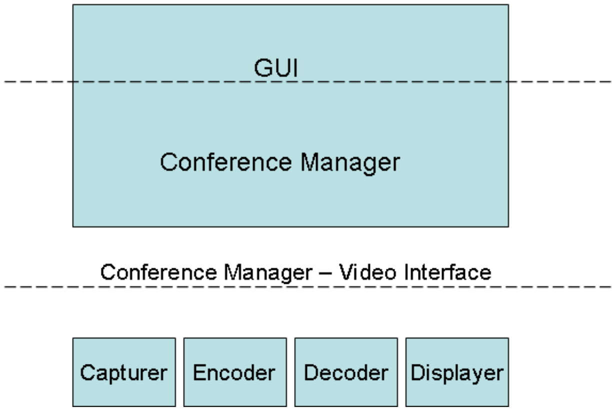


Figure 4.1: Peer modules and interfaces.

4.3.1 Conference Manager Module

Conference Manager is the main application part. It is responsible for tasks that are initiated by the user through the GUI, such as sign-in, sign-out, chat, invitation and video request events and sending out their corresponding messages. Furthermore, it listens for messages from the tracker (i.e., the contact list status information) and from other peers and handles these. It uses a thread-per-request model. Each received message is handled according to the user decisions by the help of the GUI and the current state variables. Whenever a video request arrives, its outcome is calculated by the decision algorithm 3.6. According to the result, appropriate messages are sent. Any changes in the state variables of the participant are used to create, destroy and update video related processes.

4.3.2 Video Module

The Video Module consists of four independent processes. These are created and destroyed by the Conference Manager as needed.

The **Capturer** process' main task is to get the frames from the capture device and feed it to the encoder process. Since our prototype employs an MDC-like approach, even and odd frames are fed to separate ports of the encoder.

The **Encoder** process runs three threads: One is responsible for listening to the configuration changes that come from the Conference Manager module through the Interface. The other two threads independently encode even and odd frames that are supplied from the capturer process. The encoded frames are sent to another peer that is decided by the Conference Manager. Even and odd frames are sent to different ports.

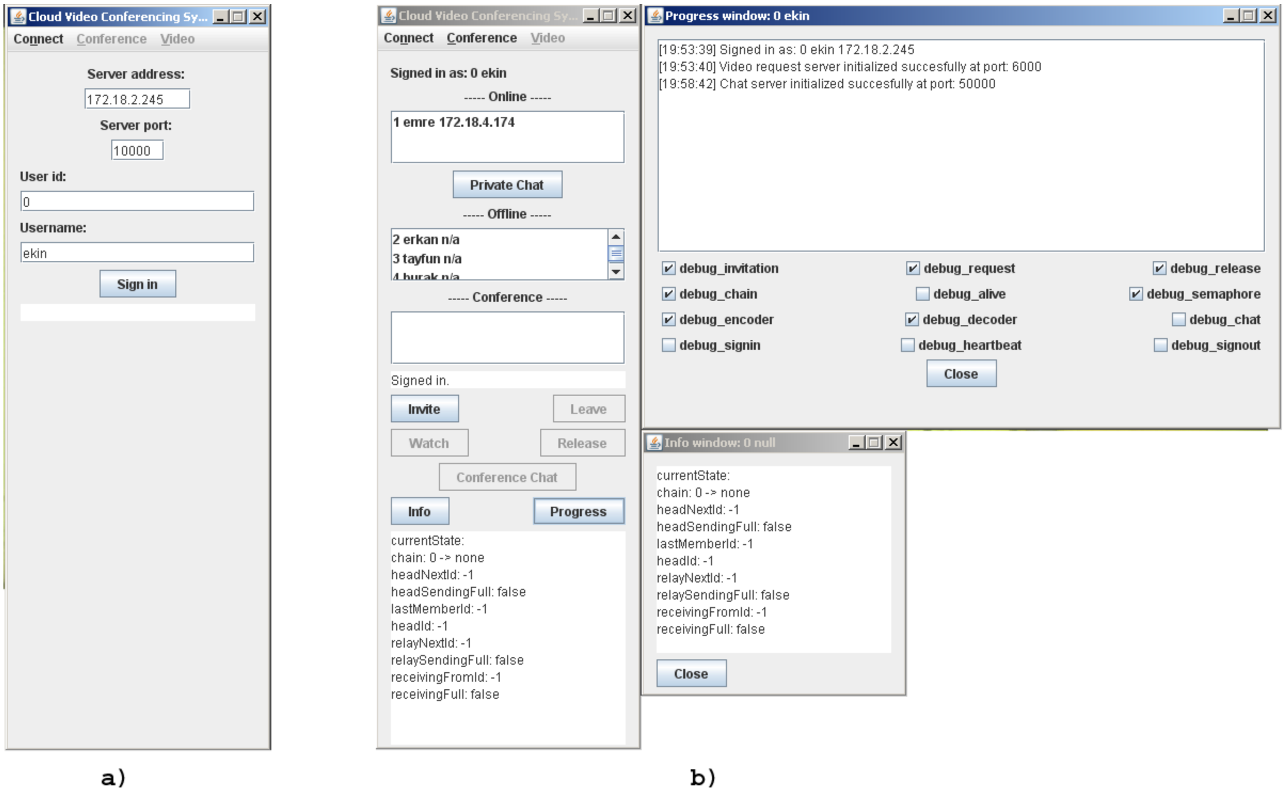
The **Decoder** process is responsible for receiving encoded packets, decoding them and sending them to the displayer process. Different ports are used for even and odd frames. The decoder process is also responsible for relaying operations. According to the information that comes from the Conference Manager module, the received packets are either forwarded to the next peer in the chain in full quality (i.e., both layers), base quality (i.e., only even-frames) or none at all.

The **Displayer** process is responsible for displaying the received frames in a synchronized fashion. The decoder feeds two different threads, one responsible for receiving of even frames and one of odd frames. Another thread accesses them from a shared buffer and displays them.

4.4 Peer Software Interfaces

4.4.1 Graphical User Interface (GUI)

GUI provides the interface for the user to interact with the conferencing software. These interactions are signing in, sending private or conference-wide text messages, inviting other online peers to a conference, accepting or rejecting invitations, making video requests to watch a participant or release a video source, leaving the conference and signing out of the system. It displays the details of the working of the peer software as it runs and the peer's current status in the conference such as the length of its chain, its members, the id of the peer it is watching if any and its relay information. The details of GUI are illustrated in Figures 4.2.



Server address: 172.18.2.245

Server port: 10000

User id: 0

Username: ekin

Signed in as: 0 ekin

----- Online -----

1 emre 172.18.4.174

----- Offline -----

2 erkan n/a
3 tayfun n/a
4 burak n/a

----- Conference -----

Signed in.

```
currentState:
chain: 0 -> none
headNextId: -1
headSendingFull: false
lastMemberId: -1
headId: -1
relayNextId: -1
relaySendingFull: false
receivingFromId: -1
receivingFull: false
```

```
[19:53:39] Signed in as: 0 ekin 172.18.2.245
[19:53:40] Video request server initialized succesfully at port: 6000
[19:58:42] Chat server initialized succesfully at port: 50000
```

☑ debug_invitation ☑ debug_request ☑ debug_release
☑ debug_chain □ debug_alive ☑ debug_semaphore
☑ debug_encoder ☑ debug_decoder □ debug_chat
□ debug_signin □ debug_heartbeat □ debug_signout

```
currentState:
chain: 0 -> none
headNextId: -1
headSendingFull: false
lastMemberId: -1
headId: -1
relayNextId: -1
relaySendingFull: false
receivingFromId: -1
receivingFull: false
```

Figure 4.2: GUI a) before the user signs in to the tracker. b) after the user signed in to the tracker.

4.4.2 Conference Manager - Video Interface

This interface is between the Conference Manager and Video related processes. Besides creating and destroying video related processes, the Conference Manager uses this interface to update the Encoder and the Decoder processes according to the current state of the peer.

The **Encoder** process is updated with the heading information, whether the process should start encoding frames received from the **Capturer** process and the video quality. It is also provided the IP address of the first member in the peer's chain.

```
UPDATE_HEADINGINFO <0 || 1> <0 || 1> <address>
```

The **Decoder** process is updated with the relay information. It includes whether the peer should be relaying, which quality of video should be relayed and the IP address of the peer that is next in the chain.

```
UPDATE_RELAYINFO <0 || 1> <0 || 1> <address>
```

4.5 Peer States

The actions that can be taken by the peers and responses to actions of other peers depend on the current status of the peer. Together with the state variables that can be viewed on Table 4.1, the current status of the peer is defined.

4.5.1 Participant

The peer in this state is in idle position. It does not watch the video signal of any participant, nor does it send its own video signal to other participants. The peer may send a video request to another participant to watch its video. When an online peer accepts an invitation from another peer, or when the invitation sent by this peer is accepted by the invited peer, in other words, when the peer joins a conference, it goes into this state.

4.5.2 Member

When a peer is watching another peer's video signal, it is in **Member** state. Whether it is *relaying* the video to another participant in the chain or not, is determined by the state variable **relayNextId**. A peer in this state cannot make a video request since it already made one. It may release the video source.

4.5.3 Chainhead

The peer in this state has a *chain*. It is sending its own video signal to the peer determined by the state variable **headNextId**. A peer in this state does not watch another peer's video signal, thus it can make a video request.

4.5.4 Chainhead_Member

When a peer is watching another peer’s video signal and sending its own video signal at the same time, it is in **Chainhead_Member** state. It is sending its own video signal to the peer determined by the state variable **headNextId**. Whether it is *relaying* or not, is determined by the state variable **relayNextId**. A peer in this state cannot make a video request since it already made one. It may release the video source.

Table 4.1: Peer State Variables

State Variable	**Explanation**	**Type**
headId	The id of the peer this peer is watching	int
relayNextId	The id of the peer this peer is relaying to; -1 if none	int
receivingFromId	The id of the peer this peer is being relayed from	int
headNextId	The id of the peer which is the first member in the chain	int
lastMemberId	The id of the peer which the last member in the chain	int
relaySendingFull	True if this peer is relaying with full quality; false otherwise	boolean
headSendingFull	True if this peer is sending its own video signal in full quality; false otherwise	boolean
receivingFull	True if this peer is receiving full quality video	boolean
isParticipant	True if this peer is in a conference; false otherwise	boolean
chain	The ordered list of the peers that are receiving this peer’s video	Vector
currentState	”” if isParticipant is false	String
	”Participant” if this peer is in a conference and watching nobody	String
	”Member” if this peer is watching another peer; but herself does not have chain	String
	”Chainhead” if this peer has a chain; but is not watching any other participant	String
	”Chainhead_Member” if this peer is watching another peer, and has a chain	String

4.6 Messages

There are two types of messages that are used in the prototype.

- messages exchanged between the peer and the tracker
- messages exchanged between the peers

4.6.1 Messages between peers and the tracker

The messages are depicted in Figure 4.3.

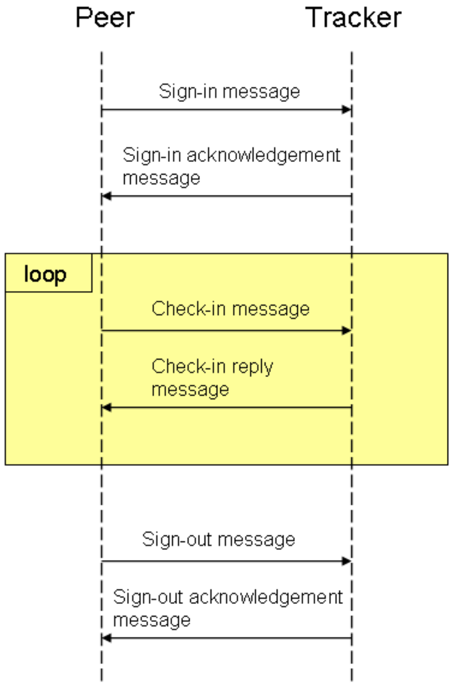


Figure 4.3: System Sequence Diagram of signing in, checking in and signing out events of a peer.

1. **Sign-in message:** This message is sent from a peer to the tracker to sign-in. The tracker updates the status of the sender from ”offline” to ”online”.

   ```
   SIGNIN <peer id> <peername> END
   ```

2. **Sign-in acknowledgement message:** This message is sent from the tracker to the peer as a response to the sign-in message.

   ```
   ACK_SIGNIN END
   ```

3. **Check-in message:** This message is periodically sent from the peer to the tracker, whenever the peer is online. The id’s of the peers in the contact list are

piggybacked at the end of the message. If a peer does not send this message (or the tracker does not receive this message) after a certain amount of time has passed, the tracker concludes that the peer experienced a crash and updates the status of this peer from ”online” to ”offline”.

```
LIST <peer id> <peername> CONTACTLIST [<peer1 id> <peer2 id> ...]
END
```

4. **Check-in reply message:** This message is sent as a response to the check-in message. It consists of the id’s of the peers in the corresponding check-in message, their availability status and their addresses.

```
ACK_LIST <peer1 id> <status1> <address1> <peer2 id> <status2>
<address2> ... END
```

5. **Sign-out message:** This message is sent from a peer to the tracker whenever the peer wants to sign-out of the system. The tracker updates the status of the sender from ”offline” to ”online”.

```
SIGNOUT <peer id> <peername> END
```

6. **Sign-out acknowledgement message:** This message is sent from the tracker to the peer as a response to the sign-out message.

```
ACK_SIGNOUT END
```

4.6.2 Messages between peers

All these messages are sent from an online peer to one or more online peers. They are depicted in Figures 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 4.10 and 4.11.

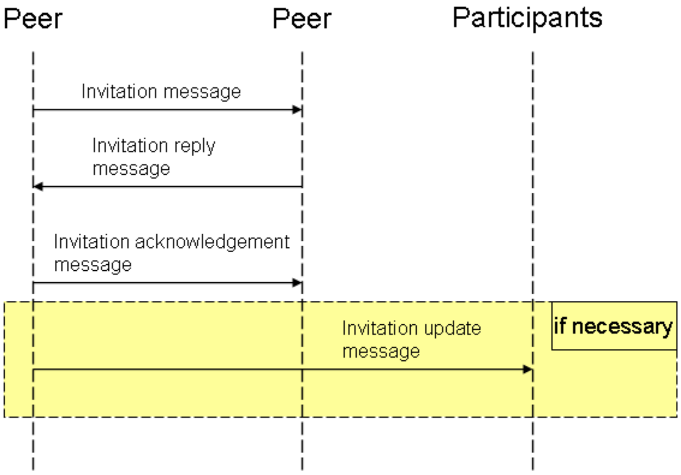


Figure 4.4: System Sequence Diagram of inviting an online peer to a conference.

1. **Invitation message:** This message is used to start a conference or to invite a peer to the current one. The receiving (and possibly replying) peer will be added to the conference if it accepts the invitation.

   ```
   INVITATION <peer id> <peername> <address> END
   ```

2. **Invitation reply message:** This message is a response to the invitation message. If the sender is already in a conference and accepts the invitation, it leaves the conference it is currently in and joins the new one. If it does not accept the invitation, nothing changes. In either case, this message is sent. The receiver of this message (i.e., the original sender of the invitation message) acts according to the reply. If the invitation is accepted, the peer is added to the conference. Other peers who are in the same conference are updated with the new participant's information. If the invitation is not accepted, the peer is notified with the reason. SORRY_0 means that the peer is in another conference. SORRY_1 means that the peer rejected the invitation.

   ```
   REPLY_INVITATION <peer id> <peername> <address> <OK || SORRY_0 ||
   SORRY_1> END
   ```

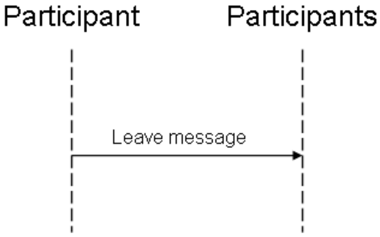


Figure 4.5: System Sequence Diagram of a peer leaving a conference.

3. **Invitation acknowledgement message:** This message is a response to the invitation reply message only when the reply message is positive (i.e., the replier has accepted the invitation). It is a list of the id's of the peers in the current conference, their peernames and their addresses.

