MULTI-STREAM MANAGEMENT FOR SUPPORTING MULTI-PARTY 3D TELE-IMMERSIVE ENVIRONMENTS

BY

ZHENYU YANG

B.E., Shanghai Jiao Tong University, 1994
M.S., University of Illinois at Urbana-Champaign, 2002

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2007

Urbana, Illinois

Abstract

Three-dimensional tele-immersive (3DTI) environments have great potential to promote collaborative work among geographically distributed participants. However, extensive application of 3DTI environments is still hindered by the problems pertaining to scalability, manageability and reliance of special-purpose components. Most existing 3DTI systems either do not provide multi-party connectivity or require dedicated resources. Thus, one critical question is how to organize the acquisition, transmission and display of large volume real-time 3D visual data over commercially available computing and networking infrastructures so that "*everybody*" would be able to install and enjoy 3DTI environments for high quality tele-collaboration.

In the thesis, we explore the design space from the angle of multi-stream Quality-of-Service (QoS) management to support multi-party 3DTI communication. In 3DTI environments, multiple correlated 3D video streams are deployed to provide a comprehensive representation of the physical scene. Traditional QoS approach in 2D and single-stream scenario has become inadequate. On the other hand, the existence of multiple streams provides unique opportunity for QoS provisioning. Previous work mostly concentrated on compression and adaptation techniques on the per stream basis while ignoring the application layer semantics and the coordination required among streams.

As the result of research, we propose an innovative cross-layer hierarchical and distributed multi-stream management middleware framework for QoS provisioning to fully enable multi-party 3DTI communication over general delivery infrastructure. The major contributions of our management framework are as follows. First, we introduce the *view* model for repre-

senting the user interest in the application layer. The design of the management framework revolves around the concept of *view*-aware multi-stream coordination, which leverages the central role of view semantics in 3D free-viewpoint video systems. Second, in the *stream differentiation* layer we present the design of view to stream mapping, where a subset of relevant streams are selected based on the relative importance of each stream to the current view. Conventional streaming controllers focus on a fixed set of streams specified by the application. Different from all the others, in our management framework the application layer only specifies the view information while the underlying controller dynamically determines the set of streams to be managed. Third, in the *stream coordination* layer we present two designs applicable in different situations. In the case of end-to-end 3DTI communication, a learning-based controller is embedded which provides bandwidth allocation for relevant streams. In the case of multi-party 3DTI communication, we propose a novel *ViewCast* protocol to coordinate the multi-stream content dissemination upon an end-system overlay network. Finally, we embed 3DTI session management in the framework which facilitates the session initialization, resource registration, and membership maintenance.

We implement the prototype of multi-stream management framework and evaluate it through both simulation and real 3DTI session among tele-immersive environments residing in different institutions across the Internet2. Our experimental results have demonstrated the implementation feasibility and performance enhancement of the management framework.

To ...

Acknowledgments

Most of all, I would like to express my deepest gratituide to my advisor, Professor Klara Nahrstedt, for her invaluable support throughout my Ph.D. pursuit. With her insightful suggestions, she guided me towards important research topics while helping and encouraging me to explore solutions. To me, Professor Klara Nahrstedt has been a great mentor. I am deeply impressed by how she cared about every developmental detail of her students from plan of study towards career choice. It is such a wonderful experience working with her. The memory that I will cherish forever.

I would like to thank my honorable committee members: Professor Roy H. Campbell, Professor Thomas S. Huang, and Professor Benjamin W. Wah for their helpful technical discussions. Their insightful comments on my thesis have greatly helped me to improve the quality of this work.

Thanks go towards people involved in the TEEVE (**T**ele-immersive **E**nvironments for **Eve**rybody) project. Most importantly, I would like to thank Professor Ruzena Bajcsy in the University of California at Berkeley for her critical support of the project. I would also like to thank Wanmin Wu, Renata Sheppard, Gregorij Kurillo, Peter Pajcsy, Yi Cui, Bin Yu, Jin Liang, Dongyun Jin, Miles Johnson, Lisa Wymore, Sang-Hack Jung, Cynthia Bruyns, Katherine Mezur, Ross Diankov, Samuel Johnston, Roger Cheng, Muyuan Wang, Zahid Anwar, Robert Bocchino, Nadir Kiyanclar, Art Yeap, William Yurcik, Bradford Wilson, Jeffrey Naisbitt, Jigar Doshi, and Ravishankar Sathyam for their important contribution in implementation, construction, experiment and publication, which manifested the very essence of collaborative work.

I am grateful to my colleagues in the MONET (**M**ultimedia **O**perating Systems and **Net**working) group. It is very lucky for me to work with them and be influenced by their high standard of devotion to research. Special thanks go to Wanghong Yuan, Baochun Li, Xiaohui Gu, Kai Chen, Yuan Xue and Xiao Li for their excellent samples of writing. Thanks to Wenbo He for several informative discussions. I am grateful to Anda Ohlsson, Erna Amerman, Barb Cicone, Anthony Hooker, Mary Beth Kelley, Debby Reynolds, Kay Tomlin and Shirley Finke for their great administrative support during my graduate study in the University of Illinois at Urbana-Champaign.

Finally, I would like to say *"Thank You ..."* to my wife, Yihe Zu, and my whole family for their greatest love and support.

The work presented in the thesis was supported by National Science Foundation under NSF SCI 05-49242 and NSF CNS 05-20182. However, views and conclusions of this thesis are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of NSF.

Table of Contents

List of Tables

List of Figures

List of Abbreviations and Notations

Chapter 1

Introduction

The work of multi-stream management middleware framework is motivated by the provision of Quality-of-Service (QoS) to support three-dimensional tele-immersive (3DTI) environments under general and often limited system resources. In this chapter, we introduce our research motivations, review currently available solutions, present the overview of the management framework, and summarize the major contributions. Finally, we outline the rest of the thesis.

1.1 Motivation

3DTI environments have great potential to promote collaborative work among geographically distributed participants. Earlier research efforts ([50, 17, 28, 42]) have illustrated possible applications of 3DTI environments in various areas such as scientific research, medical science, artistic performance, education, physical therapy, training and entertainment, where a higher level of spatial interactivity is desired. Meanwhile, end-devices (e.g., 3D cameras and displays) that make the tele-immersive edge applications possible are becoming more available and deployable due to advance in hardware. Consequently, there have been various efforts to create tele-conferencing and tele-immersive environments (e.g., [10, 15, 20, 39, 48, 47]). The current approaches represent a very good start for the next generation of tele-immersive systems where the ultimate goal is to deliver 3DTI experience to the broader audience.

However, disregarding the promise extensive deployment of 3DTI environments is still hindered by the problems pertaining to scalability, manageability and reliance of special-

purpose operating and networking infrastructures. There are two major deficiencies in current work. First, most existing 3DTI systems either do not provide 3D multi-stream immersive content or require dedicated computing and networking components. We argue that with the advance of end-devices it is now practical to further extend the application of 3DTI environments with general content creation and delivery infrastructures. Second, most existing systems only support the inter-connection of two parties across the Internet. Enabling multi-party 3DTI collaboration is still challenging due to the huge demand of computing and networking resources.

Hence, one critical question is how to organize the large volume of 3D visual data, being captured, processed, transmitted and rendered, and their corresponding resources, over current commercially available (COTS) computing and networking infrastructures for the delivery of realistic immersive experience so that "*everybody*" would be able to install and enjoy 3DTI environments for high quality tele-collaboration (Figure 1.1).

Figure 1.1: Collaborative 3DTI Environments

In the thesis, we investigate the design space from the angle of multi-stream QoS management between the 3D multi-camera/multi-display tele-immersive edges and the general

purpose operating and communicating infrastructures available. The design space includes the functions of application layer for capturing, reconstructing and displaying 3D video content, as well as the functions of distributed middleware layer for compressing, streaming, and coordinating 3D video content across the Internet.

1.2 Research Challenges

There are three basic tiers in the core of 3DTI environments (Figure 1.2). In the *capturing tier*, each environment installs an array of 3D cameras at various angles to cover a wide field of view. Using real-time computer vision techniques, the camera array dynamically derives the 3D representation of the user in multiple video streams with each corresponding to one camera. In the *transmission tier*, the generated video streams are exchanged with remote tele-immersive environments. Given a global coordinate system, the 3D representations from different environments are merged and rendered together in a common 3D virtual space by the *rendering tier*, delivering a strong awareness of immersive experience for every participant.

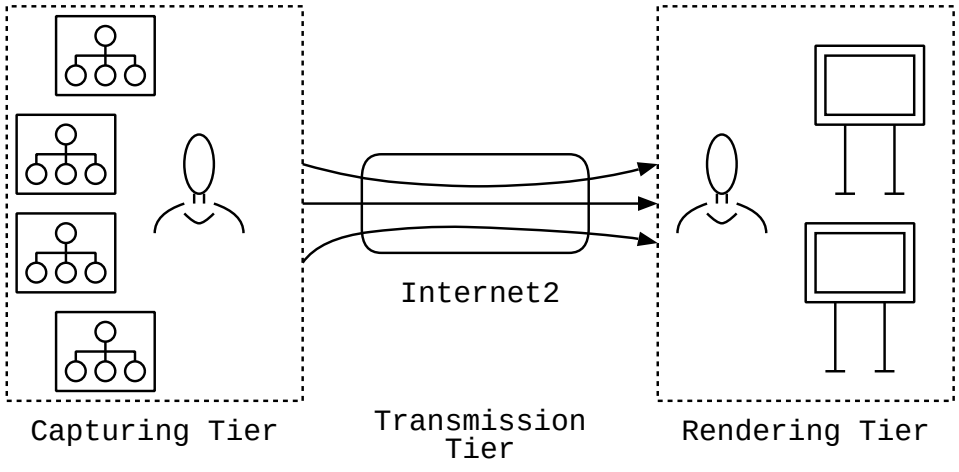


Figure 1.2: Three Basic Tiers of 3DTI Environments

The work of the thesis is mainly focused in the transmission tier, which investigates the issue of QoS provisioning to address the following challenges in multi-party/multi-stream

3DTI environments.

- **Large-volume Data.** To achieve realistic 3D visual effect, it is desirable to transmit multiple video streams from each 3DTI environment of one communication session. In our experimental system ([57, 4]), one 3D video stream has the basic rate over 30 Mbps (currently at the resolution of 320×240 pixels per 3D image frame and 10 frames per second) to support spatial collaboration, and each environment produces up to 10 streams. The frame resolution is being upgraded to 640×480. If all streams are sent, the overall bandwidth from one environment will soon exceed Gbps level. The problem would become even more exacerbated if multiple environments were connected. Suppose the basic data rate of one stream is R, the number of streams per environment is N_s, the number of environments in one 3DTI session is N_e, and the number of 3DTI renderers is N_r. Then the total amount of data to be transmitted would become $R \times N_s \times N_e \times N_r$, which is a very significant demand at the networking scale of Internet.

- **Rendering Cost.** At the resolution of 320×240 per frame and 10 frames per second, a 3DTI envrionment with 10 streams requires a rendering capacity of 7.7 M points per second. Unlike 3D capturing and reconstruction, the parallelization of rendering is much more difficult as all streams must be rendered in one single virtual space. Thus, the cost of rendering grows linearly with the total number of streams sent to the rendering process (i.e., $N_s \times N_e$).

- **Stream Correlation.** In one tele-immersive environment, video streams derived from the 3D camera array are correlated as all cameras are calibrated and synchronized to concurrently capture the visual information of a common physical scene. The rendering quality depends on the overall contribution of streams. This multi-stream content feature, combined with the bound imposed by the bandwidth and rendering overhead, demands for the design of *multi-stream coordination* in the transmission tier.

- **View-based Rendering.** Unlike traditional 2D video rendering, 3D video rendering is an interactive process. In order to render 3D objects with correct visual effect, the displaying device needs to keep track of the user view information (using tracking devices or mouse and keyboard) and render the 3D scene accordingly. The interactivity through view selection is the key feature of 3D video applications ([6]). The problem is how to incorporate the view semantics into the management design for a more efficient QoS provisioning not achievable through previous 2D QoS techniques. We need to point out that view-based rendering and stream correlation are *dynamic* concepts as the user view could change arbitrarily during one 3DTI session.

- **Multi-party Connectivity.** Finally, the problem of connecting multiple 3DTI environments has become more complicated due to the aforementioned challenges. Because of the huge data volume, it is impractical to take the approach of a unicast based dissemination scheme. However, most available multicast schemes are stream-oriented which do not have the desired flexibility to accommodate the dynamics of stream correlation and view semantics as in the 3DTI application layer.

1.3 Existing Solutions

Before introducing our multi-stream management framework, we briefly describe current available solutions. More careful review of related work is in Chapter 8. We summarize previous work in five main aspects: (1) existing tele-immersive systems, (2) real-time 3D video compression, (3) multi-stream coordination, (4) view-based camera/stream selection, and (5) multicast-based content dissemination.

1.3.1 Existing Tele-immersive Systems

There are several existing systems that aim to provide tele-immersive realism to users (e.g., [39, 48, 47, 51, 10]). Due to the huge volume of 3D data stream, it poses significant challenges

to the current networking resources. Most of them resort to either modifying the transport and network services or relying on dedicated components. Among them, only [10] has the support of multi-party connection. However, instead of multiple 3D streams only single 2D stream after rendering is transmitted which greatly lowers the spatial interactivity among different parties.

1.3.2 Real-time 3D Video Compression

Real-time 3D video compression algorithms can be classified into two categories: *inter-stream* compression and *intra-stream* compression. In inter-stream compression ([31, 29]), streams are cross-compared to exploit the spatial redundancy residing in the multi-stream setting. The advantage of inter-stream compression is the removal of redundant pixels. Thus, the rendering overhead is reduced as well. However, inter-stream compression incurs considerable communication overhead as streams are initially distributed, which affects its scalability. In addition, the performance of compression ratio is highly associated with the density of cameras. In contrast, intra-stream compression schemes ([56, 30]) process each stream individually without cross-stream comparison. Intra-stream compression usually achieves much better compression ratio and scalability than inter-stream compression. However, one disadvantage of intra-stream compression is that the number of pixels is not reduced. Real-time 3D video compression provides a low-level mechanism for solving the data volume problem. However, in a multi-stream scenario it is not sufficient to deal with high-level concepts of view semantics and multi-stream coordination.

1.3.3 Multi-stream Coordination

Coordination Protocol (CP) ([39, 40]) is a transport layer protocol used for assisting multi-stream coordination in cluster-to-cluster application. The protocol utilizes dedicated routers deployed at the aggregation point to monitor the status of application layer streams. The

advantage of CP is that it provides a general means where per stream information can be collected and disseminated for high level multi-stream management. However, the protocol has not addressed the real problem of multi-stream coordination.

1.3.4 View-based Camera/Stream Selection

In tele-immersive systems, view-based camera/stream selection is applied mainly in the 3D video processing and encoding stage to make it affordable within processing capacity (e.g., [38, 22]). However, there is not much work involved with camera/stream selection for QoS provisioning. In collaborative virtual systems, the awareness-driven model has been applied for QoS management ([21, 44, 23]). Given the awareness information of the user, the model dynamically adjusts the set of sources and the quality. However, the limitation of the approach lies in its incapability of handling multiple correlated streams at each source and among sources as required in multi-party/multi-stream 3DTI environments.

1.3.5 Multicast-based Content Dissemination

Multicast protocols including application level multicast ([26, 11, 24]) are mostly concerned with the efficient transmission of one particular stream or a set of streams for a group of receivers. However, stream-oriented multicast schemes are limited in their flexibility of accommodating the dynamics of view and stream correlation as in 3DTI environments.

As we have seen, although solid progresses have been made in several tele-immersive systems the goal of building 3DTI environments over general infrastructures with high quality and wide usage has not been fully accomplished. Many existing efforts have made explorations in individual aspects of the design space. However, none of them provide a comprehensive and semantic-aware multi-stream management framework that is adequate to support QoS provisioning in multi-party 3DTI environments.

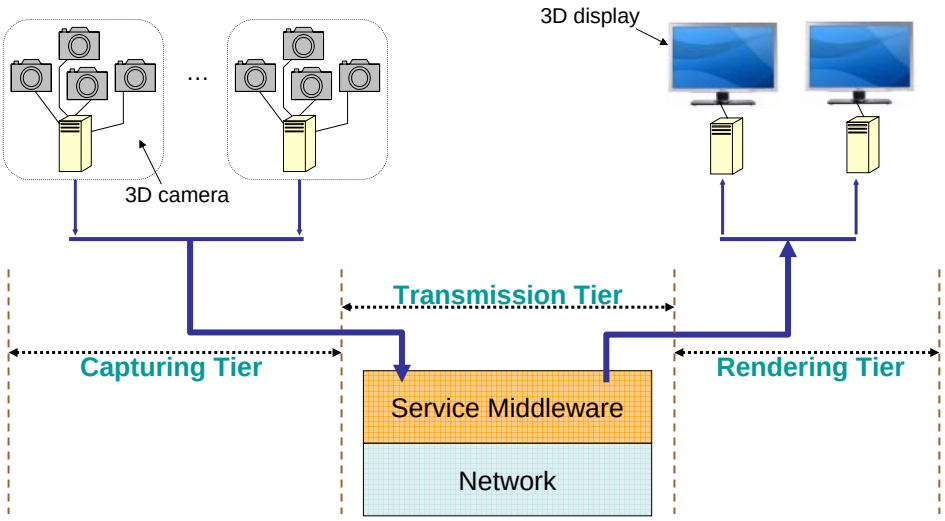


Figure 1.3: Service Middleware Layer in 3DTI Architecture

1.4 Solution Overview

As for the solution, we embed a distributed service middleware framework to provide multi-stream QoS management (Figure 1.3). The middleware is designed and implemented on the basis of COTS components and general networking infrastructure. Thus, the framework has the advantage of easy deployment and configuration. The service middleware framework has the hierarchical structure as illustrated in Figure 1.4.

- **Stream Differentiation.** The task of stream differentiation layer is to perform the view to stream mapping and to prioritize streams. When the 3D camera array is bootstrapped, it registers with the stream differentiation layer to save the meta-data of cameras. During the 3DTI session, the meta-data will be used by the stream differentiation layer to dynamically compute the relative importance of each stream based on the current user view information captured by the rendering tier. Conventional streaming controllers focus on a fixed set of streams specified by the application. Different from all the others, in our management framework the application layer only specifies the view information while the underlying controller dynamically determines

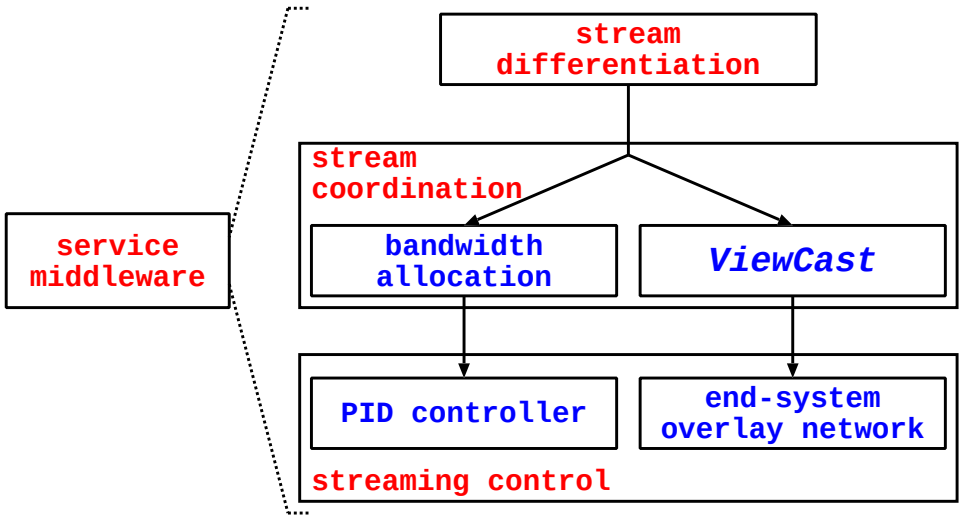


Figure 1.4: The Structure of Service Middleware Layer

the set of streams to be managed.

- **Stream Coordination** The stream coordination layer is responsible for multi-stream coordination to deliver appropriate stream content and achieve multi-stream QoS management. We introduce two different approaches depending on the application scenario. In the case of end-to-end 3DTI communication, a learning-based strategy is applied to determine the bandwidth allocation for the most important streams. In the case of multi-party 3DTI communication, we propose a novel *ViewCast* protocol to optimize the overlay topology for content dissemination at the stream level. The ViewCast protocol also includes session management functions such as session initialization, resource registration, and membership maintenance.

- **Streaming Control.** In the case of end-to-end 3DTI communication, the streaming control layer utilizes a PID (proportional-integral-derivative) controller to monitor the link status and guarantee temporal quality of data transmission for interactive tele-communication. In the case of multi-party 3DTI communication, we assume the existence of an end-system overlay network which provides the information of link status to be used for streaming control.

In real implementation, the service middleware framework is made of *service gateways*. Each 3DTI environment is managed by one or more service gateways. The end-devices of 3D camera and display need to register with their local service gateways in order to receive QoS management service. From a global scale, service gateways are connected with each other to form a distributed management overlay network. To alleviate the burden of edge computers (i.e., 3D reconstruction and rendering), service gateways are running on dedicated computers and connected with local cameras and displays via high-speed LAN. The connection between service gateways could be LAN, WAN or Internet, depending on the scale of distribution.

1.5 Major Contributions

The major contributions of our management framework are summarized as follows.

- **Comprehensive Management Framework.** The hierarchical management framework integrates different layers to achieve a comprehensive QoS provisioning. The stream differentiation layer responds to view changes and provides a high-level guidance for multi-stream coordination. In the stream coordination layer, the granularity reduces to a set of relevant streams. Finally, the streaming control layer focuses more on temporal streaming quality of assigned streams. Through the layering organization, different QoS concerns are properly addressed within a unified framework.