   ```
   ACK_INVITATION <peer id> <peername> <address> <conference_peer1 id>
   [<conference_peername1> <address1> <conference_peer2 id>
   <conference_peername2> <address2> ...] END
   ```

4. **Invitation update message:** This message is sent from the inviter to the other peers in the conference to provide the contact information of another peer (i.e., the invited peer) who just joined the conference.

   ```
   UPDATE_INVITATION <peer id> <peername> <address> END
   ```

5. **Leave message:** This message is sent from an online peer to one or more online peers that are in the same conference as the sender whenever the sender wishes to leave the conference. If the peer is a head of a chain, its chain members update their status. If the peer is a member in a chain, it sends a video release message to the head first, so that it can update its chain accordingly.

   ```
   LEAVE <peer id> <peername> <address> END
   ```

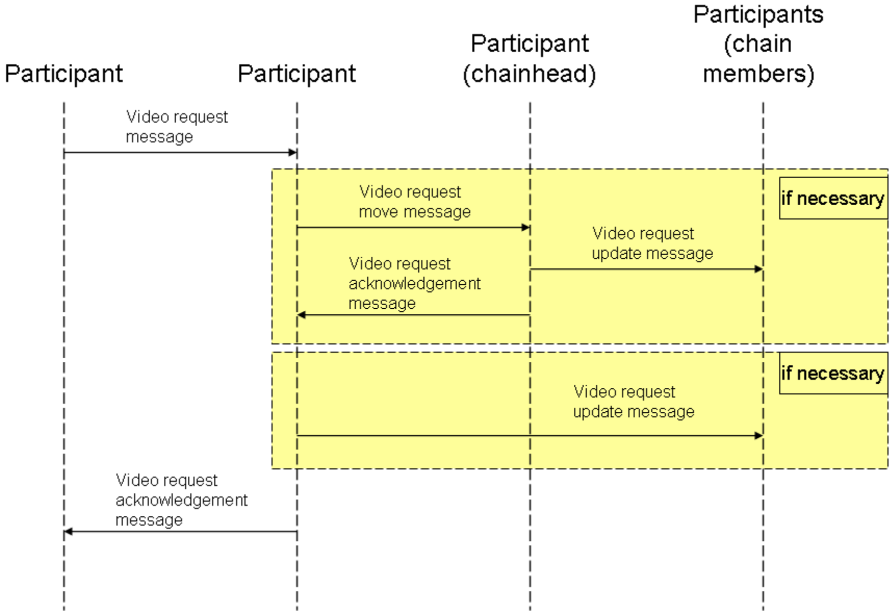

Participant
Participant
Participant (chainhead)
Participants (chain members)
Video request message
Video request move message
Video request acknowledgement message
Video request update message
if necessary
Video request update message
if necessary
Video request acknowledgement message

Figure 4.6: System Sequence Diagram of sending a video request, video request move, video request acknowledgement and video request update messages. Conference peers that are chain members are the members of the respective chains of the senders.

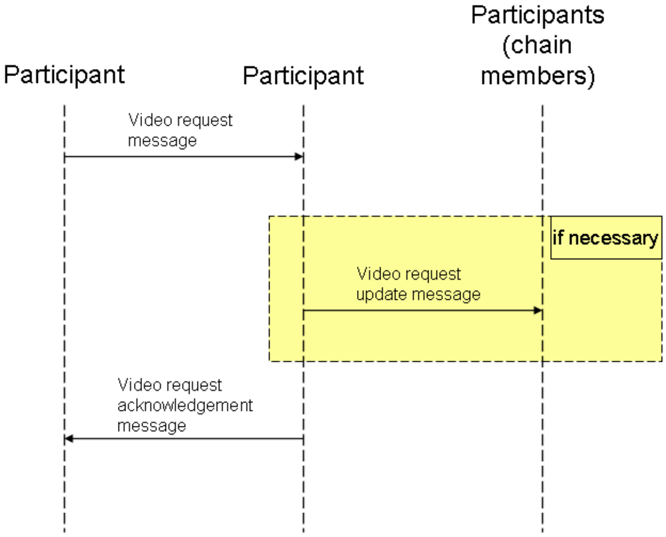

Participant
Participant
Participants (chain members)
Video request message
if necessary
Video request update message
Video request acknowledgement message

Figure 4.7: System Sequence Diagram of sending a video request, video request acknowledgement and video request update messages. Conference peers that are chain members are the members of the respective chains of the senders.

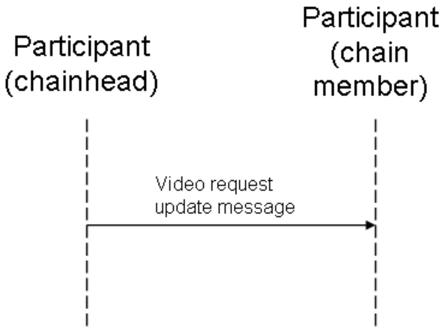


Figure 4.8: System Sequence Diagram of sending a video request update message.

6. **Video request message:** This message is sent from an online peer to another online peer that is in the same conference. The message is only sendable when the peer is not a member in a chain (i.e., have not requested to receive another participant's video). It has information about the status of the participant, whether it can forward any video in full quality or base quality and the length of its chain.

   A peer can forward a video signal in full quality, if and only if it is in **Participant** state. Otherwise, it at least needs to send one video signal in base quality. A peer can forward a video signal in base quality, when it is in **Participant** state; in **Chainhead** state, and sending in base quality; in **Member** state and does not forward the video (i.e., it is at the end of the chain) or forwarding in base quality; in **Chainhead_Member** state and does not forward the video signal it is receiving.

   ```
   VIDEO_REQUEST <peer id> <peername> <address> <Pass Through Full>
   <Pass Through Half> <chainlength> <current state> END
   ```

7. **Video request move message:** This message is sent from an online peer to another online peer when the sending peer is a member of the receiving peer's chain and has received a video request. The sending peer requests to be moved

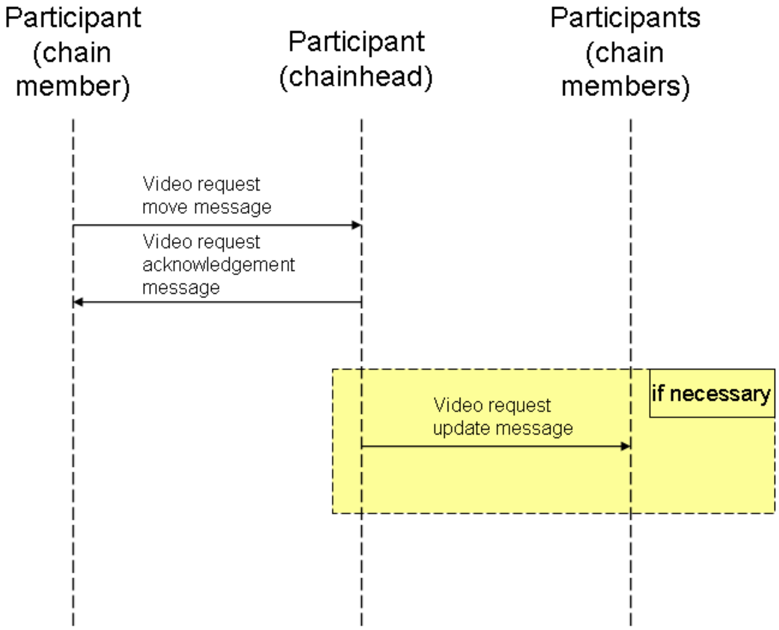

Participant (chain member)
Participant (chainhead)
Participants (chain members)
Video request move message
Video request acknowledgement message
Video request update message
if necessary

Figure 4.9: System Sequence Diagram of sending a video request move, video request acknowledgement and video request update messages. Conference peers that are chain members are the members of the respective chains of the senders.

to the end of the chain. The receiving peer tries to move the requesting peer to the end of the chain. It does whenever the length of the chain of the peer at the end of the chain is less than the requesting peer's. If not, the requesting peer is moved as close to the end as possible. This is done to minimize the number of base layer receivers.

```
VIDEO_REQUEST_MOVE <peer id> <peername> <address> <Pass Through
Full> <Pass Through Half> <chainlength> <current state> END
```

8. **Video request update message:** This message is sent from an online peer to another online peer when the sending peer is the head of the chain of which the receiving peer is a member. It contains information about the previous member, the next member and the video quality.

```
UPDATE_VIDEO_REQUEST <peer id> <peername> <address> <receive from
```

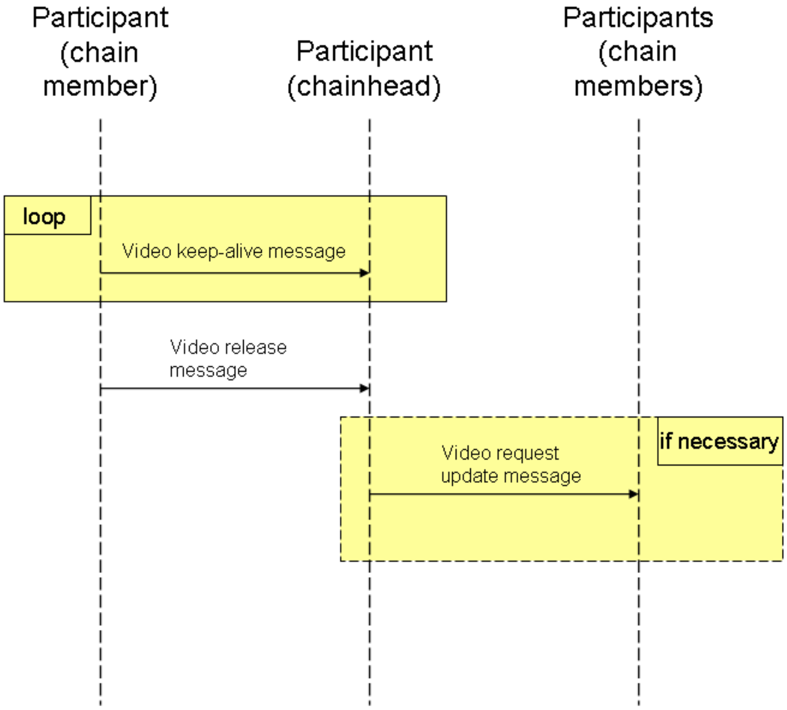


Figure 4.10: System Sequence Diagram of sending video keep-alive messages to the chainhead and releasing a video source of a peer.

```
peer id> <receive quality> <send next peer id> END
```

9. **Video request acknowledgement message:** This message is sent as a response to the video request message or the video request move message. After the head of the chain has decided where the requesting peer should be in the chain, it sends an acknowledgement to the requesting peer. This acknowledgement contains information about the previous member, the next member and the video quality.

```
ACK_VIDEO_REQUEST_<ADD || INSERT || MOVE> <peer id> <peername>
<address> <receive from peer id> <receive quality> <send next peer
id> END
```

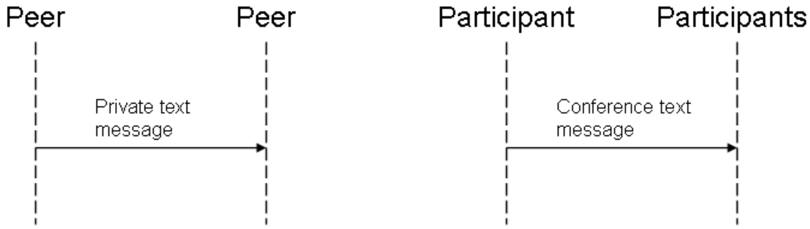


Figure 4.11: System Sequence Diagram of sending a private or conference message of a peer.

10. **Video keep-alive message:** This message is sent periodically from an online peer to another online peer when the sending peer is a member of the receiving peer's chain. This message helps the head of the chain to detect crashes of members and to update its status table consisting of the members. This way, the head can achieve fast decision whenever it receives a video request.

    ```
    VIDEO_ALIVE <peer id> <peername> <address> <current state>
    <chainlength> END
    ```

11. **Video release message:** This message is sent from an online peer to another online peer when the sending peer is a member of the receiving peer's chain and does not want to receive its video anymore.

    ```
    VIDEO_RELEASE <peer id> <peername> <address> END
    ```

12. **Chat message:** This message is sent from an online peer to another online peer if the message is private or to online peers in the conference, if the message is public.

    ```
    CHAT <peer id> <peername> <address> <private> MESSAGE <message> END
    ```

Chapter 5

MULTI-OBJECTIVE OPTIMIZATION

Optimizations to minimize the number of base layer receivers do not consider the delay experienced by the peers or the heterogeneity of their connection bandwidths as a parameter. In this chapter, we will define and derive formulations for the objective functions to

1. minimize the number of base layer receivers

2. minimize the maximum delay experienced in a chain

3. maximize the number of additional requests granted.

After explaining each objective separately, a multi-objective function is formulated to achieve these objectives simultaneously. It is then applied to example scenarios. Discussions on the formulation are presented.

5.1 Overview

Before continuing to explain each of the objectives separately, we first describe our assumptions. We give the general problem description and the solution we propose. We also present the computational complexity of our solution and show that it can be applied within our system.

5.1.1 Assumptions

Each peer is assumed to have *at least* a connection bandwidth that can support one full quality video (i.e., base *and* enhancement layers) to be received and sent at the same time. Some peers may have download bandwidths to enable them to receive more than one full quality video at the same time. These will be covered in the the multi-objective formulation. On the other hand, some peers may have upload bandwidths that can support more than one full quality video at the same time. In Section 3.3, we have shown that having multiple output bandwidths is always desirable and beneficial. Therefore, we do not consider these peers now to demonstrate that our approach works even when this kind of peers do not exist in a session.

Each chainhead obtains the delay information gathering round-trip-time (RTT) values (i.e., the time a message takes to go from one peer to another and back) from its members during the session. The one-way delays are calculated by dividing the RTT value to 2, since they are assumed to be symmetric, so that $d_{i,j}$, (i.e., the delay in the direction from peer i to peer j) is the same as $d_{j,i}$ (i.e., the delay in the direction from peer j to peer i). These values are stored in a table by the chainhead and updated periodically. A time synchronized algorithm [3] may be used for achieving one-way delays, however, the RTT information gathered seems to be sufficient without complicating the system.

The computational burden of the system is shown to be small [3]. Delays other than the end-to-end delays (e.g., processing, switching, forwarding) are assumed to be negligible compared to the network delays.

We assume that a participant can make at most two video requests, even if it has a bandwidth that is sufficient to view more than two video signals. The reason behind this assumption is that whenever a peer watches the video of another peer and interacts with it, the focus of the watching peer can be at most at two persons. Although this may look like a strong assumption, the situation is similar to whenever a group of people interact with each other in a face to face conference. It is argued that in a video conference with a *field of view* of 70 degrees *mostly* two persons are in a participant's view [32].

One can argue this situation might change, whenever the size of the group is increased. However, in such large groups, there is usually only one speaker (i.e., one video source) at a time whom the rest of the peers watch. In our case, there may be more than one video sources at the same time, most probably having different groups of receivers. Since our target in this system is relatively small groups, we believe that the assumption limiting the number of video requests to 2 is reasonable.

5.1.2 Problem Definition

Each chainhead that receives a video request needs to configure its chain accordingly to achieve the defined objectives. This is done using local information, and brings the advantage of not setting up a global information exchange mechanism and use it whenever a request is made. Omitting this overhead also allows the chainhead to decide fast, since the chain of a peer needs to be updated dynamically, whenever a video request is received by the chainhead.

One method to find the optimized solution for each of the objectives separately is to use combinatorial optimization and explore the entire solution space. However, we need to achieve *all* of them simultaneously. Therefore we need a combined objective. These objectives can be combined by choosing one objective to be achieved and setting constraints for the others (e.g., the number of base layer receivers should be less than c_b where c_b is an integer less than the number of participants in the session). However, this brings the disadvantage of deciding which objective to be achieved and how. Deciding the values of the constraints is another issue that needs to be overcome.

We claim that any comparison between the objectives to decide one is more important than the others cannot be justified. Instead, we combine these objectives by making use of the preferences of the peers. Since they are the ones who would get affected by the chain configuration that is going to be employed by the chainhead, we claim that this is reasonable and best thing to do.