- **View-oriented Content Dissemination.** We present a novel ViewCast protocol to coordinate multi-stream content dissemination on the top of end-system overlay network for supporting multi-party 3DTI communication. Different from all other stream-oriented multicast protocols, the scope of QoS management in ViewCast is not bounded by a fixed set of streams. The new view-oriented approach brings more flexibility, customization and adaptability to the design.

- **Support of Multi-party 3DTI Communication.** To the best of our knowledge, we are the first one who present a feasible solution to support multi-party 3DTI communication with multi-stream 3D video content. Our work will provide valueable reference to the future generation of 3DTI systems which extend towards larger user group and more interesting collaborative activity.

- **Implementation and Validation.** We implement the prototype of multi-stream management framework as part of the TEEVE (**T**ele-immersive **E**nvironments for **EVE**body, [57, 4]) project and evaluate it through both simulation and real 3DTI session among tele-immersive environments residing in different institutions more than 2000 miles apart across the Internet2. Our experimental investigation demonstrates the implementation feasibility and potential performance enhancement of the management framework.

1.6 Roadmap of the Thesis

The rest of the thesis is organized as follows. Chapter 2 introduces the 3DTI system and the architecture of multi-stream management framework. Chapter 3 presents the view-based stream differentiation. Chapter 4 presents the learning-based bandwidth allocation for QoS optimization in the point-to-point case, and evaluation results from real 3DTI experiment. Chapter 5 introduces the ViewCast content dissemination protocal for the multi-party 3DTI communication. Chapter 6 introduces the streaming control layer and a PID controller for maintaining temporal quality of streaming. For completeness, we describe 3D video compression in Chapter 7. We review related work in Chapter 8. Finally, Chapter 9 summarizes the thesis and suggests future work.

Chapter 2

3DTI System Overview

In this chapter, we introduce the 3DTI system. First, we describe the data, view and timing model. Second, we introduce the multi-stream management framework. Finally, we present the 3DTI architecture.

2.1 3DTI Models

2.1.1 Data Model

The overall 3DTI data model consists of two parts: (a) the 3D reconstructed video data model that represents the information coming out from one single 3D camera, and (b) the integrated data model that includes all video streams captured concurrently by the 3D camera array.

3D Video Data Model. The data from a single 3D camera constitutes one stream of 3D frames representing the color and spatial information of the physical scene captured from the particular viewpoint of the 3D camera. The spatial information can be obtained by applying stereo algorithms on images captured from multiple 2D cameras. In general, given two or more images of a scene, depth of each point in 3D space can be obtained by finding matching points on different images and using triangulation.

More specifically, in our 3DTI setting one 3D camera consists of four aligned 2D digital cameras that form a basic processing unit which is used to capture both color and depth information. Figure 2.1 shows one 3D camera unit, which has one color camera on the top

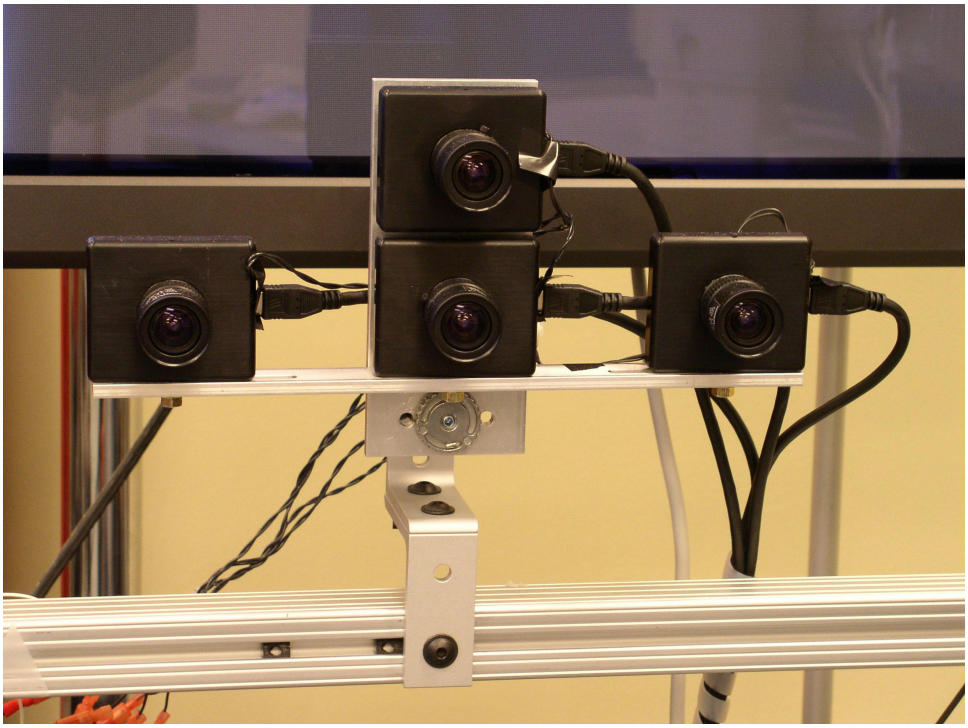
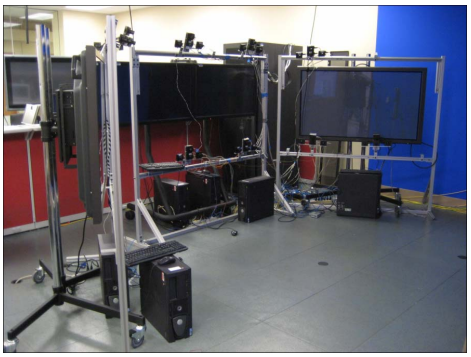


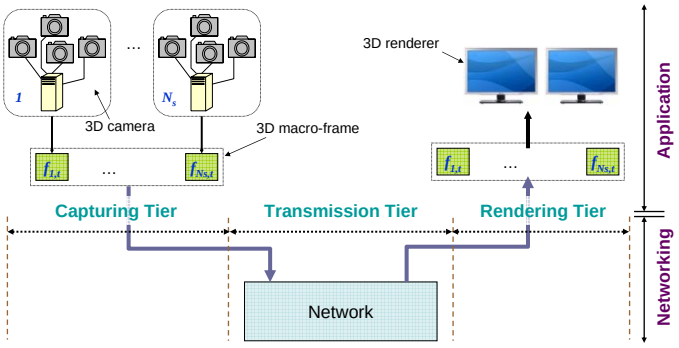
Figure 2.1: A 3D Camera Unit

and three black/white cameras at the bottom. The synchronization of cameras is achieved by connecting them via hotwires. The black and white cameras are used to compute depths by adopting a trinocular stereo algorithm ([36, 37]) and the color camera is used to extract appearance information from the corresponding viewpoint. Each 3D camera unit is linked to an edge computer via 1394b connection. The edge computer computes depth and color of each point on the reference image which is captured by the central black and white camera. The data returned by the 3D camera represents a 3D image frame which consists of a two-dimensional array of *XYZ-RGB* information (i.e., depth coordinates plus color bits) for each pixel. Each pixel requires 5 bytes to store XYZ (2 Bytes) and RGB (3 Bytes) information. Thus, assuming that w and h denote width and height of a 3D image frame (denoted as f) in pixel, the raw data size of one frame (denoted as fs) can be calculated as: $fs = w \times h \times resolution/pixel$. In our environment, each 3D camera cluster can reconstruct a 3D frame at the rate of 10 frames/second, and if we consider $w = 320$ pixels, $h = 240$ pixels, then one 3D video stream, uncompressed, demands 3,840,000 Bytes per second.

Integrated Data Model. The capturing tier consists of N_s equal 3D camera units mounted in various spatial viewpoints which form a 3D camera array (Figure 2.2(a)). All camera units are synchronized to capture at the same time instant. This means that at time t the capturing tier must have N_s 3D reconstructed frames, which constitute a *macro-frame*, to form a comprehensive 3D representation of the scene at time t. This macro-frame generated at time t is denoted as F_t, which consists of $\{f_t^1,\ f_t^2, ..., f_t^{N_s}\}$ single 3D frames from the individual camera units (Figure 2.2(b)). We denote f_t^i as the image frame captured by the ith 3D camera at time t. Hence, the tele-immersive application yields a stream of macro-frames $(F_{t_0},\ F_{t_1}, ..., F_{t_\infty})$ that are captured at the fixed synchronized rate (e.g., 10 macro-frames per second). The advantage of the 3D multi-camera array is the coverage of large field of view, which provides more freedom of view selection to the user. With accurate global calibration, the view of the scene can be rendered in a seamless fashion, offering a better quality than that of 2D multi-camera array.



(a) 3D Camera Array



(b) Integrated Data Model

Figure 2.2: 3DTI Data Model

2.1.2 View Model

There are many types of displays for rendering 3D scene ranging from stereoscopic displays to traditional 2D monitors. One important thing to note is that 3D video displaying is not a passive process. In order to render 3D objects with correct visual effect, the display device needs to acquire the user view information and render the 3D scene accordingly. The system

detects user view change using various tracking devices. In our 3DTI setting, the user is allowed to manipulate his view using keyboard and mouse.

One of the unique characteristics of 3D multi-camera array is the *stream correlation* in the sense that 3D cameras are concurrently capturing data with a synchronized clock and presenting complementary visual information of a common physical scene. As each video stream only covers part of the whole content, the importance of each stream according to what the user is currently watching is different. Therefore, it has become an interesting problem to design a multi-stream QoS adaptation based on the semantic link among different cameras, and the interaction between the user view and the orientation of cameras. Thus, we adopt a view model to formally represent those relationships.

To facilitate future discussion, here we extend the data model and assume there are N_e 3DTI environments. After 3D video reconstruction, the camera array of each environment produces a set of 3D video streams S_i where $1 \leq i \leq N_e$ and $(S_i = \{s_{i,1}, s_{i,2}, ..., s_{i,|S_i|}\})$ with each $s_{i,j}$ $(1 \leq j \leq |S_i|)$ corresponding to one stream of 3D video frames (i.e., $s_{i,j} = \{f_{t_0}^{i,j},\ f_{t_1}^{i,j}, ...\}$) generated in discrete time instants. We denote $f_t^{i,j}$ as the image frame captured by the jth 3D camera in the ith environment at time t. For convenience, we define the set of all streams as S with $S = \cup_{i=1}^{N_e} S_i$. Accordingly, we denote the macro-frame as F_t^i where $F_t^i = \{f_t^{i,1},\ f_t^{i,2}, ..., f_t^{i,|S_i|}\}$. Due to the one-to-one correspondency between 3D camera and stream, we will use these two terms interchangeably without causing any confusion. Each camera (or stream) has its own intrinsic and extrinsic parameters used for the 3D reconstruction and rendering. The extrinsic parameters indicate the camera orientation in a global coordinate system, which is denoted as the unit vector $s.\vec{w}$ for stream s. The set of camera orientations is denoted as W_i with $W_i = \{s_{i,1}.\vec{w}, s_{i,2}.\vec{w}, ..., s_{i,|S_i|}.\vec{w}\}$. Meanwhile, we assume there are N_r renderers and the set of renderers is denoted as U with $U = \{u_1, u_2, ..., u_{N_r}\}$. Each renderer has its user view information denoted as the unit vector $u_k.\vec{w}$ $(1 \leq k \leq N_r)$, which represents the current viewing direction of the user.[1]

[1]We need to point out that for the view model to be complete it should include not only the directional

At any given time, the edge computer running the 3D rendering program only handles one particular view of the 3D virtual space. When it receives a macro-frame F_t^i, it starts to process each 3D video frame within it (i.e., $f_t^{i,j}$). For a 3D video frame, the 3D coordinate of every pixel can be independently decoded because its (x, y, z) coordinate can be restored via its row and column index of the array, its depth information, and the camera parameters. If the global calibration is correct, the pixels of multiple streams rendered in the global 3D coordinate system will form into the shape of the 3D object.