In order to formulate the objectives, we define some variables.

- i**:** id of the peer
- c**:** a *possible* chain configuration

- l_i**:** the length of the chain headed by peer i (i.e., the cardinality of the set consisting of the peers that receive peer i's video signal)

Let $f_{v,c}(o)$ be an integer valued function of the positions of peers that return the id of a peer given its position in a *possible* chain configuration c headed by peer v.

Suppose that, there are three peers receiving the video of peer 1, namely peer 3, 4 and 5. One possible chain configuration of peer 1 is given by: <4, 3, 5>. In this particular chain configuration, $f_{1,<4,3,5>}(3) = 5$.

The number of possible chain configurations headed by peer v is given by the factorial of the number of the peers receiving the video of v, namely l_1, the chain length of peer 1. In this example, $l_1 = 3$, so the number of possible chain configurations is $3! = 6$. Since the chain head would always be in the 0'th order, it is omitted in the chain representation.

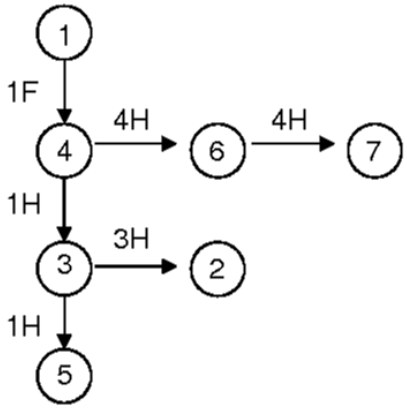


Figure 5.1: A possible chain configuration of peer 1 with the set of receivers 3, 4, 5. The chain is represented as <4, 3, 5>. Arrows indicate the direction of video transmission. F stands for full video quality and H stands for base layer quality. Peer 4 and peer 3 are also heads for the chains, <6, 7> and <2>, respectively.

5.1.3 Computational Complexity

The solution space consists of the possible chain configurations headed by a peer v. The number of these configurations is given by the factorial of the number of the peers receiving the video of v, namely the chainlength of peer v. Since we use combinatorial optimization and explore all the possible solution space, this gives a computational complexity of $O(n!)$ for the enumeration solution and would be problematic when the chain of a peer becomes very long (i.e., $l_v > 10$). However, our target group size for this application is small (i.e., participant count < 10). Since the chain configuration needs to be updated dynamically and in real-time, enumeration of possible chains is not as costly as applying special optimization formulations or running a special optimization software.

5.2 Optimization Objectives

Optimizations to minimize the number of base layer receivers are explained in the previous chapter. However, the delay experienced by the peers is not considered as a parameter. Furthermore, the work in [3] and [62] targeted end-points with low bandwidth connections that can support only one video signal to be sent and received. However, some peers may have higher bandwidth and asymmetric (i.e., download rate > upload rate) Internet connections.

5.2.1 Objective 1: Minimize the number of base layer receivers

By using layered video, the system allows each conference participant to see any other participant at any given time under *all* multipoint configurations of any number of users, with a caveat that some participants may have to receive only the base layer video. An objective of the system is to maximize the quality of the video each participant receives, that is, both the base and the enhancement layers of the video of a requested peer should be delivered, as long as this does not require the denial of another participant's request. Therefore, the system's objective is *to minimize the number of participants receiving only the base layer video in a given configuration.*

The following variable is defined to formulate the objective function.

$k_{v,c}$: the number of lower quality video receivers in a *possible* chain configuration c headed by peer v

Now, suppose that one or more of the receivers in this peer set {3, 4, 5} is also a chain head. For example, in the chain configuration <4, 3, 5>, let peer 3 and peer 4 be chain heads and $f^{-1}_{1,<4,3,5>}(3) = 2$ and $f^{-1}_{1,<4,3,5>}(4) = 1$, where $f^{-1}_{v,c}(j)$ is the inverse function $f_{v,c}(o)$. So, it gives the position of the peer with the id j in a *possible* chain configuration c. Let o be the smallest of these values, namely 1. Also, suppose that peer 3 has a chain length of 1 and peer 4 has a chain length of 2. This is illustrated in Figure 5.1.

Remember that peers 3 and 4 are chain heads and relays at the same time. This means that the peers in the chain of peer 3 (i.e., peer 2) and the peers in the chain of 4 (i.e., peer 6 and peer 7) receive only base layer as well as peer 3 and peer 5.

In this particular chain configuration of peer 1, the total number of base layer receivers is 2+1+1+1 = 5. (i.e., the chain length of peer 4 + peer 3 + the chain length of peer 3 + peer 5). Note that, if peer 5 was also a chain head, it would not relay, so its chain would receive full quality video and thus, would not be included in the sum. So $k_{v,c}$, the number of lower quality video receivers in a possible chain configuration c headed by peer v, is calculated as

$$k_{v,c}=\begin{cases} 0 \quad \text{if } o = l_v \textbf{ or } \textit{no such peer} \\ l_{f^{-1}_{v,c}(o)} + 1 + \sum\limits_{j=o+1}^{l_v-1} \left(l_{f^{-1}_{v,c}(j)} + 1 \right) \text{ otherwise} \end{cases}$$

The number of base layer receivers in a possible chain configuration c is 0, if $o = l_v$ (i.e., the corresponding peer is at the end of the chain) or there is no such peer that is head of a chain and relay, so that every peer receives full quality video. Otherwise, it is calculated as given above.

The first objective function $g_v(c)$ is defined as

$$g_v(c) = k_{v,c} \tag{5.1}$$

5.2.2 *Objective 2: Minimize the maximum delay experienced by a peer*

Since a video signal might have to be forwarded from a peer to another, this may cause the peers located towards the end of a chain experience large delays. Another objective of the system is *to minimize the delays experienced by the peers in a chain.* Since the maximum delay in a chain will be experienced by the peer at the end of the chain, we aim to minimize the delay of that peer and do our calculations based on that. We define the delay of a chain as the delay experienced by the peer at the end of that chain.

In order to calculate the delay of a chain, the end-to-end delay information between the peers are needed by the chain head. To do this, we re-define some of the messages described in [3]. Whenever a peer makes a video request, it piggybacks the RTT information between itself and the other peers. Thus the requested peer has information about the end-to-end delay between the requesting peer and other peers, so it can calculate the delay of a possible chain configuration.

Besides the request message, the keep-alive message is also re-defined. Keep-alive messages are sent by the members of a chain to the chain head periodically, so that the chain head can find out whether a peer has crashed or lost its connection and, then rearrange its chain accordingly. These messages are also used to keep the head informed of peers' current status and chain lengths (if they have any), so that the chain head can use this information to evaluate a request without needing to exchange any additional messages. Each member piggybacks the RTT information between itself and other peers in the conference to the keep-alive messages. This way, the chain head has all the information it needs to calculate the delay of a possible chain.

Since the maximum delay in a chain configuration is experienced by the peer at the end of the configuration c, the following holds.

$d_{v,c}$: delay experienced by the peer at the end of a *possible* chain configuration c headed by peer v

The chainhead that received a video request calculates the delay value of each possible chain configuration by adding the one-way delay values between the peers.

$$d_{v,c} = \sum_{j=1}^{l_v} d_{f_{v,c}(j-1), f_{v,c}(j)}$$

The second objective function $h_v(c)$ is defined as

$$h_v(c) = d_{v,c} \tag{5.2}$$

The possible chain configurations and their delay values for the example given in Figure 5.1 are given below.

$$\begin{aligned}
h_1(< 3, 4, 5 >) &= d_{1,3} + d_{3,4} + d_{4,5} \\
h_1(< 3, 5, 4 >) &= d_{1,3} + d_{3,5} + d_{5,4} \\
h_1(< 4, 3, 5 >) &= d_{1,4} + d_{4,3} + d_{3,5} \\
h_1(< 4, 5, 3 >) &= d_{1,4} + d_{4,5} + d_{5,3} \\
h_1(< 5, 3, 4 >) &= d_{1,5} + d_{5,3} + d_{3,4} \\
h_1(< 5, 4, 3 >) &= d_{1,5} + d_{5,4} + d_{4,3}
\end{aligned}$$

5.2.3 Objective 3: Maximize the number of additional requests granted

The P2P approach we presented in [3] and [62] assumed that the peers have a connection bandwidth that is enough to send and receive only one video signal. However, the diversity of the Internet connections and the asymmetry of these connections lead to the fact that peers may have spare bandwidth for sending and receiving more than one video signal.

Having an upload bandwidth that can support more than one video signal is always beneficial. Suppose a peer R receives a video signal (i.e., watches another peer H) and forwards it to another peer E in a chain. Whenever another peer A requests the video of peer R, peer R may need to drop the quality of the forwarded video to base layer, in order to be able to send its own video with the remaining bandwidth (also in base layer). If peer R has an upload bandwidth to support more than one video signal at the same time, it does not need to drop the quality of the forwarded video, instead it uses its spare bandwidth to grant the request. So, peer E and peer A would both receive full quality video (i.e., base + enhancement layers), instead of only the base layer. This applies to all similar situations whenever a peer needs to be a chain head and relay at the same time. Therefore, we do not investigate the cases where peers have spare bandwidths for upload, rather we concentrate on peers having spare bandwidth to receive more than one video signal (i.e., watching more than one peer).

Considering these, another objective of the system is *to maximize the number of such additional requests that are granted.* Thus, we formulate the objective so that the value of the objective function is 1 if the additional request is granted, whereas it is -1 if the request is declined. Every chain head investigates each possible chain configuration c whenever a peer makes an additional video request to the corresponding chain head.

$$s_{v,c} = \begin{cases} \text{-1} & \text{if the request is not granted} \\ 1 & \text{if the request is granted} \end{cases}$$

The third and last objective function $m_v(c)$ is defined as

$$m_v(c) = s_{v,c} \tag{5.3}$$

5.3 Multi-objective Optimization

In this section, we will describe our multi-objective optimization approach. We will first present the mechanism to assign importance weights to each objective using the preferences of the peers. Then, we will demonstrate how the technique is applied to the system through example scenarios.

5.3.1 Formulation

In order to determine the best solution, we use the weighted sum method [63]. The issue of determining importance weights to be assigned to each objective is overcome by employing a preference mechanism. Peers being aware of the optimization objectives choose one of the objectives as their preference. These will be exchanged during the initialization of the conference. Peers may change their preferences during the conference, but they need to inform the others. The assigned importance weights of each objective function f_i is defined as

$$w_{f_i} = \frac{p_{f_i}}{n} \tag{5.4}$$

where p_{f_i} represents the number of peers that prefer f_i to the other optimization objectives. n is the number of participants in the conference.

The importance weights are determined using the number of participants in the *entire* conference (i.e., n), rather than the number of participants in a corresponding chain. The reason is that a peer's preference (e.g., a chain head) may affect other peers' (e.g., the peers in its chain) received video quality. Consider the conference in Figure 5.2 with 5 participants. Suppose that peers 2 and 3 prefer minimum delay and peers 1, 4 and 5 prefer maximum video quality. Peers 1 and 2 are geographically closer, so the chain configuration in Figure 5.2a will be employed by the chain head (i.e., peer 1) to minimize the delay. This will cause the other chain head (i.e., peer 2) send its own video signal in base layer quality, although *all* peers in its chain prefer maximum video quality. Since they can receive only the base layer, their preferences will have no effect, even if they constitute the majority in the conference. Therefore, rather than using only the preferences in the corresponding chain, all peers' preferences are taken into account while determining the importance weights. As a consequence, the configuration in Figure 5.2b should be used to satisfy the majority.

Each chain head v would calculate the scaled versions of the optimization functions while determining which chain configuration they are going to employ whenever they receive a video request message. The formula for that is

$$f_{i,v,scaled}(c) = w_{f_i} \frac{f_{i,v}(c) - f_{i,v,min}(c)}{f_{i,v,max}(c) + f_{i,v,min}(c)} \tag{5.5}$$

where $f_{i,v,min}$ and $f_{i,v,max}$ represent the minimum and the maximum value of that optimization function, respectively. The combined objective function would be $u_v(c)$ as given below.

$$u_v(c) = min(\sum_{m} f_{i,v,scaled}) \tag{5.6}$$

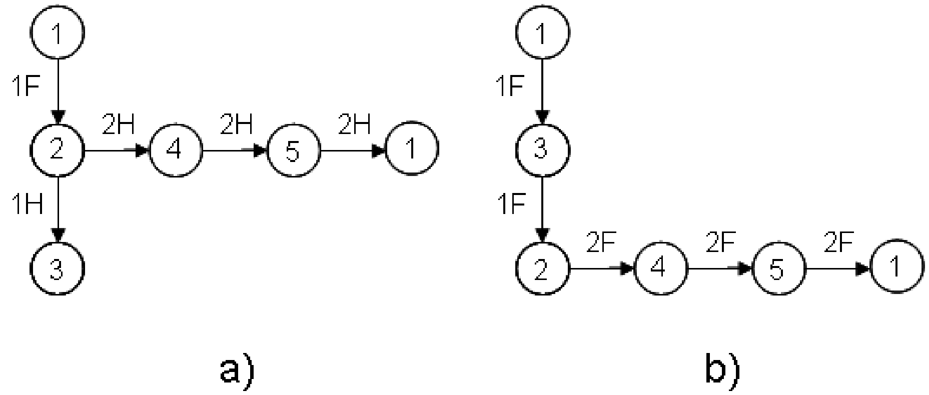


Figure 5.2: a) If the importance weights are assigned only with respect to the preferences of the peers in a chain. b) If the importance weights are assigned according to the preferences of all peers. The majority of the peers get what they want: maximum video quality.

where m is the number of optimization objectives that are used in the multi-objective solution. The chain head v would employ c^*, the configuration optimizing the objective function.

5.3.2 *Example scenario: Minimize the number of base layer receivers and the maximum delay in a chain*

In this part, we will illustrate the multi-objective optimization technique with an example scenario. For simplicity and ease of understanding, we assume that all peers have sufficient connection bandwidth only for one full-quality video. Thus, we do not take the additional requests into account; they will be further investigated in the second example.

Each chain head v should try *to minimize the number of base layer receivers in its chain* and *minimize the maximum delay experienced in that chain*, at the same time. First, we will show that these optimization objectives may be conflicting with each other.

Suppose there is a conference session with 7 participants. Peers 1 and 4 are located in the USA, peers 3 and 5 are located in Turkey, peers 6 and 7 are located in Germany and peer 2 is located in Canada. Typical one-way delay values between the peers are given in Table 5.1.

Let the video request configuration of this conference be the following: Peers 3, 4 and 5 request to view peer 1's video, peer 2 requests peer 3's video and peers 6 and 7 request peer 4's video. So peers 1, 3 and 4 have chains with lengths 3, 1 and 2, respectively.

Suppose that peer 1 needs to configure its chain, so that the number of base layer receivers and the maximum delay experienced by a peer are minimized. The minimum number of base layer receivers is achieved by the chain configuration <5, 3, 4> (i.e., peer 4 has the longest chain length, and thus, it should be at the end of the chain, giving a total of 2 base layer receivers). The minimized maximum delay is achieved by the chain configuration <4, 3, 5> and yields a maximum delay of 129 ms as calculated from Table 5.1. These possible configurations can be seen in Figure 5.3.

Table 5.1: Latency Table

peer id	1	2	3	4	5	6	7
1	0	26	**89**	**24**	**95**	66	70
2	26	0	104	78	108	73	75
3	**89**	104	0	**85**	**20**	55	42
4	**24**	78	**85**	0	**98**	65	71
5	**95**	108	**20**	**98**	0	53	47
6	66	73	55	65	53	0	12
7	70	75	42	71	47	12	0

$$g_1(<4,3,5>) = 5 \text{ and } h_1(<4,3,5>) = \mathbf{129\ ms}$$

$$g_1(<5,3,4>) = \mathbf{2} \text{ and } h_1(<5,3,4>) = 200 \text{ ms}$$

If the chain head (i.e., peer 1) *were* to minimize *only* the number of base layer receivers, then the chain configuration it should be employing would be <5, 3, 4>.