The above concept of *rendering independency* provides a simple yet powerful mechanism to perform adaptation at the macro-frame or individual 3D frame level. For example, we could use a subset of the macro-frame, $F_t^{i'} \subset F_t^i$. Alternatively, we could use a subset of 3D video frame, $f_t^{i,j'} \subset f_t^{i,j}$ where not all pixels of the 3D video frame are transmitted and rendered. As shown later, the view model provides an important high-level basis for multi-stream coordination and QoS adaptation to achieve high visual quality under the resource constraints.

2.1.3 Timing Model

The timing model specifies the temporal requirement for the transmission of macro-frames as shown in Figure 2.3. It is important to realize that although the macro-frame F_t^i from one environment consists of $|S_i|$ reconstructed frames at time t, they cannot be transmitted at the same time due to the sharing of network path. Hence, the transmission of individual frames $f_t^{i,j}$ of the macro-frame F_t^i needs to be shaped within a certain completion time interval $T_{snd\_int}(F_t^i)$ and $T_{snd\_int}(F_t^i)$ should be less than the period of macro-frame. Furthermore, it is important for the rendering of the macro-frame at the receiver that the individual frames f_t^i arrive within a completion time interval $T_{rcv\_int}(F_t^i)$ which is either equal to $T_{snd\_int}(F_t^i)$ or $T_{rcv\_int}(F_t^i) - T_{snd\_int}(F_t^i) \leq \delta$, where δ is small. The receiving time of F_t^i is denoted as

information but also the depth of the view. However, in our real 3DTI application it is quite sufficient to only use the former since the current 3D collaborative space is relatively small.

$T_{rcv}(F_t^i)$. At the receiver, all frames $f_t^{i,j}$ will then be received and assembled together into the resulting macro-frame F_t^i and displayed at time $T_{disp}(F_t^i)$.

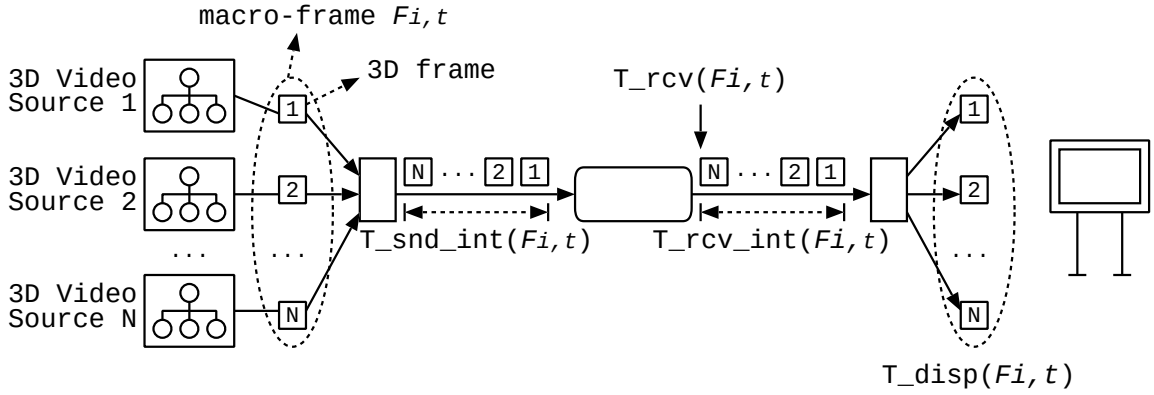


Figure 2.3: 3DTI Timing Model

2.2 Multi-stream Management Framework

To accommodate heterogeneous networking environments, the management framework adopts an integrated hybrid design for both end-to-end and multi-party 3DTI communication. In general, the management framework has three basic layers including *stream differentiation*, *stream coordination*, and *streaming control*. The framework is embedded as a service middleware in the overall 3DTI architecture between the application and networking layers as illustrated in Figure 2.4.

The task of application layer is to manipulate the raw 3D video and control the high-quality multi-camera/display environment. In the data plane, it performs real-time 3D reconstruction and rendering. In the control plane, it keeps track of the camera and user view information. The service middleware layer contains the core of multi-stream management framework and is composed of one or more *service gateways* (SG) which are connected with edge computers through high-speed LAN to perform view-oriented multi-stream differentiation, coordination and streaming control in the control plane and 3D video compression in the data plane. The service middleware layer is a distributed layer which is connected

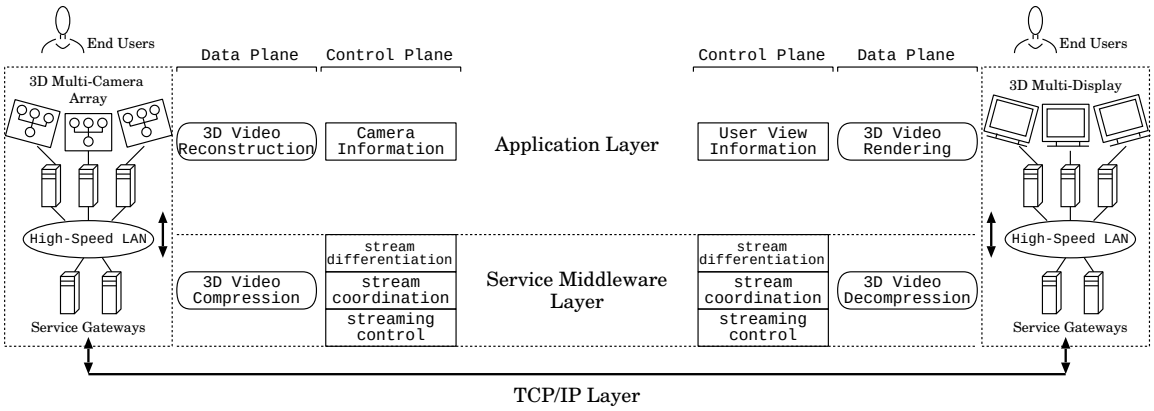


Figure 2.4: 3DTI Architecture

on top of the general TCP/IP layer. The basic layers of service middleware management framework are introduced below.

2.2.1 Stream Differentiation

The design of stream differentiation revolves around the concept of *view-awareness*. It is the front-end of multi-stream management to aggregate the information of the camera orientation and user view as described in the view model. The main task of stream differentiation layer is to calculate the *contribution factor*, denoted as $cf(s, u)$ with $s \in S$ and $u \in U$, as the indicator of stream importance regarding to the user view. The contribution factor provides a rough but very important criterion for stream coordination to filter out less important streams and prioritize important ones.

2.2.2 Stream Coordination

As introduced earlier, the stream coordination layer consists of two parts, *bandwidth allocation* and *ViewCast* for accommodating end-to-end and multi-party communications, respectively.

Bandwidth Allocation. In the case of end-to-end 3DTI communication, the content dissemination topology is relatively simpler. Such simplicity allows us to perform finer-

granularity multi-stream coordination. The bandwidth allocation utilizes the visual quality feedback to dynamically determine the bit budget for the most important streams. In the stream differentiation layer, the contribution factor provides a rough guidance for multi-stream coordination. For example, we could apply a priority-based scheme to allocate more bandwidth to more important streams. Such scheme works well under certain circumstances ([59]). However, the relationship between the contribution factor and the rendering quality is more complicated. When the available bandwidth becomes very low, balancing the bandwidth allocation among selected streams could achieve better quality. The main drawback of contribution factor is that it is a static criterion in terms of the user view. Thus, it cannot adjust to the change of the networking condition. For more active and accurate QoS control, we propose to apply the control based on the method of *reinforcement learning*.

ViewCast. For multi-party 3DTI communication we are no longer interested in stream-wise bandwidth allocation due to the trade-off between complexity and benefit. Instead, we are interested in designing more efficient dissemination protocols. As mentioned in Chapter 1, due to the huge data volume, it is impractical to take the unicast-based approach for connecting multiple environments. On the other hand, most available multicast schemes are stream-oriented which do not have the desired flexibility to accommodate the dynamics of stream correlation and view-based rendering as in the 3DTI communication. Thus, we propose an innovative *ViewCast* approach for more QoS adaptability by leveraging the high-level user view concept.

2.2.3 Streaming Control

The task of streaming control layer is to guarantee the temporal property of transmission as we introduce in the 3DTI timing model. For the case of end-to-end 3DTI communication, a *PID-based controller* is introduced which monitors the arrival time of macro-frame at the receiver end (i.e., $T_{rcv}(F_t^i)$). The receiver starts a periodic timer for rendering frames at fixed interval. Therefore, each macro-frame has a deadline that it must be received for rendering

on time. The difference between the arrival time and the deadline for each macro-frame is monitored and used as feedback to the PID controller to derive an appropriate *target macro-frame size* at the sender side. The case of multi-party 3DTI communication is more complicated. Currently, we assume the existence of an end-system overlay network upon which the ViewCast protocol is built. The overlay network will provide the information of link status (e.g., available bandwidth) to be used for streaming control.

2.3 3DTI Architecture

We abstract the distributed service middleware layer into an overlay network (Figure 2.5), $G = \langle V, E \rangle$, where each vertex v_i represents the service gateway (SG) in one 3DTI environment. There are several tasks performed in each vertex at the local and global scale.