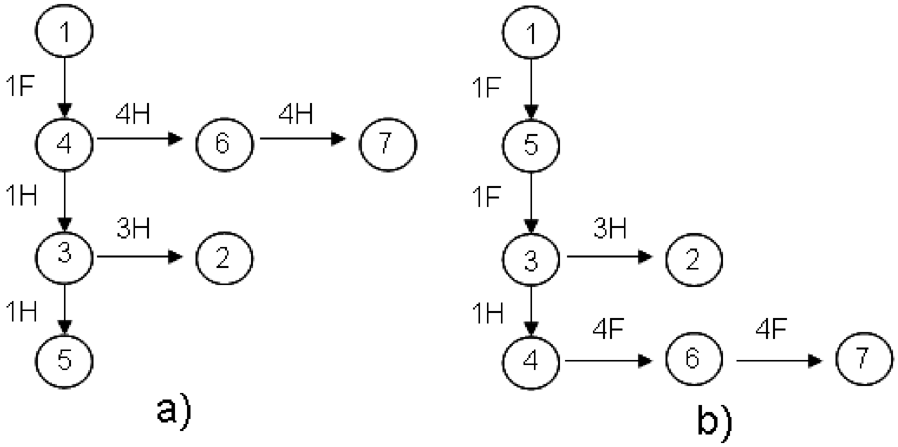


Figure 5.3: a) Chain configuration <4, 3, 5> yielding 5 as the number of base layer receivers (Peers 6, 7, 3, 2 and 5). b) Chain configuration <3, 5, 4> yielding 2 as the number of base layer receivers (Peers 2 and 4).

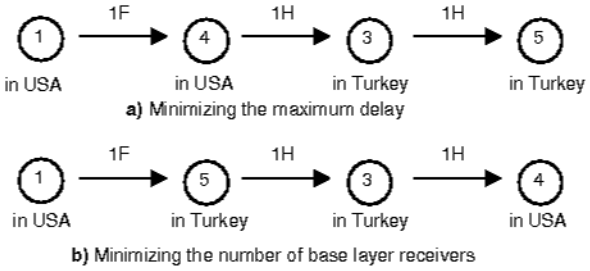


Figure 5.4: a) Correct chain order to minimize the maximum delay. b) Correct chain to minimize the number of base layer receivers.

On the other hand, if it *were* to minimize *only* the maximum delay in a chain, then the chain configuration <4, 3, 5> should be employed. Clearly, these two objectives conflict with each other. Figure 5.4 shows details of this example.

In our scenario, peers 5 and 6 prefer maximum video quality and peers 1, 2, 3, 4 and 7 prefer minimum delay. So, $p_g = 2$ and $p_h = 5$. Then the weights would be $w_g = 0.29$ and $w_h = 0.71$. $g_v(c)$ and $h_v(c)$ values are scaled according to the equation (5.5). The best solution is determined according to the optimization function $u_v(c)$. The entire set of the possible chain configurations and their $g_v(c)$ and $h_v(c)$ values are given in Table 5.2.

According to Table 5.2, the chain configuration <4, 5, 3> gives the minimum value of the objective function. This chain is employed by the chain head. Remember that, 4 of 6 peers had told that they would prefer minimum delay over high quality of video. We can see the trade-off between minimizing the delay and minimizing the number of base layer receivers. Peer 1 employs a chain that has the *second* best minimized delay and *third best* minimized number of base layer receivers.

Table 5.3 gives $g_v(c)$ and $h_v(c)$ values calculated with the preference values $w_{pref,g} = 0.71$ and $w_{pref,h} = 0.29$, so that now 2 of 7 peers prefer high video quality over minimum delay.

Since the preference values have changed, peer 1 has to change its chain to the best configuration determined by the objective function. The chain configuration <5, 3, 4> has the minimum $g_v(c)$ value and has the *third best* delay. The compromise is done to match the preferences of the peers.

5.3.3 Example scenario: Minimize the number of base layer receivers and maximize the number of additional requests granted

In this example, we assume that the peers are geographically close to each other and thus, do not take the delays into account. The peers prefer either maximum video quality or additional video requests.

Table 5.2: $g_1(c)$ and $h_1(c)$ values with $w_g = 0.29$ and $w_h = 0.71$.

c	$g_1(c)$	$h_1(c)$	$g_{1,scaled}(c)$	$h_{1,scaled}(c)$	$u_1(c)$
<3, 4, 5>	5	272	0.29	0.68	0.97
<3, 5, 4>	3	207	0.10	0.37	0.47
<4, 3, 5>	5	129	0.29	0.00	0.29
<4, 5, 3>	**4**	**142**	**0.19**	**0.06**	**0.25**
<5, 3, 4>	2	200	0.00	0.34	0.34
<5, 4, 3>	3	278	0.10	0.71	0.81

Table 5.3: $g_1(c)$ and $h_1(c)$ values with $w_g = 0.71$ and $w_h = 0.29$.

c	$g_1(c)$	$h_1(c)$	$g_{1,scaled}(c)$	$h_{1,scaled}(c)$	$u_1(c)$
<3, 4, 5>	5	272	0.71	0.28	0.99
<3, 5, 4>	3	207	0.24	0.15	0.39
<4, 3, 5>	5	129	0.71	0.00	0.71
<4, 5, 3>	4	142	0.47	0.03	0.50
<5, 3, 4>	**2**	**200**	**0.00**	**0.14**	**0.14**
<5, 4, 3>	3	278	0.24	0.29	0.53

The conference has 6 participants and is illustrated in Figure 5.5. The chains of peer 1, peer 3 and peer 5 consist of peers 2 and 3, peers 4 and 5, and peer 6, respectively. Suppose that peer 5 has sufficient bandwidth to make an additional video request and that it requests peer 1's video. If peer 1 would arrange its chain just to minimize the number of base layer receivers, peer 5's additional video request would be rejected. Figure 5.5a depicts this situation. If the request were granted employing a chain configuration like in Figure 5.5b, that would mean that 3 peers would receive base layer video (i.e., 3 requests would be granted, but the requesters would receive base layer video; requesters 4, 5 *and* 5). However, the configuration shown in Figure 5.5c would allow that a smaller number of requests would receive base layer video (i.e., only 2; peers 2 and 6).

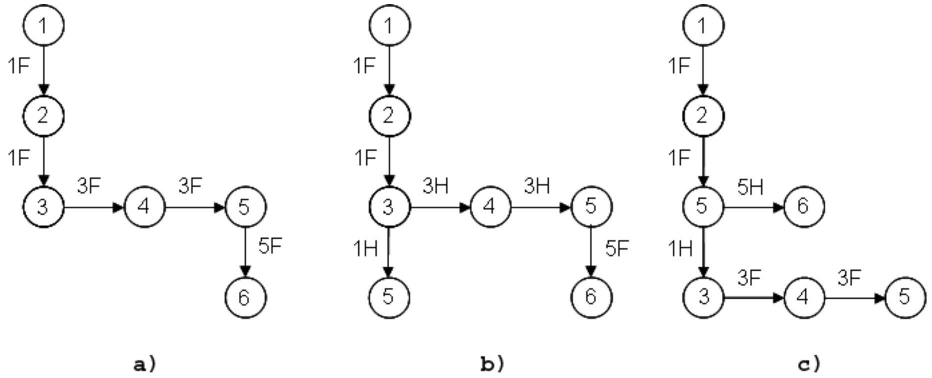


Figure 5.5: a) The configuration after peer 5's additional request is rejected. b) A possible chain configuration of peer 1 after it granted peer 5's additional request. c) Another possible chain configuration of peer 1 with only 2 base layer receivers.

As can be seen, the two objectives conflict again, as one requires that the request is rejected and the other requires that some peers receive base layer video. Assuming that 4 out of 6 peers would prefer that additional requests are granted over the video quality, the importance weights for the objective functions would be: $p_g = 2/6 = 0.33$ and $p_m = 4/6 = 0.67$.

The $u_v(c)$ values for all chain configurations are presented in Table 5.4. Since the second objective aims to *maximize* the number of additional requests granted, its value is negated to *minimize* the overall objective function $u_v(c)$. According to the Table 5.4, the chain configuration <2, 5, 3> is the best-compromise solution and thus, would be employed by the chain head (i.e., peer 1). Since 4 out of 6 peers prefer that additional requests are granted, the system has tried to maximize that number as well as to minimize the number of base layer receivers at the same time.

Table 5.5 shows how the objective function values change as the importance weights change when only one peer would prefer that additional requests are granted. Since the chain head would calculate the objective function values according to the new weights and employ the best-compromise chain configuration, the second request of peer 5 would be rejected.

Table 5.4: $g_1(c)$ and $m_1(c)$ values with $w_g = 0.33$ and $w_m = 0.67$.

c	$g_1(c)$	$m_1(c)$	$g_{1,scaled}(c)$	$m_{1,scaled}(c)$	$u_1(c)$
<2, 3, 5>	3	1	0.20	0.67	-0.47
<2, 5, 3>	**2**	**1**	**0.13**	**0.67**	**-0.53**
<5, 2, 3>	3	1	0.20	0.67	-0.47
<5, 3, 2>	5	1	0.33	0.67	-0.33
<3, 2, 5>	4	1	0.27	0.67	-0.40
<3, 5, 2>	5	1	0.33	0.67	-0.33
<2, 3>	0	-1	0.00	0.00	0.00

Table 5.5: $g_1(c)$ and $m_1(c)$ values with $w_g = 0.83$ and $w_m = 0.17$.

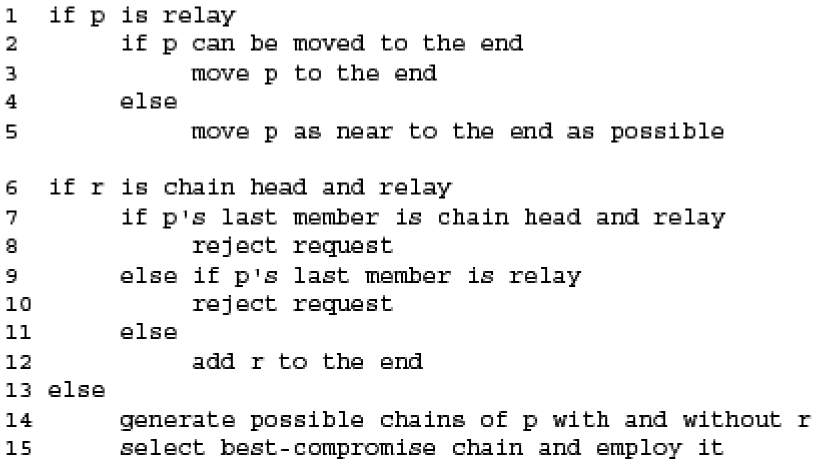
c	$g_1(c)$	$m_1(c)$	$g_{1,scaled}(c)$	$m_{1,scaled}(c)$	$u_1(c)$
<2, 3, 5>	3	1	0.50	0.17	0.33
<2, 5, 3>	2	1	0.33	0.17	0.16
<5, 2, 3>	3	1	0.50	0.17	0.33
<5, 3, 2>	5	1	0.83	0.17	0.66
<3, 2, 5>	4	1	0.67	0.17	0.50
<3, 5, 2>	5	1	0.83	0.17	0.66
<2, 3>	**0**	**-1**	**0.00**	**0.00**	**0.00**

5.4 Simulation results

In this section, we present simulation results covering possible scenarios with up to 10 participants. Each participant requests the video of another participant randomly. Participants with additional bandwidth make another video request in addition to their first requests. We again do not take the delays into account.

For each conference case with different participant counts, we generated 100,000 cases randomly, in which the number of participants with additional bandwidths is increased from 1 up to the participant count for that case. For example, for a conference scenario with 6 participants, we generated 100,000 random cases with only one participant having additional bandwidth; we generated another 100,000 with two of the participants having additional bandwidth and so on.

During our simulations, we assumed that the participants with no additional bandwidth prefer maximum video quality and the rest prefers that additional requests are granted. This is plausible, because a participant with additional bandwidth would be more likely willing that additional requests are granted. The importance weights for maximizing the video quality and maximizing the number of granted additional requests are calculated accordingly.



```
1  if p is relay
2       if p can be moved to the end
3            move p to the end
4       else
5            move p as near to the end as possible

6  if r is chain head and relay
7       if p's last member is chain head and relay
8            reject request
9       else if p's last member is relay
10           reject request
11      else
12           add r to the end
13 else
14      generate possible chains of p with and without r
15      select best-compromise chain and employ it
```

Figure 5.6: Pseudo code to handle a request.

Whenever a request comes to a participant p from a participant r, the actions that p can take are given in the pseudo code (Figure 5.6). In the first part (lines 1-5), the status of the requested participant p is checked. If it is relaying video, the chain head of p tries to move p to the end of the chain, so that it would not forward the video anymore. This way, the number of base layer receivers in the chain is minimized [62]. If the last member of the chain has a shorter chain or no chain, the head of p can move p to the end. If not, then p is moved to a position where the number of base layer receivers is minimized according to our optimization formulations described in [62] (i.e., near the end of the chain).

The second part deals with the request. If the requesting participant r is a chain head and a relay at the same time, the status of the last member in p's chain needs to be checked (lines 6-12). We are going to explain each case in detail.

The last member l might have made two requests, for p's and o's video signals. Also, at least one other participant might have requested l's video, so l is a chain head. Suppose that o's chain requires that l must relay video. Since l is also a chain head, it will relay only base layer of o's video. The remaining bandwidth will be used for l's own video. The only way that l receives p's video is that it is at the end of p's chain. Once another participant r, a chain head and a relay itself, requests p's video, the only position r could be was at the end, but since l cannot be in another position in p's chain and relay p's video to r, the request is rejected.

Similarly, l might have not been a chain head, but only a relay for o. Suppose that another participant m did not make a request yet and we allowed that r relay two different video signals, so that l also relays p's video to r. When m makes its first request to receive l's video, this would be rejected, because l's uploading bandwidth is used for relaying two base layer video signals (i.e., p's and o's). However, our condition that each participant can see any other participant anytime contradicts with this. So, we do not allow that a participant relay two different video signals. Therefore, r's request is rejected. Remember that r had additional bandwidth and the request was its second. r has already a request granted, so our condition holds.

The rest of the code is for when r is not a chain head and a relay at the same time. This means, that it could also be relaying in p's chain, if the best-compromise chain requires it to do so. Participant p generates all possible chain configurations with r in its requester list. The best-compromise chain is selected by the head p using the calculations given in Section 5.2 regarding the importance levels. According to the best-compromise chain, the request is either granted or rejected.

The results of the scenarios for each participant count are averaged. Figure 5.7 shows the percentages of the rejected requests, granted base layer video receiving requests and granted full quality receiving requests. The percentage of the rejected requests does not exceed 15% and decreases as the number of participant count increases. Although the percentage of the base layer video receiving requests increases with the participant count, this increase is asymptotic. Our system was able to grant at least 50% of the requests to receive full quality video.

Even when additional requests were not granted, there was a slight increase in the base layer receiving requests with the increase in the participant count. However, the ratio of the average number of base layer receiving requests to the total number of requests is decreasing as the participant count increases [62]. For the multi-objective case, simulations show that the increase in the participant count, increases the number of base layer video receiving requests as well. But this increase is also asymptotic and does not damage the system's scalability.

Instead of rejecting the requests, the system makes use of layered video and grants the additional requests according to the best-compromise chain found. Increased participant count increases the probability for a request to receive base quality layer video; however, in cases where base layer video is used, the average percentage of these requests to all requests stays below 45% percent and is generally about 39% (Figure 5.8).

In some cases, there may be rejected requests; however, all of these requests are additional requests, so that the requesters already receive a participant's video. Our system tries to maximize the number of granted additional requests, as long as this does not cause other participants' requests to be rejected.

5.5 Discussions

The best-compromise solution makes a trade-off between the $g_v(c)$ and $h_v(c)$ or $m_v(c)$ values. Each change in the number of base layer receivers may cause a change in the other objective value. The maximum delay experienced by the peers in the corresponding chain may increase or decrease; an additional request of a peer may be rejected or granted. However, one cannot make a clear statement saying that 'a change in the number of base layer receivers corresponds to a certain increase or decrease in the maximum delay of that chain' or 'a change in the number of base layer receivers corresponds to a certain number of additional requests to be granted or rejected'. The

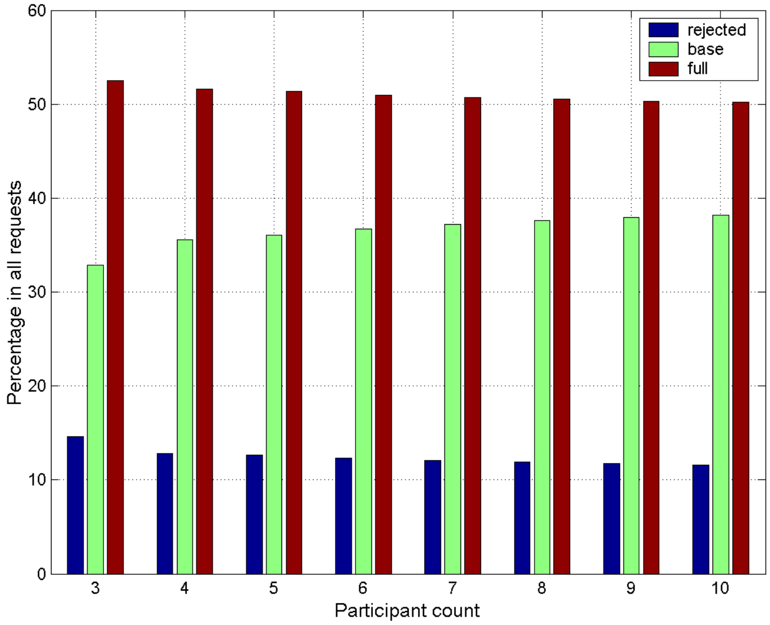


Figure 5.7: Percentages of all requests.

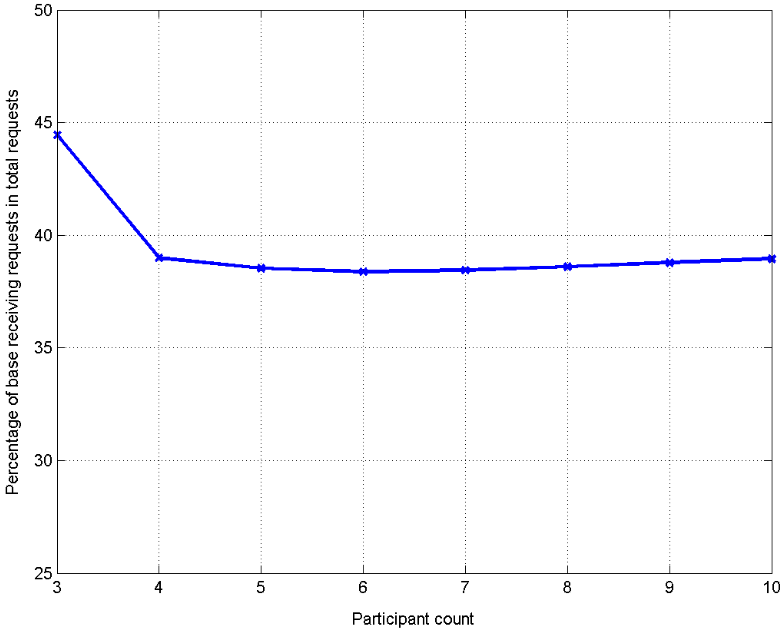


Figure 5.8: Average percentage of base layer receiving requests to total requests in cases where base layer is used.

uniqueness of the video request set (i.e., which peers request the video of which other peers), the pair-wise delay values and which peers have sufficient bandwidth to make additional requests prevent this.

Since we cannot know how much delay is worth how many base layer receivers or how many full quality video receivers we can sacrifice to grant one more additional request, we need a way to determine the sensitivity of this trade-off. Therefore, we need to assign importance weights to the objective functions. This assignment is a hard task which is overcome by the preferences of the peers. There is no way to compare the number of base layer receivers with the delay values or additional requests to conclude one is more important than the other. The preferences of the peers are used to determine the importance weights of the optimization objectives. This is a plausible assumption since the peers will be the ones affected by the employed chain configuration.

The system makes use of layered video to guarantee that each video request at each participant is granted. With the presented extensions, the system makes compromises between the quality of the video peers receive, delays experienced by peers and additional requests of peers with sufficient bandwidth. Our multi-objective formulation successfully finds the best-compromise solution. We have shown with a counter-example that the solution should be calculated according to the preferences of *all* participants in the conference, not just in a corresponding chain.

Instead of denying an additional request when the combined objective function does not allow it to be granted, one may queue the requesting peer r. This way, this peer will have higher priority than the new requester n, when the chainhead receives a new video request and tries to update its chain. However, this may cause the problem of having the peer n wait in queue a long time, when the combined objective function never allows the request of r to be granted.

The sensitivity analysis showed that our objective function reacts to the changes in the preferences of the participants and satisfies them in the best way. Although the employed solution causes more requests to receive base layer video quality, simulation results show that this does not damage the system's scalability. Since the intended group size is relatively small, generation of possible chain configurations does not bring much computational burden. This makes the system easy to implement.

Chapter 6

FORMAL SPECIFICATION, VERIFICATION AND SECURITY ASPECTS

In this chapter, we discuss other aspects of the presented P2P MP video conferencing system. Formal specification and verification process of our protocol will be explained. Besides, a scheme that provides security to the relayed video packets will be described and analyzed.

6.1 Applying Formal Methods

Although simulations have been done to validate that the given protocol is functioning properly and results have been given, a formal model and verification of the developed protocol is still needed. Many tools and methods exist for formal modeling and verification of distributed systems. Among them, TLA+ is the best to be used with our system. TLA+ is a formal specification language enabling to formally model systems and their properties [5]. TLA+ specifications are checked using TLC model checker [6]. In this section, the formal specifications, the iterations in the process and the steps taken to formally specify the system are explained. Also, some analytical results and discussions are given.

The model is specified assuming that the users have already joined the conference and can see other participants so that they can make video requests to them. It does not cover a special service which takes care of showing who is online and who is not, joining or leaving a conference and inviting other users to a conference.

The most important restriction of the system is that the users can only issue a request to watch just one user's video signal. Before they make another, they have to first release the one they are already receiving and then issue another request.

Although this release operation was implemented and specified using TLA+, it drastically increased the search space. Therefore, this operation is left without employment in the specification.

6.1.1 Formal Specification

In this section, we will describe the states, variables and messages that are used in the TLA+ specification. We will also provide their correspondents in the prototype implementation. However, some of these are used only in TLA+ specification in order to cover atomicity of our distributed system.

We can explain the atomicity principle applied to our system by defining *negotiation*. A negotiation is the event of sending a message and waiting for a reply or receiving a message and generating a reply. There are three types of negotiations:

- Type 1: Sending a video request and receiving an acknowledgement
- Type 2: Receiving a video request, handling it and sending an acknowledgement
- Type 3: Receiving a video request, sending a ”move” request, receiving a ”moveReply” message and sending an acknowledgement

Receiving a ”move” request and generating a reply is not considered as a negotiation, since this request can be handled as a regular video request without causing any deadlocks.

Negotiations are considered as atomic operations, so that a user cannot issue more than one video request to different users at the same time and a user cannot process more than one request for different users at the same time. This achieves that a user in a negotiation cannot interact with another user in the same way and ensures that a user is only in *one* negotiation phase. However, to cover the atomicity principle in the specification of our distributed system, five steps are needed at most.

1. Send a video request to a user and enter a negotiation state (type 1)

2. Receive the video request from *receiveQueue* to *requestQueue*, so that the receiving user enters a negotiation state (type 2)

3. Handle the video request. If required, send a ”move” request and enter a negotiation (type 3)

4. Receive the reply from the *moveReplyQueue* and continue to handle the video request state (type 1)

5. Send an acknowledgement to the requester and exit negotiation state (type 3).

Although the atomicity principle is not violated during these steps, it does not prevent the system to interleave the steps when different users are in interaction. This interleaving increases possible states of the system. To a user, it may look like it is making one atomic operation when it is in negotiation with another user, but to the system, this happens in five steps. As the number of the users increases, the probability that these steps interleave is increased and thus, the state space is increased as well.

Table 6.1: User states in the TLA+ specification with a correspondent in the prototype implementation.

State name in TLA+	**State name in implementation**	**Brief explanation**
Participant	Participant	Initial state; does not send or receive any video signal.
Relay	Member	Receives a video signal; forwards it to the next user.
Chainhead	Chainhead	Sends its own video signal; does not receive any video signal.
Chainhead_Relay	Chainhead_Member	Receives and forwards a video signal, and sends its own video signal.
Waiting	-	Negotiation state (type 1)
Processing	-	Negotiation state (type 2)
Processing_II	-	Negotiation state (type 3)

Table 6.2: Variables in the TLA+ specification with a correspondent in the prototype implementation.

Variable name in TLA+	**Variable name in implementation**	**Explanation**
sendingToAsRelay	relayNextId	The id of the user that is next in the chain.
nextMember	headNextId	The id of the user that is the first member in this user's chain.
lastMember	lastMemberId	The id of the user that is the last member in this user's chain.
headId	headId	The id of the user that this user is watching.
receivingFrom	receivingFromId	The id of the user that this user is being relayed from.
sendingChainFull	headSendingFull	True if this user is sending its video signal in full quality.
sendingNextFull	relaySendingFull	True if this user is relaying in full quality.
currentState	currentState	”Participant” in both, ”Relay” in TLA+; ”Member” in implementation, ”Chainhead” in both, ”Chainhead_Relay” in TLA+; ”Chainhead_Member” in implementation
chain	chain	The ordered list of the group that is watching this user's video signal.
chainlength	chain.size()	The cardinality of the group that is watching this user's video signal.
prevState	-	Indicates the state of the user before it went into one of the negotiation states.
myCurrentStatusTable	-	Each user's own state variables define its status.
otherUsersStatusTable	chain	**Chain** vector elements are used to store the status of chain members.
requestQueue receiveQueue ackQueue updateQueue infoQueue moveReplyQueue	- - - - - -	All messages are handled in a thread-per-request fashion. The prioritization of the threads is achieved by using synchronization primitives such as semaphores and locks. Each sent message causes the entire system to be in a different state as a whole. The model checking process handles the differences between different interleavings of the steps.
canPassThroughFull	canPassThroughFull	Temporary variable; indicates whether the user can relay full quality video.
canPassThroughHalf	canPassThroughHalf	Temporary variable; indicates whether the user can relay base quality video.
newNext	sendNext	Temporary variable; indicates the next member's id in the ”update” message.
newFrom	recvFrom	Temporary variable; indicates the member's id before this user in the ”update” message.

Table 6.3: Messages in the TLA+ specification with a correspondent in the prototype implementation.

Message id	**Message name in TLA+**	**Message name in implementation**	**Explanation**
1	request	Video request	Video request message.
2	ack	Video request acknowledgement	Acknowledgement message to inform the requester.
3	info	Video keep-alive	Information message to update the head.
4	move	Video request move	Move request message sent to the head.
5	update	Video request update	Update message to the chain members with the new relay information.
6 7 8 9	moveOK moveNOTOK moveNOTOKSwap moveNOTOKSwap2	- - - -	These are the move reply message types in the TLA+. In the prototype, video request update messages are used instead of these. The handling thread *wait*s until the participant receives a video update message from the head and a *notify* event occurs.

Peer States in TLA+ specification

According to the specification, a user can be in one of the states given in Table 6.1 at any time of the model-checking process. We will describe the states used only in the TLA+ specification in detail.

Waiting: This is the state when a user in **Participant** or **Chainhead** state issued a request to another user and waits for the result of that transaction. The user will be in **Relay** or **Chainhead_Relay** state after the reception of the result, respectively. This is one of the negotiation states (type 1) defined to cover atomicity in the TLA+ specification. There is not a correspondent state in the prototype implementation; rather a semaphore permit is acquired so that no other thread can execute at the same time. When the "video request acknowledgement" arrives, the semaphore permit is released. This way, it is ensured that the user interacts only with one user.

Processing: This is the state when a user in **Participant**, **Relay**, **Chainhead** or **Chainhead_Relay** states received a video request from another participant and processes the request according to the decision algorithm described above. If the user is in one of the **Participant** or **Chainhead** states, the result of the algorithm would be sent to the issuer of the request. An "update" message may be sent to affected chain members if a change in the chain configuration is needed. If the user is in one of the **Relay** or **Chainhead_Relay** states, a "move" message is sent to the head and this user goes into **Processing_II** state to wait for the result. This is one of the negotiation states (type 2) defined to cover atomicity in the TLA+ specification. There is not a correspondent state in the prototype implementation; rather a semaphore permit is acquired so that no other thread can execute at the same time. The request is granted according to the decision algorithm. When the "video request acknowledgement" message is sent, the semaphore permit is released. This way, it is ensured that the user interacts only with one user.

Processing_II: This is the state when a user in **Relay** or **Chainhead_Relay** states received a video request from another participant and needs to send a "move" request to its chain head. According to the reply from the head, an appropriate acknowledgement would be sent to the issuer of the video request and possibly to another relay if an update in the chain configuration is needed. This is one of the negotiation states (type 3) defined to cover atomicity in the TLA+ specification. It is a continuation of the state **Processing**. There is not a correspondent state in the prototype implementation; rather a lock is used to synchronize the threads. The handling thread *wait*s a "video update". When it is received, the handling thread *notifies* the *wait*ing thread. Since the semaphore permit was acquired as this user was in **Processing** state and is not released yet, there can be only one thread executing. After the arrival of the "video update" message, the "video request" is handled and the "video request acknowledgement" message is sent. The semaphore permit is released after this.

State variables

Besides the state information given above, some state variables are also needed to be used in the decision algorithm. These are given in Table 6.2. Variables without a correspondent in the prototype implementation are described in detail. The states **Waiting**, **Processing** and **Processing_II** are not considered when describing the variables, since these are negotiation states. The variables are kept also in these states, but the users cannot stay in these states forever.

prevState: It indicates the state of the user before going to one of the negotiation states. This information is used to determine in which state the user would be after the decision algorithm ends. It can be any state except the negotiation states, because the atomicity principle prevents that a user is in negotiation with more than one user at the same time. Therefore, a user needs to go first to one of the four states other than these negotiation states. If the user needs to go to the negotiation states, the *prevState* variable is updated accordingly.

myCurrentStatusTable: Each user is a separate entity. The most simple way to keep their states and their state variables is to use this table. It includes the *currentState*, *chainlength*, *sendingToAsRelay*, *nextMember*, *lastMember*, *receivingFrom*, *headId*, *sendingChainFull*, *sendingNextFull* and *prevState* information. In the implementation, each peer software keeps its information.

otherUsersStatusTable: This table is used to keep the information about users. Whenever ”info” messages arrive, the information in this table is updated and later used in the decision algorithm. Although it keeps all the users' information, the specification ensures that a user running the decision algorithm only accesses the information about its members if it has any.

The message exchanges are handled using queues. Any message sent to a user is appended to the corresponding queue and the user then picks the message from the queue and acts accordingly. These are explained below.

receiveQueue: The receiveQueue is used to send request messages to this user. These messages can either be ”video request” or ”move” messages. The user then picks the message and changes its state to **Processing**.

requestQueue: The requestQueue is an internal queue. The peer receives messages from *requestQueue* and append them to this queue. It allows to distinguish the current state of the user and the actions according to it.

updateQueue: The updateQueue is used for messages related to updates. This only can be used if the owner is a member of a chain, so that the head of that chain (i.e., the user specified by the headId of this user) makes some updates on who is sending to whom and receiving from whom according to the requests it has received.

ackQueue: The answer to any of the video request messages are sent to this queue. The user picks the message, and changes its state from **Waiting** to some state according to its *prevState* variable (e.g., if the user were in **Participant** state, it goes into **Relay**; if the user were in **Chainhead** state, then into **Chainhead_Relay** state).

infoQueue: A user who is in one of the **Relay** or **Chainhead_Relay** states may send an ”info” message to its head in order to keep its updated. These messages are sent into this queue. A user who is in **Chainhead** or **Chainhead_Relay** state has an *infoQueue.*

moveReplyQueue: This queue is used for receiving a reply for a ”move” request.