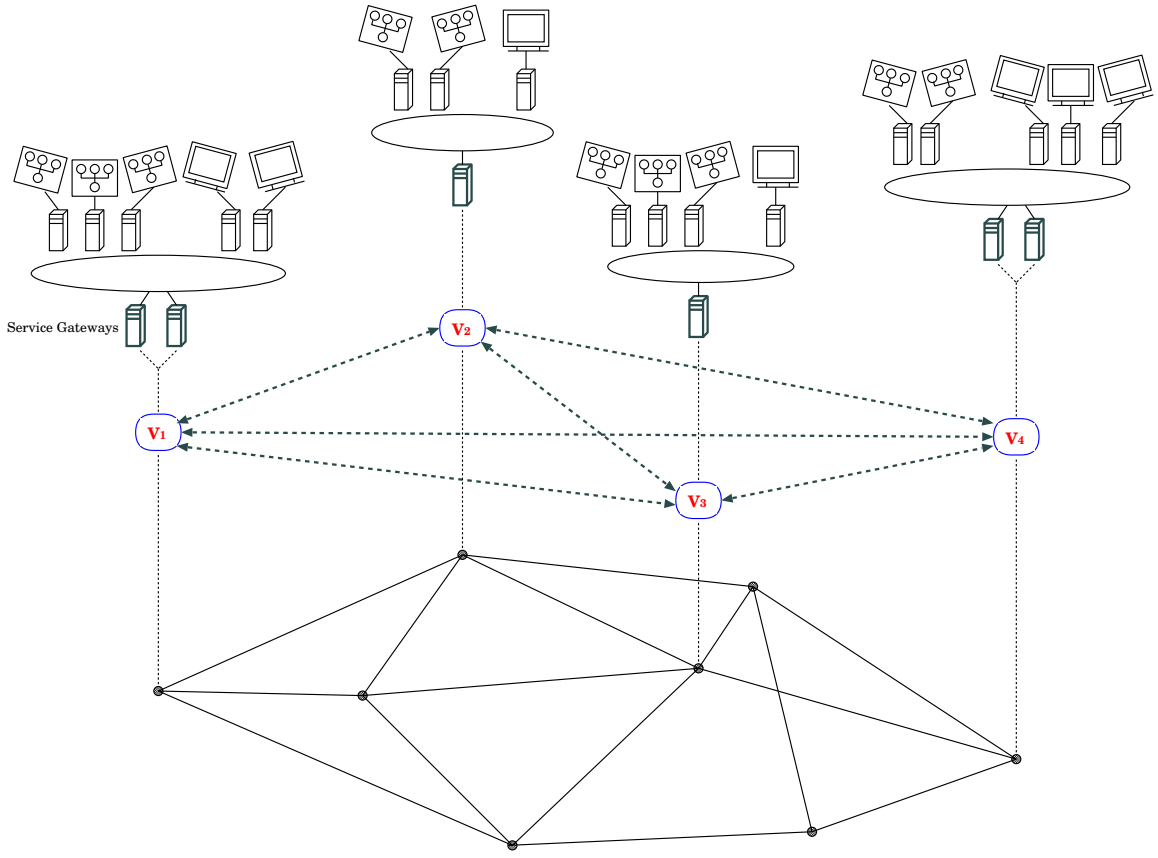


Figure 2.5: Multi-party Service Middleware Architecture

- **Local Management.** When the capturing and rendering tiers are initiated, they register with its local service gateway. Thus, each vertex needs to keep track of its own local resource information including the number of 3D cameras and displays, the parameters of the cameras and the user view information.

- **Overlay Maintenance.** The service gateways need to communicate with each other to maintain the functioning of the overlay structure during the 3DTI session. The information exchanged includes overlay network membership, link status, local resources and available streams.

- **View Dissemination.** The 3DTI view content is disseminated through the cooperative work among service gateways (or vertices). The view content is retrieved in a hierarchical manner. First, when the *user view* information is captured by the rendering tier, it decides whether the view request can be accommodated locally. If not, the request is forwarded to the local service gateway. The service gateway maps the view request to a set of relevant streams and checks whether they are available. If necessary, the service gateway either tries to obtain required streams from the source or request other service gateways to relay the streams based on the current status of view dissemination topology.

Chapter 3

Stream Differentiation

We discuss the stream differentiation problem in this chapter. The stream differentiation lies in the front end of the service middleware management framework. It captures the information of camera orientation and the user view to compute the function of contribution factor for each stream regarding to the user view. The contribution factor serves as a high-level guidance for multi-stream coordination.

3.1 Overview

The idea of stream differentiation is formulated to address issues of the bandwidth requirement and the rendering cost. As mentioned in Chapter 1, there are four major levels for the increase in the data volume, (1) the data rate of one video stream, (2) the number of video streams per 3DTI environment, (3) the number of environments, and (4) the number of renderers in one 3DTI session. First, each pixel of the 3D video frame carries two extra bytes for storing the depth information in addition to the three bytes RGB data used for the color information. Second, 3DTI environments require the installation of multiple 3D camera units in order to provide a wide view coverage and spatial definition of the scene. From the rendering prospective, when the scene is rendered in the 3D virtual space the user has more freedom to choose his view. As more video streams applied, the change from one view to another will become more seamless, delivering more enhanced immersive visual experience to the end user. However, the overall data rate in one 3DTI environment grows linearly with the number of streams deployed. Finally, when connecting multiple 3DTI environments and

multiple renderers the problem of overall data traffic becomes even more exacerbated as every rendering process needs to acquire the streams from all 3DTI environments.

The value of stream differentiation is highly associated with the property of rendering independency. That is, every pixel generated from the 3D reconstruction of camera array can be correctly rendered without the coexistence of any other pixel. Therefore, it is possible to tune the bandwidth budget at the pixel granularity according to the importance of streams.

There is one point to note here that some 2D multi-view video systems also apply multi-camera array (e.g., [34]). However, the nature of the problem is quite different. In such systems, the view change is accomplished via camera (or video stream) switching. When a user change his view, the system switches to the corresponding video stream. Compared with the 3DTI communication, the number of available views is rather fixed (according to the number of 2D cameras in use), the view change is not as smooth as in the 3D rendering, and the spatial interactivity is restricted due to the lack of 3D information. Although the 2D multi-view system may still need to deal with the multi-stream differentiation issue if the system is expected to tranfer multiple streams simultaneously, the problem would be much easier since there is a clear-cut in the view/stream relation. The situation in 3DTI video is more complicated as the rendering system merges multiple 3D video streams to display a single complete view in a common geometric space. As streams are correlated in the rendering quality of the final view, the problem becomes more complicated as how to differentiate the level of contribution of each stream.

3.2 Contribution Factor

The basic approach of stream differentiation is to derive the function of contribution factor, $cf(s, u)$, which indicates the importance of stream to the given user view. The applicability of stream differentiation is due to the fact that at any time instant the 3D renderer only displays one particular view of the virtual space. Because of the spatial distribution of cameras, the

relative importance of each camera and its correspondent stream must be different regarding to what is currently being rendered. However, the actual relation between streams and the rendering quality is very complicated. Therefore, we investigate the problem of calculating contribution factor based on two important aspects, (1) the *angular difference* and (2) the *viewing volume*.

3.2.1 Angular Difference

The usage of angular difference is based on an important observation that when a camera rotates away from the viewing direction of the user, its effective image resolution as projected onto the viewing plane will decrease due to the effect of foreshortening and occlusion. For example, if the user is currently looking at the front of a 3D object streams generated from side cameras are not as important as the front cameras. If we assume the object is not transparent (which is usually valid in 3DTI case), then the cameras from the back are the least important.

The effect of angular difference is illustrated in Figure 3.1, showing two cameras and the user view. The orientation of two cameras are denoted as $s_1.\vec{w}$ and $s_2.\vec{w}$ respectively. The orientation of user view is denoted as $u.\vec{w}$. As shown, the angular difference of $s_1.\vec{w}$ and $u.\vec{w}$ is larger than that of $s_2.\vec{w}$ and $u.\vec{w}$. As a direct consequence, the captured frame from Camera 2 is more similar to the rendered frame of the user view. In other words, Camera 2 has larger *contribution* to the rendered 3D scene and therefore is more important.

The angular difference can be calculated using the operation of vectors. For unit vectors, the dot product $(s_i.\vec{w} \cdot u.\vec{w})$ gives the value of $cos\theta_i$, where θ_i is the angle between $s_i.\vec{w}$ and $u.\vec{w}$. When the angular difference is small (i.e., θ close to $0\,^\circ$), then the value of dot product will become close to 1. Otherwise when θ is close to $180\,^\circ$, then the value of dot product will become close to -1. Based on that, we define the function of contribution factor $cf(s, u)$ for

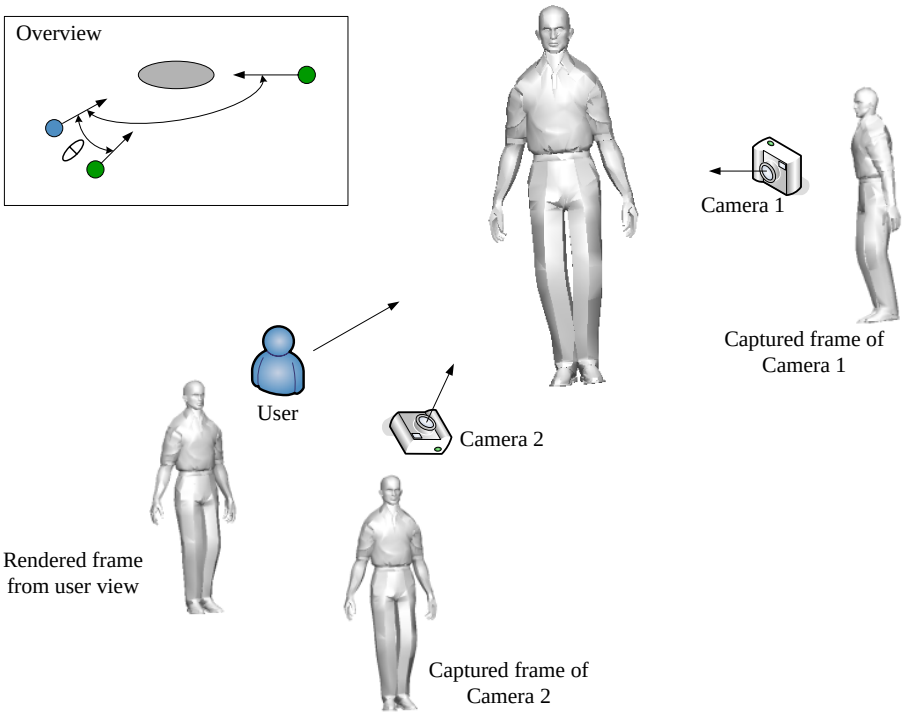


Figure 3.1: Effect of Angular Difference

$s \in S$ and $u \in U$ as in Equation (3.1).

$$cf(s,u) = s.\vec{w} \cdot u.\vec{w} \tag{3.1}$$

For practical concern, we use the normal of the camera image plane to represent the vector $s.\vec{w}$. The normal can be computed using the rotation matrix of the camera which is derived from the extrinsic parameters of the camera. Suppose the rotation matrix for camera i in the global coordinate system is r_i. Then the normal of the camera is computed using Equation 3.2.

$$s_i.\vec{w} = r_i(0,0,-1)^T \tag{3.2}$$

The vector $u.\vec{w}$ of user view is defined as the direction along the z-axis in the virtual space with $u.\vec{w} = (0,0,-1)$. Figure 3.2 shows the relationship of the normal of camera and the orientation of the user. In the figure, the coordinate system of camera is specified by

the axes of x_i, y_i, and z_i and the global system is specified by the axes of x_u, y_u and z_u.