A user who is in **Processing_II** state can check whether any reply is present. If not, it stays until there is one. The decision algorithm proceeds according to the type of the reply.

One may wonder why more than one queue is used for the message exchange. This is done so because the atomic operations would need message exchanges so that any message triggered in the negotiation phase going to the regular queue would be a problem. All the requests would also go the regular queue, so a user in negotiation would not be allowed to receive any messages from the queue until the negotiation is over (according to the atomicity principle), but the negotiation would not be over until some negotiation messages are exchanged which are in the queue behind the requests. This surely causes a deadlock, in which the regular request waits for the negotiation to be over and the negotiation waits the messages to be handled so that the negotiation message is next to be handled. On the other hand, different queues for the negotiation messages allow the negotiation to finish so that the regular message can be handled later. Therefore, besides the regular queue, *receiveQueue*, also an *ackQueue* and a *moveReplyQueue* are used for negotiation messages.

Besides, an *updateQueue* is implemented as well, in order to handle update messages whenever a request is processed and a result for the decision algorithm is obtained. This way, any update changing a state of the user would not affect the negotiation with another user (i.e., a user might have requested another user's video with the state information at time *t*, but at time *t'* that state information might have changed so that the decision algorithm gets affected in the sense that it uses the state information of the requester at time *t*, but at the time the decision is made the state information has been changed, violating the atomicity principle).

With different queues, a priority scheme is needed. Since the atomicity principle is not to be violated, updates are needed to be handled first. Secondly, only one negotiation is allowed, so that before beginning another one, the previous one needs to be finished. Therefore, the acknowledgements have a higher priority than regular requests. To handle the requests, a user moves them first to the *requestQueue* from the *receiveQueue*, so in order to begin another negotiation the request in the *requestQueue* needs to be processed first. The order is *updateQueue*, *moveReplyQueue*, *ackQueue*, *requestQueue*, *receiveQueue*.

Until the higher priority queues are empty, the messages are kept waiting in the queues. The usage of different queues for different type of messages ensures that the system is deadlock-free along with the priority scheme.

Messages

In order to make transitions for a particular user as described above, some input and output is required. In a distributed P2P application, they are messages exchanged between the peers. These are explained in Table 6.3.

Table 6.4: Messages that cause transitions (input), messages that are sent during the transition (output) and state variables that have an effect on the transition.

State transition id	**Message id (Input)**	**Message id (Output)**	**State variables**
1	-	1	prevState = ”Participant”
2	-	1	prevState = ”Chainhead”
3	2	-	prevState = ”Participant”
4	2	-	prevState = ”Chainhead”
5	1	-	prevState = ”Participant”
6	1	-	prevState = ”Chainhead”
7	-	2	prevState = ”Participant” or ”Chainhead”
8	1	-	prevState = ”Relay”
9	3	-	-
10	4	6 or 7 or 8 or 9	-
11	1	-	prevState = ”Chainhead_Relay”
12	5	-	-
13	3	-	-
14	4	6 or 7 or 8 or 9	-
15	5	-	-
16	-	4	prevState = ”Relay” or ”Chainhead_Relay”
17	-	6 or 7 or 8 or 9	prevState = ”Relay” or ”Chainhead_Relay”

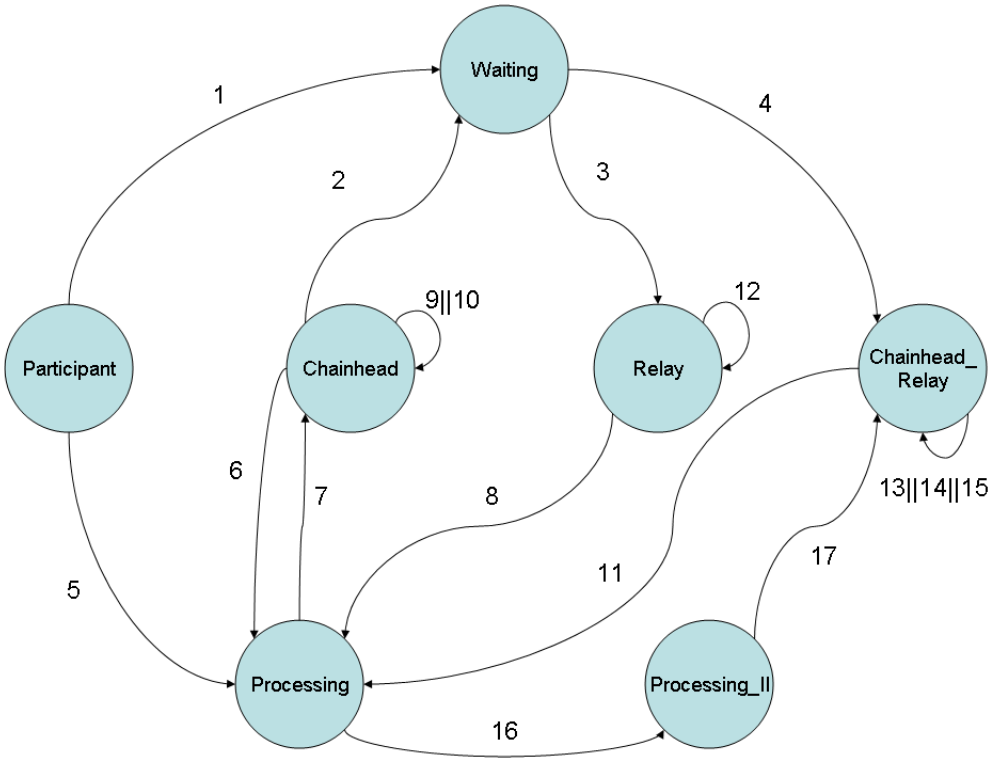


Figure 6.1: Finite State Machine. The numbers on the transition arrows indicate transition id. The messages causing this transition (input) and sent during the transition (output) are given in Table 6.4.

6.1.2 Illustrating the Abstraction Map

In this part, the abstraction map between the TLA+ specification and the prototype implementation will be illustrated with two crucial operations.

Sending a video request message

Video requests can be sent only by those users who are in either **Participant** state or in **Chainhead** state. A participant that sends a video request message goes into **Waiting** state in the TLA+ specification. Until the corresponding ”ack” message arrives, the user stays in that state. During this time, it cannot send another video request to another participant or receive and handle a video request from another participant. This way the atomicity of ”sending a video request message” is preserved, since the sender of the video request message does not interact with another user during the negotiation.

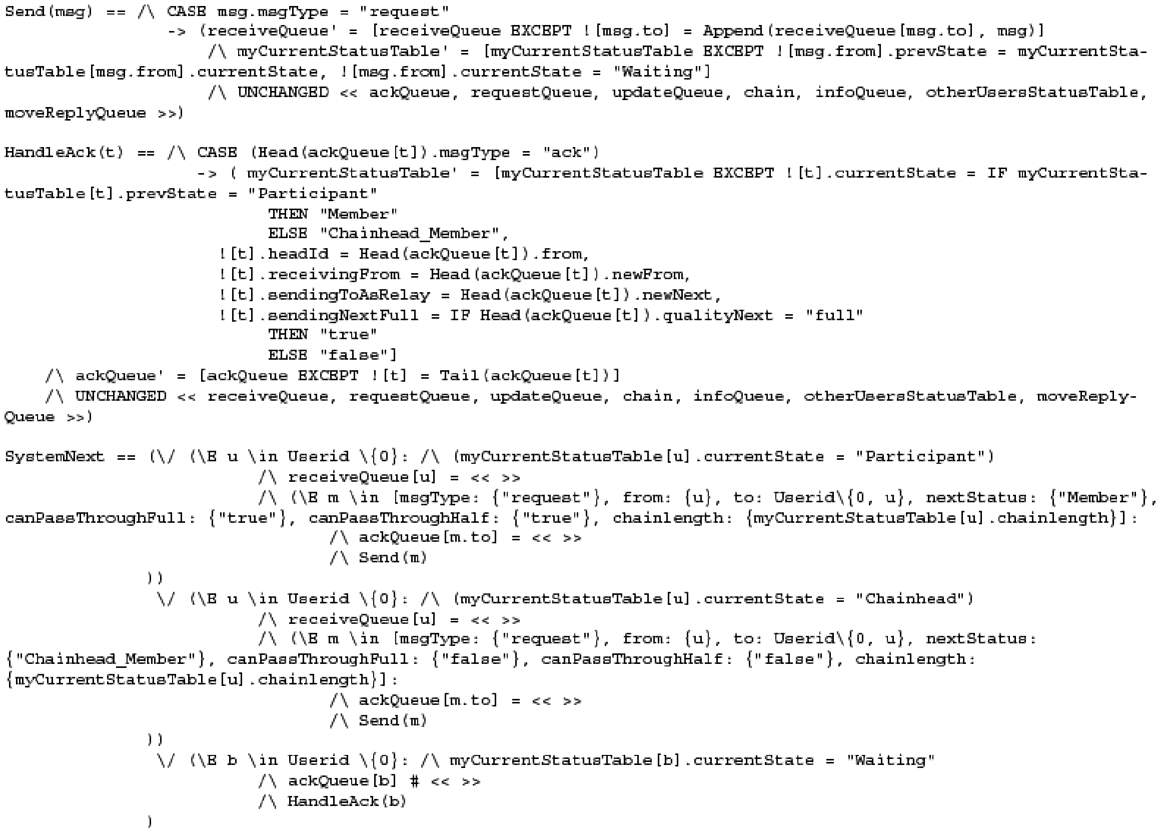
Likewise, a participant acquires a semaphore *permit* when it is sending a video request in the prototype. The semaphore *permit* is not released until the corresponding "video request acknowledgement message" arrives. All video request messages that are received by this user and the threads that are created to handle them, are queued until the lock is released. The user cannot make a second video request until the first one is resulted; this is handled by the GUI by blocking the user to make another request. One can view both, TLA+ specification and prototype implementation in Figures 6.2 and 6.3, respectively.

Handling a video request message

A participant in any state other than **Waiting**, **Processing** and **Processing_II**, may receive a request message. In other words, participants who are not in negotiation, may receive a request from their *receiveQueue* to their *requestQueue*. Its *prevState* variable is updated to have the value of the *currentState* variable, before the state change. As soon as it does this, the participant goes into **Processing** state. Since it is in a negotiation now, it cannot send and receive video request at this time.

If the participant's *prevState* is equal to **Participant** or **Chainhead**, the participant handles the request in its *requestQueue* and sends back the "ack" message according to the decision algorithm. If the participant's *prevState* is equal to **Relay** or **Chainhead_Relay**, the participant goes into **Processing_II** state and sends a "move request" to the head of its chain. It stays until a "move reply" message is received into the *moveReplyQueue*.

After receiving the "move reply" message, the participant handles the request in its *requestQueue* and sends back the "ack" message according to the decision algorithm. A participant receiving a "move request" message, handles this as a regular "request" message (i.e., updates its *prevState* variable, goes to "Processing_I" state, handles the "request", sends back the "move reply" message).

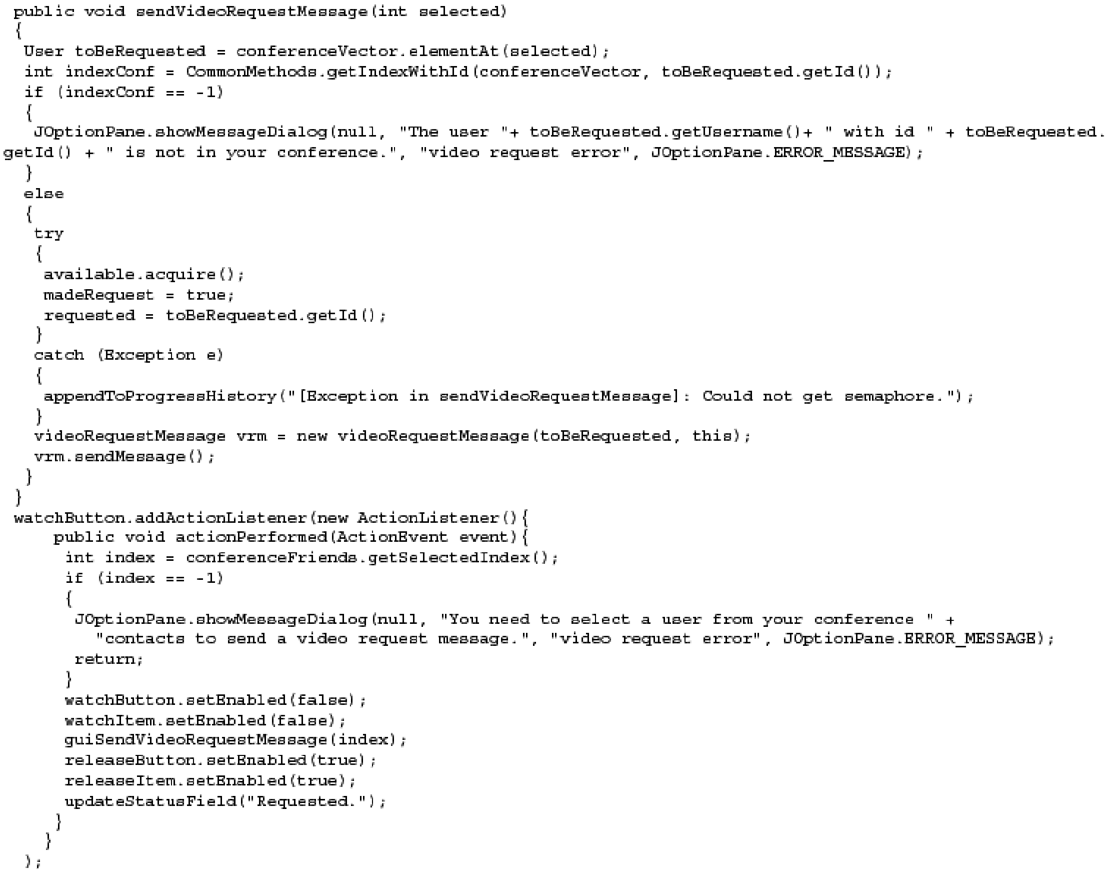


```
Send(msg) == /\ CASE msg.msgType = "request"
                -> (receiveQueue' = [receiveQueue EXCEPT ![msg.to] = Append(receiveQueue[msg.to], msg)]
                    /\ myCurrentStatusTable' = [myCurrentStatusTable EXCEPT ![msg.from].prevState = myCurrentSta-
tusTable[msg.from].currentState, ![msg.from].currentState = "Waiting"]
                    /\ UNCHANGED << ackQueue, requestQueue, updateQueue, chain, infoQueue, otherUsersStatusTable,
moveReplyQueue >>)

HandleAck(t) == /\ CASE (Head(ackQueue[t]).msgType = "ack")
                   -> ( myCurrentStatusTable' = [myCurrentStatusTable EXCEPT ![t].currentState = IF myCurrentSta-
tusTable[t].prevState = "Participant"
                          THEN "Member"
                          ELSE "Chainhead_Member",
                     ![t].headId = Head(ackQueue[t]).from,
                     ![t].receivingFrom = Head(ackQueue[t]).newFrom,
                     ![t].sendingToAsRelay = Head(ackQueue[t]).newNext,
                     ![t].sendingNextFull = IF Head(ackQueue[t]).qualityNext = "full"
                          THEN "true"
                          ELSE "false"]
    /\ ackQueue' = [ackQueue EXCEPT ![t] = Tail(ackQueue[t])]
    /\ UNCHANGED << receiveQueue, requestQueue, updateQueue, chain, infoQueue, otherUsersStatusTable, moveReply-
Queue >>)

SystemNext == (\/ (\E u \in Userid \{0}: /\ (myCurrentStatusTable[u].currentState = "Participant")
                         /\ receiveQueue[u] = << >>
                         /\ (\E m \in [msgType: {"request"}, from: {u}, to: Userid\{0, u}, nextStatus: {"Member"},
canPassThroughFull: {"true"}, canPassThroughHalf: {"true"}, chainlength: {myCurrentStatusTable[u].chainlength}]:
                                /\ ackQueue[m.to] = << >>
                                /\ Send(m)
              ))
               \/ (\E u \in Userid \{0}: /\ (myCurrentStatusTable[u].currentState = "Chainhead")
                         /\ receiveQueue[u] = << >>
                         /\ (\E m \in [msgType: {"request"}, from: {u}, to: Userid\{0, u}, nextStatus:
{"Chainhead_Member"}, canPassThroughFull: {"false"}, canPassThroughHalf: {"false"}, chainlength:
{myCurrentStatusTable[u].chainlength}]:
                                /\ ackQueue[m.to] = << >>
                                /\ Send(m)
              ))
               \/ (\E b \in Userid \{0}: /\ myCurrentStatusTable[b].currentState = "Waiting"
                         /\ ackQueue[b] # << >>
                         /\ HandleAck(b)
              )
```

Figure 6.2: TLA+ specification of sending a video request.