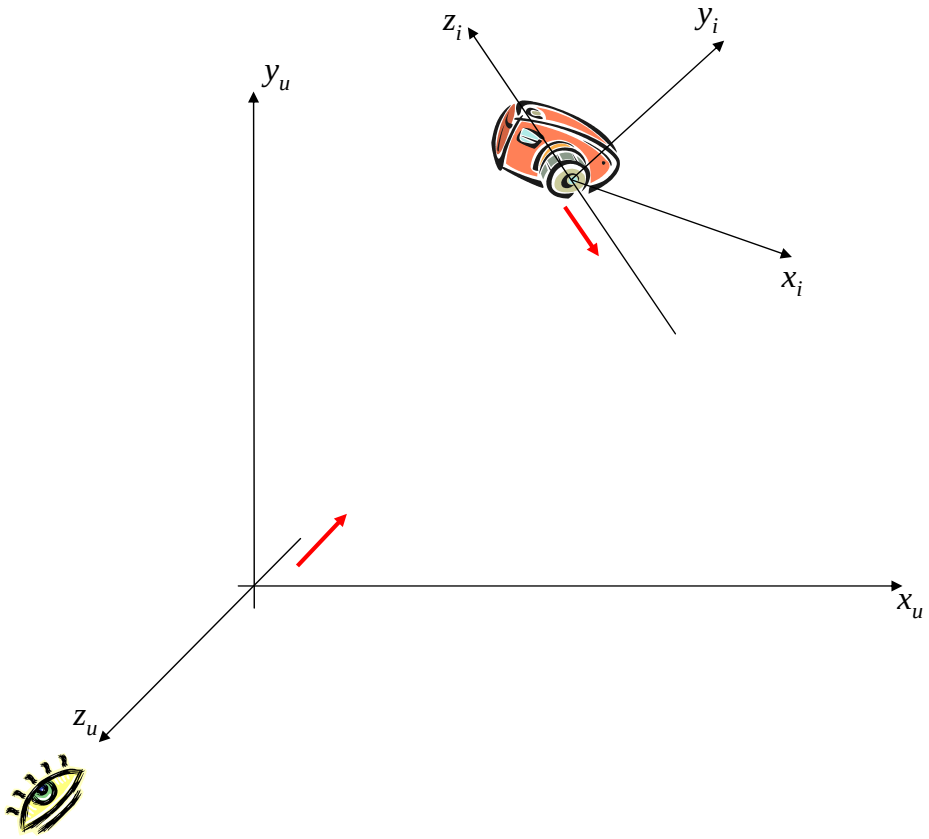


Figure 3.2: Relation between Camera and User View Orientations

3.2.2 Viewing Volume

The effect of viewing volume takes into account of view changes in addition to rotation. The viewing volume is a user-defined space within which objects are considered *visible* and rendered by the graphics system (a method known as *culling*). For example, the graphics library of OpenGL ([3]) provides a set of APIs for the user to define the viewing volume. Thus, we attach a new attribute of viewing volume to each renderer, denoted as $u.v$ for $u \in U$.

Figure 3.3 shows the effect of viewing volume. When the perspective projection mode is used, the viewing volume becomes a frustum. In the figure, the feet of the object are outside

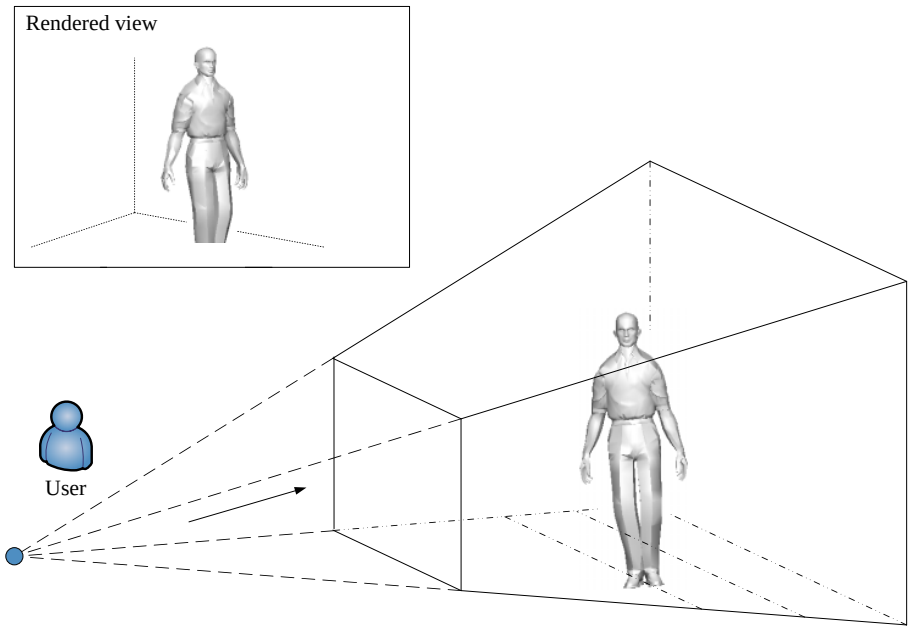


Figure 3.3: Effect of Viewing Volume

the viewing volume. Therefore, it is not rendered in the display.

For a pre-defined viewing volume at the renderer, the importance of each camera depends on how much of the captured 3D image lies inside the viewing volume. If most pixels of the 3D image are outside the viewing volume, then the camera is not important (i.e., has little contribution). Therefore, viewing volume is an important factor to evaluate the importance of camera.

It is relatively simple to compute whether a point is inside an enclosed space or not. Suppose the space is bounded by several planes. We define the normal of each plane to face towards the inside of the space. Then a point is inside the space if its distance to every plane is greater than or equal to zero.[1]

Based on that, we may compute the visibility of every pixel in a given 3D image. However, to reduce the computational cost, we divide the image frame into 16×16 blocks and choose

[1] A plane divides a space in two halves and the sign of distance depends on whether the point is in the same half space pointed by the normal of the plane or not.

the block center as the reference point. For each stream s, we compute its *visibility ratio*, denoted as $vr(s,u)$, which is the number of visible reference points versus the total number of reference points. Thus, we have another version of the contribution factor function as given in Equation 3.3 which combines the effect of angular difference with viewing volume.

$$cf(s,u) = vr(s,u) \times (s.\vec{w} \cdot u.\vec{w}) \tag{3.3}$$

In practice, we ignore the effect of viewing volume because the virtual space is relative small. Therefore, we only calculate the contribution factor based on the angular difference as in Equation (3.1).

3.3 Evaluation

We perform an experiment using a macro-frame captured in real 3DTI experiment. The macro-frame consists of 12 individual 3D image frames covering 360°. Figure 3.4 shows the color portion of the macro-frame with its individual frames.

Figure 3.4: Color Portion of a Macro-frame

Figure 3.5 shows the 3D rendering effect when all cameras are used, while Figure 3.6 only uses the cameras by choosing streams with contribution factor ≥ 0 (i.e., a maximum of 90° from the viewing direction). The visual quality of two settings has very little perceivable difference.

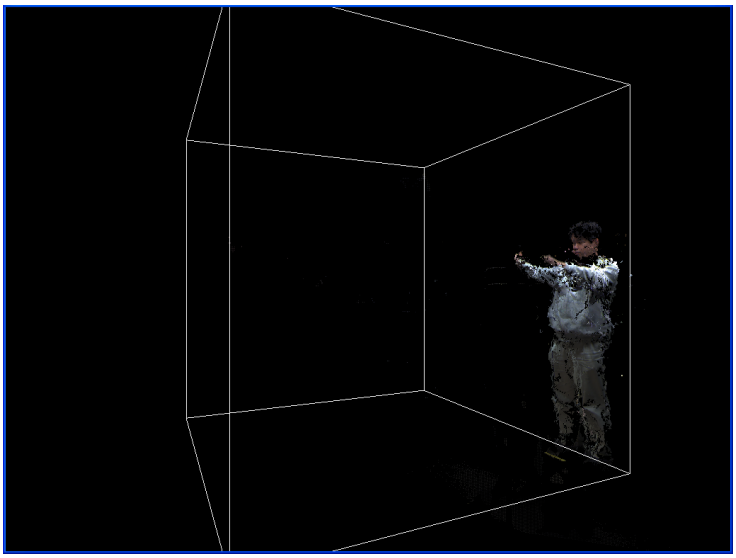


Figure 3.5: Rendering using all Cameras

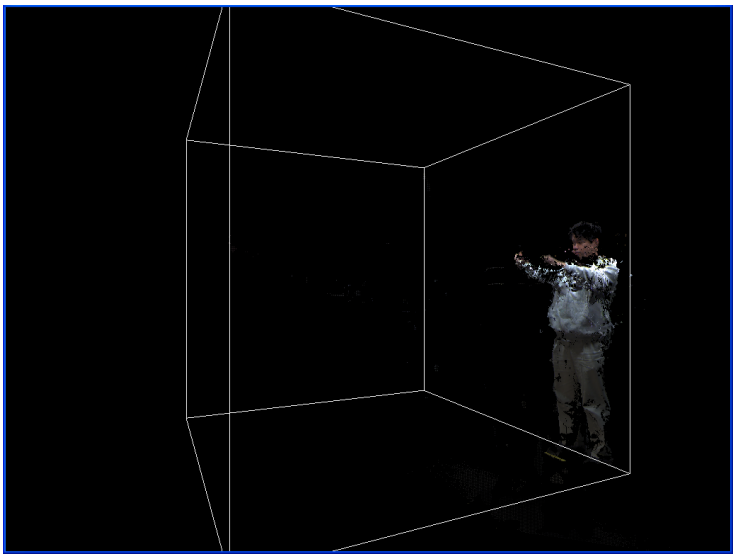


Figure 3.6: Rendering using selected Cameras

Chapter 4

Bandwidth Allocation

We discuss the bandwidth allocation problem in this chapter. The bandwidth allocation is the core of the stream coordination layer for the end-to-end 3DTI communication. The problem is how to allocate the bit budget for the transmission of streams that are considered as important.

4.1 Overview

So far, we have seen that one level of multi-stream adaptation can be achieved by dropping streams whose value of contribution factor is below certain threshold (e.g., < 0). Assuming cameras are uniformly distributed in space, this simple approach could reduce the required data rate by half without significantly affecting the visual quality as demonstrated in Chapter 3.

The problem remains as how to deal with important streams, particularly, if available bandwidth or rendering capacity is not sufficient to accommodate all important streams with full content. We propose three bandwidth allocation schemes as the solution. For the first scheme, we formulate the problem based on the general model of Markov decision process (MDP). We then apply the method of reinforcement learning upon the model to search for an optimal bandwidth allocation. For the second scheme, we use a priority-based method for bandwidth allocation. Under the priority-based scheme, streams with bigger value of contribution factor are assigned larger bit budget. For the last scheme, we use a non-priority method which allocates bit budget evenly among important streams. We

evaluate those three schemes in this chapter.

4.2 Reinforcement Learning

To help with the discussion, we briefly introduce the Markov decision process followed by the concept of reinforcement learning. More details of Markov decision process and reinforcement learning can be found in [13, 27]. After introduction, we describe how to apply the reinforcement learning method in solving the allocation problem through model mapping.

4.2.1 Markov Decision Process

Markov decision process is a discrete time stochastic control process for modeling the system (*environment*) influenced by probability and the action of the decision marker (*agent*). The markov decision process is illustrated in Figure 4.1. In the figure, the environment is represented by a set of states. In each state, the agent selects an *action* which changes the environment to the next state according to certain probability. When state transition happens, the agent gathers information about the immediate *reward*.

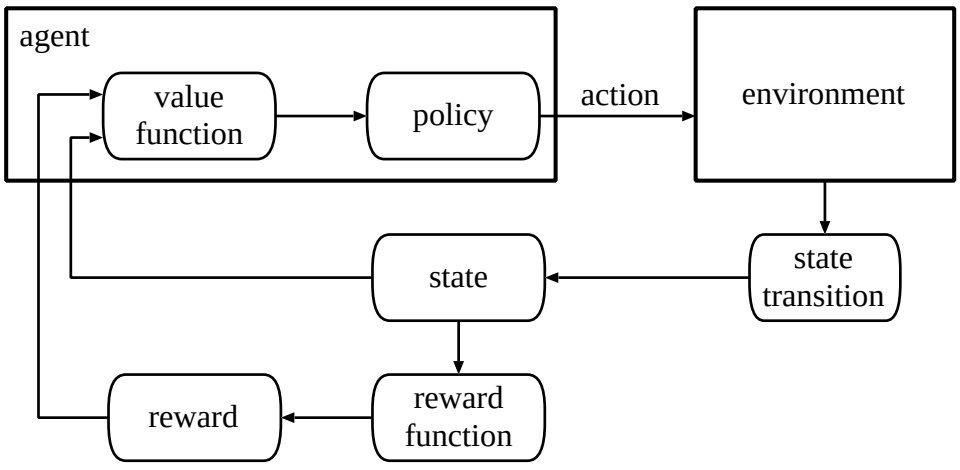


Figure 4.1: The Markov Decision Process

There are four basic components of a Markov decision process.

- *State Space*: $\Phi = \{\sigma_1, \sigma_2, ..., \sigma_n\}$.
- *Action Space*: $A = \{a_1, a_2, ..., a_m\}$.[1]
- *State Transition Probability*: $P_a(\sigma, \sigma')$ with $P_a(\sigma, \sigma') = Pr\{\sigma_{t+1} = \sigma' | \sigma_t = \sigma, a_t = a\}$ for $a \in A$ and $\sigma, \sigma' \in \Phi$, which indicates the probability of state transition from σ to σ' given the action a.
- *Reward Function*: $rf : \Phi \to \Re$, $rf(\sigma_t)$ indicates the immediate reward at the state σ_t.

The goal of the Markov decision process is to determine the action a_t at each state σ_t to maximize certain cumulative function of the reward, for example, the discounted sum as in Equation (4.1).

$$\sum_{t=0}^{\infty} \gamma^t rf(s_t) \tag{4.1}$$

where γ denotes the discount factor. The mapping from state to action is called the *policy* function, denoted as π with $\pi : \Phi \to A$. The optimal policy is denoted as π^*. The set of all policies is denoted as Π.

To derive π^*, we introduce the *value function* Ψ with $\Psi : \Pi \times \Phi \to \Re$. The value function is defined as in Equation (4.2).

$$\Psi^{\pi}(\sigma) = rf(\sigma) + \gamma \sum_{\sigma' \in \Phi} P_{\pi(\sigma)}(\sigma, \sigma') \Psi^{\pi}(\sigma') \tag{4.2}$$

The value function of the state is its immediate reward plus the expected discounted reward of the next state given that the action specified by the policy is taken. It indicates the desirability of a state not in its immediate reward but the long-term potential. Given the value function, the *optimal policy function* is defined as in Equation (4.3),

$$\pi^*(\sigma) = \arg\max_a \sum_{\sigma' \in \Phi} P_a(\sigma, \sigma') \Psi^{\pi^*}(\sigma') \tag{4.3}$$

[1]Usually, it is also defined $A_\sigma \subseteq A$ as the set of possible actions for state $\sigma \in \Phi$. We simplify the notation here.

namely, the optimal policy function chooses the action for a given state which will produce the maximum expected long-term reward.

If the system model is well established, the optimal policy can be solved using iterations of Equation (4.2) and Equation (4.3) based on dynamic programming.

4.2.2 Reinforcement Learning

Reinforcement learning is one way of deriving the optimal policy based on the Markov decision process when it is difficult to acquire system information such as the reward function or the probability of state transition. For example, in the situation of the bit budget allocation problem, the method based on reinforcement learning does not require a priori understanding of the model regarding to how each stream affects the rendering quality, a very complicated process in practice.

More specifically, we use the Q-learning algorithm ([54]) which is one of the most popular and effective algorithms for learning from delayed reward to determine the optimal policy. For policy π, we introduce the Q value function with $Q : \Pi \times \Phi \times A \rightarrow \Re$. The definition of Q value function is given in Equation (4.4).

$$Q^{\pi}(\sigma, a) = rf(\sigma) + \gamma \sum_{\sigma' \in \Phi} P_a(\sigma, \sigma') \Psi^{\pi}(\sigma') \tag{4.4}$$

Different from the value function Ψ, the Q value function indicates the goodness of state-action pair under certain policy π. Regarding the optimal policy π^*, the relation between the value function Ψ and the Q value function is given in Equation (4.5).

$$\Psi^{\pi^*}(\sigma) = \arg\max_{a \in A} Q^{\pi^*}(\sigma, a) \tag{4.5}$$

Thus, in terms of optimal policy we transform Equation (4.4) into the recursive form as in

Equation (4.6).

$$Q^{\pi^*}(\sigma, a) = rf(\sigma) + \gamma \sum_{\sigma' \in \Phi} P_a(\sigma, \sigma') \max_{a' \in A} Q^{\pi^*}(\sigma', a') \tag{4.6}$$

Based on that, the Q-learning algorithm updates the Q value function using Equation (4.7).

$$Q(\sigma_t, a_t) \leftarrow (1 - \alpha) Q(\sigma_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_{a \in A} Q(\sigma_{t+1}, a) \right) \tag{4.7}$$

where α denotes the learning rate. In practice, at each iterative step t the Q-algorithm chooses an action a_t according to the Q value function and the rule of ϵ-greed. That is, with probability ϵ we choose the action a_t of maximum $Q(\sigma_t, a_t)$ and with probability $1 - \epsilon$ we choose other actions randomly. The goal of ϵ-greedy algorithm is to balance exploitation and exploration. We then observe the next reward, denoted as r_{t+1} and the next state σ_{t+1} to update the Q value in $Q(\sigma_t, a_t)$ accordingly.

4.2.3 Model Mapping

The reinforcement learning method provides a general strategy of searching when the system information is incomplete. For application, the main task of model mapping is to define basic components including the state space, the action space and the reward function. The challenge is to keep the complexity of the model in bound. Since the bandwidth allocation scheme is only applied in the end-to-end case (i.e., $N_e = 1$ and $N_r = 1$), we simplify the notation here by ignoring the index of the environment.

We treat the multi-stream management as a discrete event system. An event occurs when it is ready to send the next macro-frame F_t with $F_t = \{f_t^1, f_t^2, ..., f_t^{|S|}\}$. Due to the property of rendering independency a 3D video stream can change its bit amount to adapt to the fluctuating resource availability. The bandwidth allocation is performed at discrete levels of frame size fs with a step size δ. Assume that there are K quantized levels where $K = \frac{fs}{\delta}$, each frame of the stream can choose frame size from the possible set of $\{fs_0, fs_1, ..., fs_K\}$

where $fs_0 = 0$, $fs_k = fs_{k-1} + \delta$ for $0 < k \leq K$, and $fs_K = fs$. We denote the bandwidth allocation vector as $\vec{C} = (c_1, c_2, ..., c_{|S|})$, where each component c_j specifies the frame size allocated for stream s_j $(1 \leq j \leq |S|)$. The problem is to find an optimal allocation vector $\vec{C}^*$ to optimize the visual quality under the constraint of available bandwidth bw as depicted in Figure 4.2.

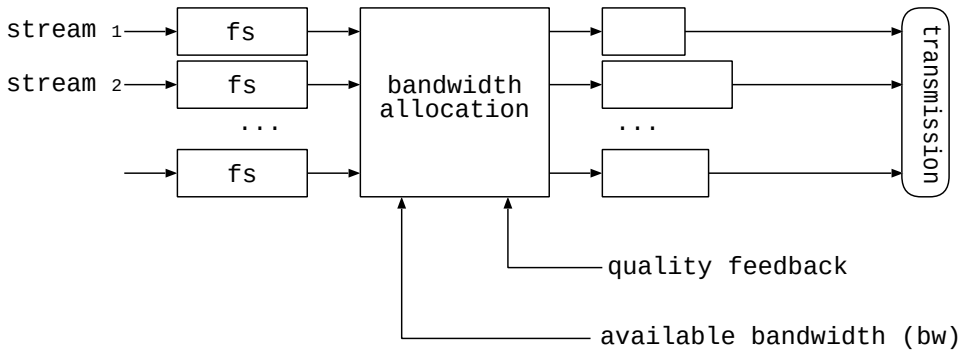


Figure 4.2: Stream Adaptation for Optimal Quality

Initial Design

State Space. We define the state space as in Equation (4.8). For simplicity, we define *target frame size* (TFS) as the maximum macro-frame size that can be transmitted and rendered given the current available bandwidth and the rendering capacity. The TFS is expressed in units of δ.

$$\Phi = \left\{ \sigma = (\vec{C}, TFS) \;\; | \;\; \sum_{j=1}^{|S|} c_j \leq TFS \right\} \tag{4.8}$$

Action Space. When the macro-frame is generated from the capturing tier, the decision agent chooses an action for bandwidth allocation. As an initial design, we define action a $(a \in A)$ as vector $\vec{a} = (a_1, a_2, ..., a_{|S|})$, where $-K \leq a_j \leq K$ for $1 \leq j \leq |S|$.

To illustrate, suppose the state σ has the form $(c_1, c_2, ..., c_{|S|}, TFS)$ and the next target frame size is TFS'. After applying the action, the next state σ' becomes

$$\sigma' = (c_1 + \delta a_1,\ c_2 + \delta a_2,\ ...,\ c_{|S|} + \delta a_{|S|},\ TFS')$$

where the action should be chosen such that the bandwidth constraint $\sum_{j=1}^{|S|}(c_j+\delta a_j) \leq TFS'$ still satisfies.