Similarly, a participant may receive a ”video request” message in the prototype. The handling thread is created, but blocked until a *semaphore permit* is acquired. Since the semaphore allows only one thread to have the permit to execute, no other threads can execute at the same time. If the *currentState* is **Participant** or **Chainhead**, the video request message is handled and a video request acknowledgement message is sent back. The permit is released after this event. If the participant’s *currentState* is equal to **Member** and **Chainhead_Member**, sends the move request to its respective head and the thread waits a *lock* to be notified. The notification arrives as soon as the participant receives a ”video move reply” message.

After the reception of the ”video move reply”, the participant continues to handle the video request according to the decision algorithm. It sends a ”video request



```
 public void sendVideoRequestMessage(int selected)
 {
  User toBeRequested = conferenceVector.elementAt(selected);
  int indexConf = CommonMethods.getIndexWithId(conferenceVector, toBeRequested.getId());
  if (indexConf == -1)
  {
   JOptionPane.showMessageDialog(null, "The user "+ toBeRequested.getUsername()+ " with id " + toBeRequested.
getId() + " is not in your conference.", "video request error", JOptionPane.ERROR_MESSAGE);
  }
  else
  {
   try
   {
    available.acquire();
    madeRequest = true;
    requested = toBeRequested.getId();
   }
   catch (Exception e)
   {
    appendToProgressHistory("[Exception in sendVideoRequestMessage]: Could not get semaphore.");
   }
   videoRequestMessage vrm = new videoRequestMessage(toBeRequested, this);
   vrm.sendMessage();
  }
 }
 watchButton.addActionListener(new ActionListener(){
     public void actionPerformed(ActionEvent event){
      int index = conferenceFriends.getSelectedIndex();
      if (index == -1)
      {
       JOptionPane.showMessageDialog(null, "You need to select a user from your conference " +
         "contacts to send a video request message.", "video request error", JOptionPane.ERROR_MESSAGE);
       return;
      }
      watchButton.setEnabled(false);
      watchItem.setEnabled(false);
      guiSendVideoRequestMessage(index);
      releaseButton.setEnabled(true);
      releaseItem.setEnabled(true);
      updateStatusField("Requested.");
     }
    }
  );
```

Figure 6.3: Prototype implementation of sending a video request.

acknowledgement" message to the requester.

6.1.3 Formal Verification through Model Checking

The specifications written in TLA+ are verified using TLC model checker. The verification makes sure that the specified system works deadlock-free. It also checks the peer states and state variables. The type invariant of the system consists of these variables and their allowed values. The model checking process uses the type invariant and the next state action of the system and tests whether the results of the next state action contradicts with the type invariant and causes any problems in the system's functioning.

The next state action of the system satisfies one of the following conditions. The actions that are taken when one of these is true are indicated in ***bold and italic***. There may be cases when more than one of these is true. Then TLC handles everything in order and checks every possible case. The entire specification can be found in Appendix CD.

- There can be a user in **Participant** or **Chainhead** state that may ***make a request***.

- There can be a user not in **Processing**, **Processing_II** and **Waiting** state whose *updateQueue* is empty, but *receiveQueue* is not and there is no other request waiting in the *requestQueue*, then the user may ***receive*** the request from the *receiveQueue* to the *requestQueue*.

- There can be a user in **Processing** state and therefore its *requestQueue* is not empty, then the user may ***receive*** the request from the *requestQueue* and ***process*** it according to the decision algorithm given.

- There can be a user in **Processing_II** state and therefore its *requestQueue* is not empty and its *moveReplyQueue* is not empty, so that it can ***receive the reply*** and ***handle the request*** according to the decision algorithm given.

- There can be a user whose *updateQueue* is not empty, then it can ***receive the update*** and ***change its variables accordingly***.

- There can be a user in **Waiting** state and its *ackQueue* is not empty, then the user can ***receive the "ack" message*** and ***change its state and update its variables accordingly***.

- Nothing in the state and variables of the users may change.

As shown in Table 6.5, the number of states that the system can be in increases as the number of users increases. This is related to the atomicity principle in our distributed system described and explained in Section 6.1.1.

Table 6.5: State information of TLC checks

# of users	# of states	# of distinct states	Depth of state graph
2	19	9	9
3	291	116	14
4	4341	1469	22
5	63326	18454	31
6	356736	236232	40

6.1.4 Discussions

The specification is verified using TLC model checker. It is shown that the system works deadlock-free. There are no violations of the type invariant, so peers take legal actions defined by the next-state action to go from one legal state to another.

Whenever a deadlock would occur, TLC would give the state graph starting from the initial predicate. An example of this can be found in Appendix CD. The reason for the problem in the appendix is that the system is not specified to allow that nothing in the states and state variables of the peers changes. Another error caught by TLC is that the system cannot find an action to do when none of the conditions specified are satisfied. TLC reports that "TLC encountered a CASE with no conditions true". This is a condition dependent action, so whenever one of the conditions is true, that action is taken. However, in this case, there is no condition that is satisfied so that the system can not find any action to take, and therefore stops. The example output can be found in Appendix CD.

6.2 Secure Video Transmission

In this section, an approach for securing the authentication, integrity and non-repudiation of transmitted video on our system is described. This approach is also applicable to video transmission via an application level multicast system. The transmitted video may be seen by everyone. The approach employs public key cryptography, so that the private key is known only to the sender and the public key is freely available.

6.2.1 Method

The approach introduced is independent of the encoding algorithm of the video since it operates with packets independently. Integrity and authentication can be provided either by encrypting the entire packet using chunks or by creating a unique digest value and encrypting it. The real-time video demands imposes limits on the key size on both methods. Therefore the encryption and decryption times should be very short.

The first method describes encrypting and decrypting packets either entirely or after dividing them into chunks. With larger key sizes, these operations take enough time to decrease the performance of the solution. As the simulations will show, operations on the entire packet do not work (because of the packet size) and dividing into chunks would either require many packets to be sent (after each division) or to be buffered to build one packet, which would complicate implementation.

The second method; however, can provide a high security level without increasing the overhead much. The encryption and decryption operations take less time and processing power which make it more suitable for secure video transmission. In this method, the sender side generates a unique digest value for each packet. This unique digest value is then encrypted by the private key of the sender creating a digital signature which is appended to the end of the packet. Upon reception of the packet at a receiver, the receiver extracts this signature part from the packet and decrypts it with the public key of the sender. Also, it generates the unique digest value of the packet and compares this with the value received from the sender. If the values are the same, this means that the packet received was not altered along the way. If not, the packet is dropped and appropriate action is taken (e.g., the source may be informed that someone is trying to modify the contents of the packets).

6.2.2 Simulation Results

Simulations were performed on a Pentium IV 2.4 GHz processor with 512 MB of RAM running a 2.4.20 kernel Linux. The bit rate of the video was assumed to be 200kbps which makes 25kBps. Packet size was set to 1400 bytes leading an approximate value of 18 packets/sec. The public exponent used was 25-bit and the private key was generated according to the corresponding key size for 50 runs of simulation.

Encryption and decryption of the entire packet

One approach is to treat the entire packet as a big message and encrypt/decrypt it using the private/public key. This idea has both advantages and disadvantages. First of all, treating the entire packet as a big message makes the implementation easy. Also, encrypting and decrypting it, is a trivial task. However, if the packet is large enough, then the message it represents (1400 bytes = 11200 bits, in our experiments) can overflow the key size, so that the encryption/decryption function would not be one-to-one. This means that an encrypted packet could not be restored by decrypting it. Assuming that the key size is big enough to handle such cases, performance becomes an issue. Since this operation needs to be done in one second for several packets to meet the real-time demands of streaming video, this is clearly not realizable.

Another approach is to divide packets into chunks so that each chunk can be encrypted and decrypted independently: This idea's advantage lies in the part that chunks, a packet is divided into, can be independently encrypted and decrypted. The encrypted and decrypted parts only need to be assembled back to back to form and restore a packet. Since the packet is divided into smaller chunks, preserving the one-to-one mapping of the operations becomes easier. However, this method requires either many packets to be sent after each encryption which would increase communication overhead or to be buffered to build one packet which complicates the implementation.

When considering this method, the issue of determining the chunk size arises. To preserve one-to-one mapping of the encryption and decryption operations, the chunk size should not exceed the key size. Figure 6.4 and Figure 6.5 show encryption and decryption times with different chunk sizes for different key sizes. In Figure 6.4 and Figure 6.5, it can be seen that the encryption and decryption times decrease as the

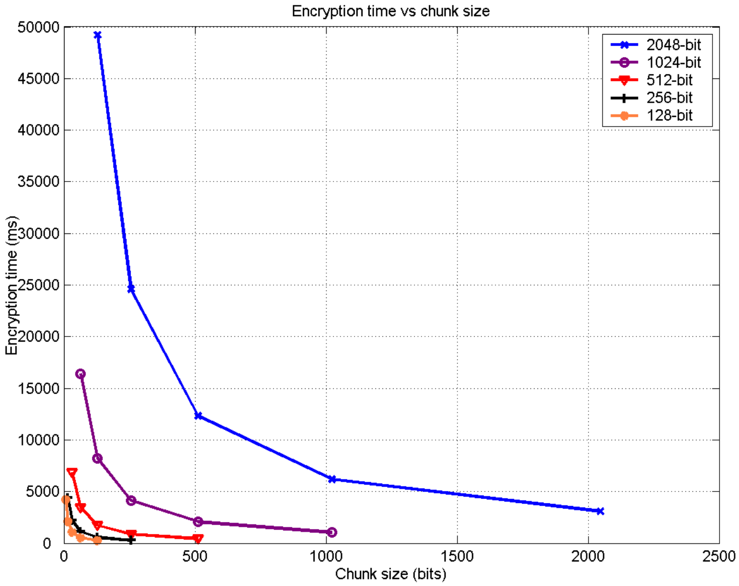


Figure 6.4: Encryption times versus the chunk sizes. Corresponding series show the key sizes.

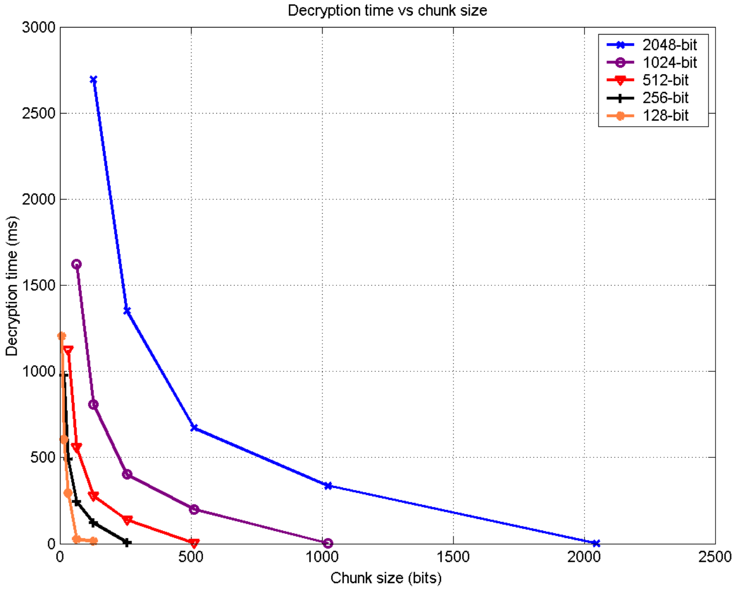


Figure 6.5: Decryption times versus the chunk sizes. Corresponding series show the key sizes.

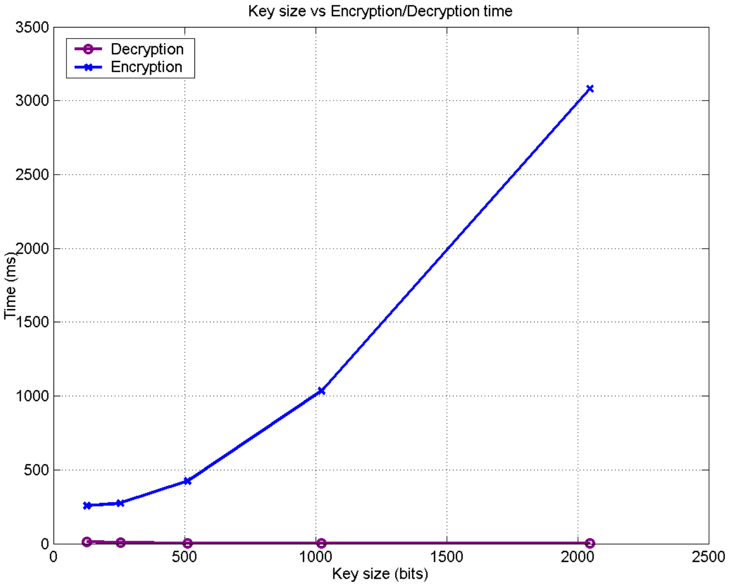


Figure 6.6: Encryption/Decryption times versus the key size. Chunk size is equal to the key size.

chunk size is increasing. Increasing key size increases the encryption and decryption times as shown in Figure 6.6. The optimum values for the key and chunk sizes can be determined considering this information. To provide sufficient security, a large key should be picked. To meet the real-time demands of the streaming video with that key, the chunk size should also be large.

As can be seen from the figures, the encryption times are much larger than the decryption times. This is because the public key exponent is a much smaller number than the corresponding private key exponent generated according to it. These values can be further optimized by using similar sized exponent pairs so that the encryption and decryption times are closer to each other.

Encryption of the unique digest value of each packet

Another technique would be to create a unique digest value of the packet and encrypt this value with the private key like done in digital signatures. Here, the entire packet

Table 6.6: Total time spent on digest value calculation and encryption/decryption times with corresponding key sizes

RSA with SHA-1					
Key size (bits)	128	256	512	1024	2048
Encryption time (ms)	1.08	6.52	19.98	96.36	592.68
Decryption time (ms)	0.41	1.63	3.61	9.74	32.19

is treated as one large message. The digest operation is a one-way function so that the message can not be restored from the digest value. This digest value is ensured to be unique for each packet so that a malicious user would not be able to create another message that goes with this valid digest value and deceive the next receiver. Instead of the packet, its digest value is encrypted with the private key and appended to the packet. The receiver then checks the validity of the digest value by simply decrypting it with the public key.

Since the digest value is just calculated and encrypted only once for the entire packet, this has surely better performance than trying to encrypt all the chunks. In our simulations, we used SHA-1 for digest generation, and RSA for public-key encryption of the digest. Table 6.6 gives the encryption and decryption times measured. As can be seen, calculating a unique digest value and encrypting/decrypting it can meet real-time demands without burdening the sender or the receiver, respectively.

6.2.3 Discussions

In order to achieve integrity, authentication and non-repudiation, encrypting the entire video packet seemed to be a valid way; however, the simulations showed that the security that could be achieved using such an idea would be limited. The maximum key size that could be used in this part of simulation was 1024-bit with the given public and corresponding private key exponent sizes. Although with the optimization described above (using similar sized public and private key exponents) in practice, a lower value is expected since the application would consume processing power while dividing the packets into chunks at the sender side and putting them back together at the receiver side.

On the other hand, calculating a unique digest value and encrypting it worked under all key sizes given, even without the optimization. The calculated digest value is not large (20 bytes when SHA-1 is used) that the sender could not handle, and the main advantage is that it is calculated once per packet. Calculating the digest value is a one-way function, so that recovering the data is impossible. However, by encrypting it, the sender can be sure that the receiver can check the integrity of the packet. Also, source authentication and non-repudiation can be ensured without considering confidentiality because the publicly available service does not need it. Since the encryption is possible only by the sender side because of the private key, an intermediate peer could not change the contents of the packet and send it to another receiver, because the last receiver would notice after calculating the digest value and comparing it with the decrypted one.

The distribution of the public keys of the peers can be done at the beginning of the conference. Since only the public keys are distributed, there is no harm if any malicious user eavesdrops the exchange. The reason is that the public key is used only for verifying the received video packets and not generating new ones.

Chapter 7

CONCLUSION

7.1 Concluding Remarks

We proposed and presented an extension to the P2P architecture for MP video conferencing [3]. Employing layered video, this extension makes it possible to find a feasible solution under all configurations, so that a participant's video request can always be granted at anytime. Simulations show that with the increasing number of participants, the use of layers and thus, base quality video receivers is inevitable; however, the ratio of the base quality receivers to the total number of participants remains under 50%. We implemented the proposed architecture and developed the fully distributed protocol. The effects of multiple output bandwidths are investigated.

Two layered video encoding techniques, Layered Coding and Multiple Description Coding, that can be employed within our architecture are presented. We also have shown that even with limited bandwidth (i.e., 30 kbps for base and 60 kbps for full quality), the video is in acceptable quality.

Furthermore, we proposed a multi-objective optimization approach for a P2P video conferencing system. The system makes compromises between the quality of the video peers receive, delays experienced by peers and additional requests of peers with sufficient bandwidth. Our multi-objective formulation successfully finds the best-compromise solution. We have shown with a counter-example that the solution should be calculated according to the preferences of *all* participants in the conference, not just in a corresponding chain. The sensitivity analysis showed that our objective function reacts to the changes in the preferences of the participants and satisfies them in the best way. Although the employed solution causes more requests to receive base layer video quality, simulation results show that this does not damage the system's scalability. Since the intended group size is relatively small, generation of possible chain configurations does not bring much computational burden. This makes the system easy to implement.

The formal specification is made using TLA+ and verified with TLC model checker. The correspondence between the TLA+ specification and the prototype implementation has been described and explained. No error or deadlock has been found on the protocol.

Since peers relay video to other peers, intermediate peers may alter the content. To prevent this, a packet-based security scheme is proposed. In this scheme, the sender calculates a digest per packet and encrypts it with its private key. The receivers decrypt the digest value with the sender's public key and compare the calculated value with the piggybacked value. Simulations to show the efficiency and low overhead of the proposed system have been carried out. It meets the demands of real-time video even with large key-sizes and provides authentication, integrity and non-repudiation.

7.2 Future Work

It is shown that only two layers of video are required to grant each request in a conference where participants have one input, one output and request to watch one participant. We call this type of conference as a "basic conference". One can extend this scheme for conferences where participants have n inputs and n outputs and request to watch n other participants. Each request set of participants can be thought as a basic conference and each request can be granted by using two layers. This brings an *upper bound* of *2n* to the number of required layers of video that can be used in this type of conference to grant each request. Future work may try to determine the *lower bound*.

Future work also includes exploring of other multi-objective optimization techniques, in which greedy approaches may be employed. The number of additional granted requests may be increased until the number of base layer receivers exceeds some threshold. A possible threshold value is the half of the participant count. The number of base layer receivers does not exceed 50% of the participant count as shown in [62]; so this would be a good candidate. However, this may require a mechanism to exchange information about the number of base layer receivers between the chain heads. Also, an algorithm for better estimation of the one-way delays between the participants needs to be implemented, instead of assuming symmetric delays.

BIBLIOGRAPHY

[1] ITU-T Study Group XV - Recommendation H.231: Multipoint Control Units for audiovisual systems using digital channels up to 1920 kbits/s, (1993).

[2] M. R. Civanlar, R. D. Gaglianello, G. L. Cash: Efficient Multi-Resolution, Multi-Stream Video Systems Using Standard Codecs, Journal of VLSI Signal Processing, (1997).

[3] M. R. Civanlar, O. Ozkasap, T. Celebi: Peer-to-peer multipoint video conferencing on the Internet, Signal Processing: Image Communication 20, pp. 743-754, (2005).

[4] T. Celebi: Peer-to-peer Multipoint Video Conferencing, Master's Thesis, Koc University, Graduate School of Sciences and Engineering, (2005).

[5] Lamport L.: The Temporal Logic of Actions, ACM Toplas 16, 3 pp. 872-923, (1994).

[6] http://research.microsoft.com/users/lamport/tla/tlc.html, (last accessed on January 29, 2007).

[7] Y. Chu, S. G. Rao, H. Zhang: A case for end system multicast, Proceedings of ACM Sigmetrics, (2000).

[8] Y. Chu, S. G. Rao, S. Seshan, H. Zhang: Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture, Proceedings of ACM SIGCOMM'01, San Diego, CA, (2001).

[9] Y. Chu, S. G. Rao, S. Seshan, H. Zhang: A Case for End System Multicast, IEEE Journal on Selected Areas in Communications, vol.20, No.8, October (2002).

[10] S. Banerjee, B. Bhattacharjee, C. Kommareddy: Scalable application layer multicast, Proceedings of ACM SIGCOMM'02, Pittsburgh, Pennsylvania, (2002).

[11] S. Banerjee, S. Lee, B. Bhattacharjee, A. Srinivasan: Resilient Multicast Using Overlays, Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, San Diego, CA, (2003).

[12] D. A. Tran, K. A. Hua, T. Do: ZIGZAG: An efficient peer-to-peer scheme for media streaming, Proceedings of IEEE Infocom, (2003).

[13] X. Jiang, Y. Dong, D. Xu, B. Bhargava: GnuStream: A P2P Media Streaming System Prototype, Proceedings of the International Conference on Multimedia and Exposure (ICME'03), Volume 2, pp. 325-328, (2003).

[14] Y. Guo, K. Suh, J. Kurose, D. Towsley: P2Cast: Peer-to-peer Patching Scheme for VoD Service, Proceedings of the 12th international conference on World Wide Web, pp. 301-309 (2003).

[15] T. T. Do, K. A. Hua, M. A. Tantaoui: P2VoD: Providing Fault Tolerant Video-on-Demand Streaming in Peer-to-peer Environment, IEEE International Conference on Communications, vol. 3, pp. 1467-1472, (2004).

[16] M. Castro, P. Druschel, A. M. Kermarrec, A. Nandi, A. Rowstron, A. Singh: SplitStream: High-Bandwidth Multicast in Cooperative Environments, ACM SIGOPS Operating Systems Review Volume 37, Issue 5 pp. 298-313, (2003).

[17] A. I. T. Rowstron, P. Druschel: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems, Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, pp.329-350, (2001).

[18] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan: Chord: A scalable peer-to-peer lookup service for internet applications, Proceedings of the

2001 conference on Applications, technologies, architectures, and protocols for computer communications, pp.149-160, San Diego, CA, (2001).

[19] M. Castro, P. Druschel, A. M. Kermarrec, A. Rowstron: SCRIBE: A large-scale and decentralized application-level multicast infrastructure, IEEE JSAC, 20(8), (2002).

[20] V. N. Padmanabhan, H. J. Wang, P. A. Chou, K. Sripanidkulchai: Distributing streaming media content using cooperative networking, Proceedings of NOSSDAV'02, Miami, FL, (2002).

[21] V. N. Padmanabhan, H. J. Wang, P. A. Chou: Resilient Peer-to-Peer Streaming, Proceedings of the 11th IEEE International Conference on Network Protocols, ICNP'03, pp. 16-27, (2003).

[22] Y. Shen, Z. Liu, S. S. Panwar, K. W. Ross, Y. Wang: Streaming Layered Encoded Video Using Peers, IEEE International Conference on Multimedia and Exposure (ICME'05), (2005).

[23] I. Lee, L. Guan: Centralized peer-to-peer streaming with layered video, International Conference on Multimedia and Exposure, ICME'03, vol. 1, pp. 513-516, (2003).

[24] Y. Cui, K. Nahrstedt: Layered Peer-to-Peer Streaming, Proceedings of NOSSDAV'03, Monterey, CA, (2003).

[25] T. Silverston, O. Fourmaux: Source vs. Data-driven Approach for Live P2P Streaming, International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL'06), (2006).

[26] M. Bawa, H. Deshpande, H. Garcia-Molina: Transience of Peers & Streaming Media, ACM SIGCOMM Computer Communication Review, volume 33 issue 1 pp.107-112, (2003).

[27] H. Deshpande, M. Bawa, H. Garcia-Molina: Streaming Live Media over Peers, Technical Report, Computer Science Department, Stanford University, April, (2001).

[28] X. Zhang, J. Liu, B. Li, T. S. P. Yum: CoolStreaming/DONet: A Data-driven Overlay Network for Peer-to-peer Live Media Streaming, Proceedings of the IEEE INFOCOM, (2005).

[29] X. Liao, H. Jin, Y. Liu, L.M. Ni, D. Deng: AnySee: Peer-to-peer Live Streaming, Proceedings of IEEE INFOCOM, (2006).

[30] S. T. Chanson, A. Hui, E. Siu, I. Beier, H. Koenig, M. Zuehlke: OCTOPUS - A Scalable Global Multiparty Video Conferencing System, Proceedings of the IEEE eight International Conference on Computer Communications and Networks (IC3N'99), (1999).

[31] M. Zuehlke, H. Koenig: Voting Based Bandwidth Management in Multiparty Video Conferences, Proceedings of the Joint International Workshops on Interactive Distributed Multimedia Systems and Protocols for Multimedia Systems: Protocols and Systems for Interactive Distributed Multimedia:, IDMS/PROMS 2002, pp.202-215, (2002).

[32] M. Hosseini, N. D. Georganas: Design of a Multi-Sender 3D Videoconferencing Application over an End System Multicast Protocol, ACM, Multimedia2003, Berkeley, CA, (2003).

[33] ISO 8807: LOTOS – A formal description technique based on the temporal ordering of observational behaviour, (1989).

[34] http://www.run.montefiore.ulg.ac.be/Projects/Presentation/index.php?project=Eucalyptus, (last accessed on January 27, 2007).

[35] http://www.run.montefiore.ulg.ac.be/Projects/Presentation/Apero/apero.html, (last accessed on January 27, 2007).

[36] http://www.spinroot.com (last accessed on January 27, 2007).

[37] Holzmann G. J., Peled D.: An Improvement in Formal Verification, Proceedings of FORTE Conference, Bern, Switzerland, (1994).

[38] Holzmann G. J.: An analysis of bitstate hashing, Formal Methods in Systems Design, November, (1998).

[39] de Renesse R., Aghvami A.H.: Formal Verification of Ad-Hoc Routing Protocols Using SPIN Model Checker, IEEE MELECON, Dubrovnik, Croatia, (2004).

[40] Ramasamy H. V., Cukier M., Sanders W. H.: Formal Specification and Verification of a Group Membership Protocol for an Intrusion-Tolerant Group Communication System, Proceedings of the Pacific Rim International Symposium on Dependable Computing, PRDC'02, (2002).

[41] Abadi M., Lamport L., Merz S.: A TLA Solution to the RPC-Memory Specification Problem, Formal Systems Specification: The RPC-Memory Specification Case Study, LNCS 1169, pp. 21-66, (1996).

[42] Lamport L., Merz S.: Specifying and Verifying Fault-Tolerant Systems, Formal Techniques in Real-Time and Fault-Tolerant Systems, LNCS 863, pp. 41-76, (1994).

[43] Akhiani H., Doligez D., Harter P., Lamport L., Tuttle M., Yu Y,: TLA+ Verification of Cache-Coherence Protocols, Compaq Project Report, (1999).

[44] Spanos, G. A., Maples, T. B.: Performance Study of a Selective Encryption Scheme for the Security of Networked Real-time Video. Forth International Conference on Computer Communications and Networks, pp. 2-10, (1995).

[45] Li, Y., Chen, Z., Tan, S. M., Campbell R. H.: Security Enhanced MPEG Player. IEEE 1st International Workshop on Multimedia Software, (1996).

[46] Agi, I., Gong, L.: An Empirical Study of MPEG Video Transmission. Proceedings of the Internet Society Symposium on Network and Distributed System Security, pp. 137-144, (1996).

[47] Venkatramani, C., Westerink, P., Verscheure, O., Frossard, P.: Securing Media For Adaptive Streaming, Proceedings of the eleventh ACM international conference on Multimedia, ACM Press, (2003).

[48] Baugher, M., McGrew, D., Naslund, M., Carrara, E., Norrman, K.: RFC3711 The Secure Real-time Transport Protocol (SRTP), (2004).

[49] Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V.: RTP: A Transport Protocol for Real-Time Applications, (2003).

[50] Tang, L.: Methods for Encrypting and Decrypting MPEG Video Data Efficiently, Proceedings of the fourth ACM international conference on Multimedia, ACM Press, (1997).

[51] Changgui, S., Bhargava, B.: A Fast MPEG Video Encryption Algorithm, Proceedings of the sixth ACM international conference on Multimedia, ACM Press, (1998).

[52] Liu, F., Koenig, H.: A Novel Encryption Algorithm for High Resolution Video, Proceedings of the international workshop on Network and operating systems support for digital audio and video NOSSDAV '05, (2005).

[53] Perrig A., Song D., Canetti R., Tygar J. D., Briscoe B.: RFC4082 Timed Efficient Stream Loss-Tolerant Authentication (TESLA): Multicast Source Authentication Transform Introduction, (2005).

[54] Wong, C. K., Lam, S. S.: Digital Signatures for Flows and Multicasts, EEE/ACM Transactions on Networking (TON), Vol 7 Issue 4, IEEE Press, (1999).

[55] D. E. Meddour, M. Mushtaq, T. Ahmed: Open Issues in P2P Multimedia Streaming, to appear in proceedings of the MULTICOMM2006, (2006).

[56] S. Kulkarni, J. Markham: Split and Merge Multicast: Live Media Streaming with Application Level Multicast, IEEE International Conference on Communications (ICC'05) vol. 2 pp. 1292-1298, (2005).

[57] S. Kulkarni: Video Streaming on the Internet Using Split and Merge Multicast, Proceedings of the Sixth IEEE International Conference on Peer-to-peer Computing (P2P'06), (2006).

[58] JSVM Software Available at CVS Repository of JSVM pserver:jvtuser@garcon.ient.rwth-aachen.de:/cvs/jvt, (last accessed on January 27, 2007).

[59] Nokia H.264 codec Available at ftp://standards.polycom.com/ IMTC_Media_Coding_AG/, (last accessed on July 20, 2007).

[60] Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 — ISO/IEC 14496-10 AVC), (2003).

[61] V. K. Goyal: Multiple Description Coding: Compression Meets the Network, IEEE Signal Processing Magazine, Volume 18, Issue 5, pp. 74-93, (2001).

[62] I. E. Akkus, M. R. Civanlar, O. Ozkasap: Peer-to-peer Multipoint Video Conferencing Using Layered Video, Proceedings of the IEEE 13th International Conference on Image Processing, ICIP2006, Atlanta, GA, (2006).

[63] L. Zadeh: Optimality and Non-Scalar-Valued Performance Criteria, IEEE Trans. Automatica Control, vol.8, pp. 59-60, (1963).

[64] I. E. Akkus, O. Ozkasap, M. R. Civanlar: Secure Transmission of Video on an End System Multicast Using Public Key Cryptography, Multimedia Content Representation, Classification and Security, Lecture Notes in Computer Science, Vol:4105, pp: 603-610, (2006).

VITA

İstemi Ekin Akkuş was born in İstanbul, Turkey on June 11, 1981. He received his BSc. degrees in Mechanical Engineering and Computer Engineering from Koç University, İstanbul. From September 2005 to August 2007, he worked as a teaching and research assistant at the Network and Distributed Systems Laboratory (NDSL) at Koç University. His project "Peer-to-peer Multipoint Video Conferencing Using Layered Video" was sponsored by TÜBİTAK.