Reward Function. The reward function is related to the rendered visual quality. We use the peak signal-to-noise ratio (PSNR) of the reconstructed image measure as the reward. However, it is very difficult to derive the reward function directly from the allocation vector and user view. Therefore, we treat the rendering process as a black box which takes the video streams and the user view. Then the RGB portion of the rendered image is saved in 2D form and used for comparison. To get the PSNR measure, the 3D video is rendered twice, one with adaptation and one without. Denote the rendered RGB portion of the 3D image as I_{wa} and I_{wo} respectively. The PSNR is computed using Equation (4.9),

$$PSNR = 20 \times \log_{10}\left(\sqrt{\frac{\sum_{i=1}^{height}\sum_{j=1}^{width}[I_{wa}(i,j) - I_{wo}(i,j)]^2}{height \times width}}\right) \tag{4.9}$$

where *width* and *height* denote the dimension of the rendered image (which is different from the dimension of the 3D frame).

Refinement: Multiple Reinforcement Learning Machines

The definition of action space is general enough. However, the drawback is also obvious which lies in the size of the action space ($|A| = O(K^{|S|})$). As a solution, we introduce the concept of multiple reinforcement learning machines $RLM = \{RLM_{TFS_1}, RLM_{TFS_2}, ..., RLM_{TFS_H}\}$ (where H is the number of machines), with each machine corresponding to one particular target frame size (Figure 4.3).

Within one learning machine RLM_{TFS_j} $(1 \leq j \leq H)$, the target frame size TFS_j is assumed to be fixed. Given that, the state space for that machine can be defined as in Equation (4.2.3).

$$\Phi_{TFS_j} = \left\{\sigma = \vec{C} \;\middle|\; \sum_{k=1}^{|S|} c_k = TFS_j\right\}$$

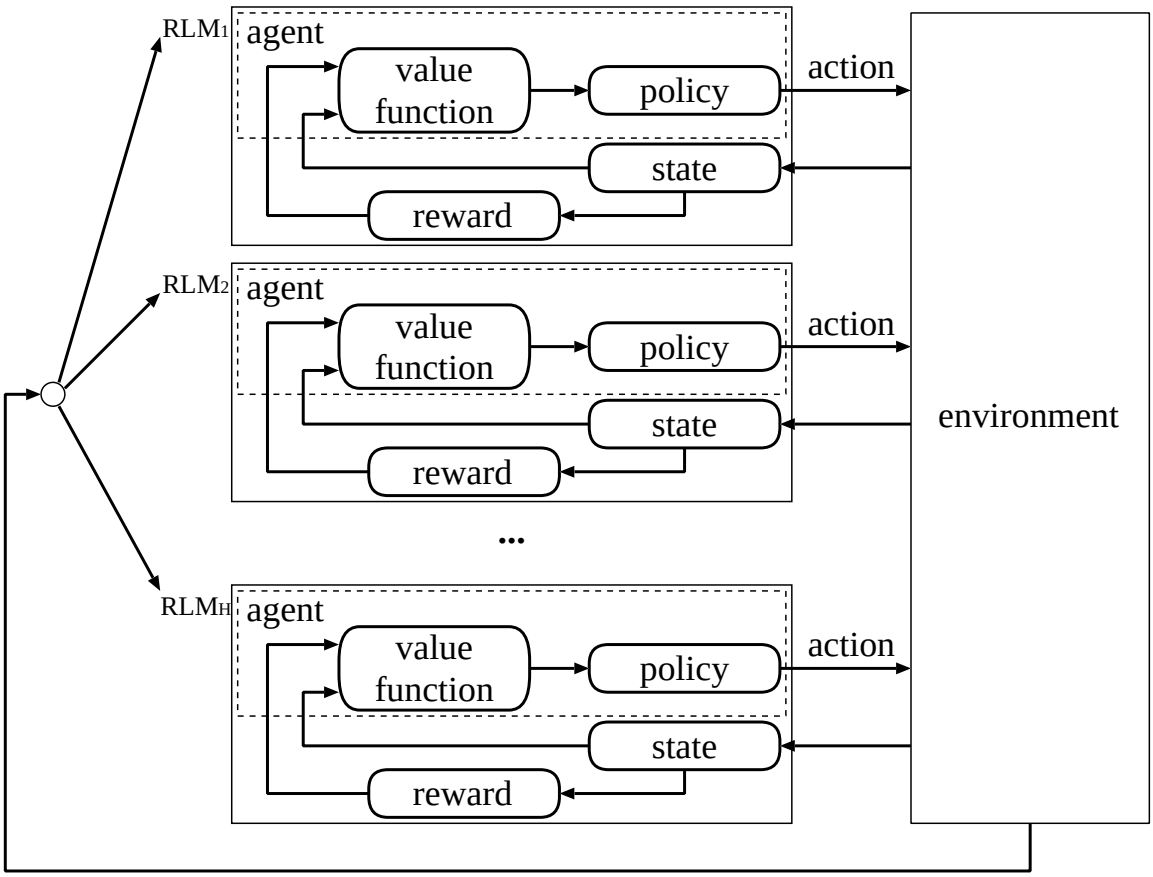


Figure 4.3: Multiple Reinforcement Learning Machines

Based on that, we define the action space A_{TFS_j} for each learning machine RLM_{TFS_j} as in Equation (4.2.3).

$$A_{TFS_j} = \left\{ a_{p,q} \;\; | \;\; c_p > 0 \wedge c_q < fs \right\}$$

The action will move one unit of allocation (i.e., δ) from stream p to stream q. To illustrate, suppose the state σ has the form $(c_1, c_2, ..., c_p, ..., c_q, ..., c_{|S|})$. After applying the action $a_{p,q}$, the next state σ' becomes

$$\sigma' = (c_1, c_2, ..., c_p - \delta, ..., c_q + \delta, ..., c_{|S|})$$

where the total macro-frame size (TFS_j) remains unchanged.

The introduction of multiple learning machines recognizes an important fact that when

the available resource remains relatively stable, it is really unnecessary to try actions that reduce the overall macro-frame size. Compared with the initial design, the size of action space is reduced to $O(H \times |S|^2)$. The size of state space remains unchanged. It is possible to further reduce the size of state space by changing the smallest unit of TFS from δ to fs. From the running time point of view, multiple RLM's do not incur more running time than single RLM. Actually, at any time only one version of RLM's is selected according to the availalbe bandwidth and updated.

4.3 Priority-based Scheme

The *priority-based scheme* performs bandwith allocation according to the value of contribution factor and the following principles.

1. Streams with larger contribution factor should have higher priority.

2. Whenever possible, a minimum frame size defined as $fs \times cf(s,u)$ should be granted.

3. Once 2. is satisfied, the priority should be given to cover a wider field of view.

To apply the priority-based scheme, we select a subset of streams S' with $S' = \{s|s \in S \wedge cf(s,u) \geq 0\}$ and let $m = |S'|$. We then sort streams in descending order of contribution factor for bandwidth allocation. First, if $TFS \geq fs \times \sum_{s \in S'} cf(s,u)$, the frame size c_j for stream s_j is allocated as in Equation (4.10).

$$c_j = min\left(fs, fs \times cf(s_j,u) + (TFS - \sum_{k=1}^{j-1} c_k) \times \frac{cf(s_j,u)}{\sum_{k=j}^{m} cf(s_k,u)}\right) \tag{4.10}$$

If the target frame size is big enough to accommodate the minimum frame size of all streams, then the minimum frame size is allocated. After that, the residue of TFS is allocated proportional to the contribution factor.

If $TFS < fs \times \sum_{s \in S'} cf(s,u)$, then we allocate minimum stream frame size in order of priority as in Equation (4.11).

$$c_j = min\left(fs \times cf(s_j,u),\ TFS - \sum_{k=1}^{j-1} c_k \right) \tag{4.11}$$

It is possible that some of the selected streams may not get the quota of transmission.

4.4 Non-priority Scheme

For the non-priority scheme, the target frame size TFS bandwidth is allocated evenly among selected streams as in Equation (4.12).

$$c_j = \begin{cases} TFS/m & \text{if } TFS < m \times fs \\ fs & \text{otherwise} \end{cases} \tag{4.12}$$

4.5 Evaluation

For evaluation, we use 12 3D video streams (320×240 resolution and 5 frames/second) pre-recorded from the multi-camera environment showing a person and his physical movement with a horizontal view of 360°. The renderer is implemented with the library of OpenGL. The experiments are performed on the local testbed, where we send video streams to the 3D renderer within the Gigabit LAN. The adaptation is configured to choose TFS between 8 fs and 1 fs. Meanwhile, we gradually rotate and shift the view during the experiment. The PSNR is calculated by comparing with the base case of full streaming (i.e., 12 streams each with 100% content). We also measure the rendering time using a Dell Precision 470 computer with 1 GByte memory running Windows. We first compare the performance of priority-based scheme and non-priority scheme. The average PSNR and rendering time are shown in Table 4.1. The average rendering time of 12 streams is 159.5 ms per macro-frame.

In the table, We combine the results of running time of both schemes as they are very similar.

TFS (fs)	Average PSNR		Time
	Priority	Non-Pri.	
7	39.75	39.38	93.83
6	36.85	33.31	84.63
5	34.46	31.48	68.36
4	31.79	30.02	55.87
3	30.15	28.98	43.82
2	27.71	27.66	31.89
1	26.42	26.91	22.59

Table 4.1: PSNR (dB) and Rendering Time (ms)

For the learning-based allocation scheme, we set $TFS = 3.6fs$ and run the experiment using the similar setting. The results are plotted in Figure 4.4.

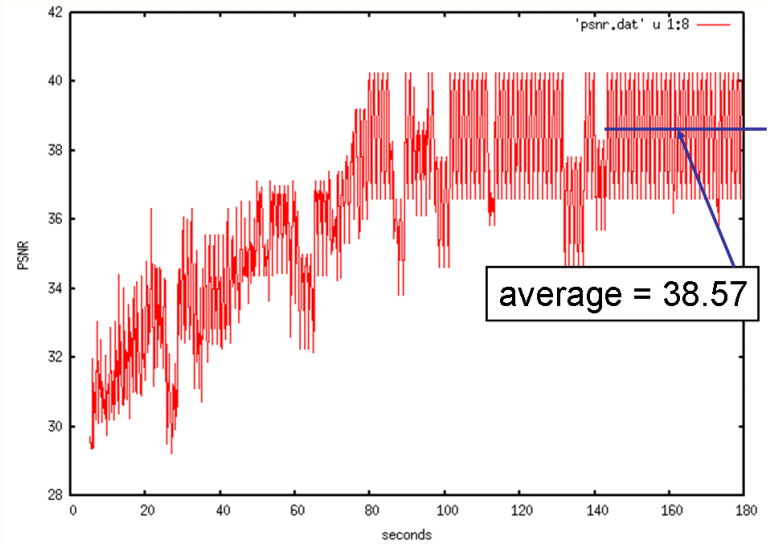


Figure 4.4: PSNR of Learning-based Scheme

Figure 4.4 shows that after 150 seconds the PSNR starts to settle down with an average PSNR of 38.57 which is much higher than priority-based and non-priority schemes.

Chapter 5

ViewCast

We introduce ViewCast in this chapter, which is a *view-based content dissemination scheme* used for stream coordination in the case of multi-party 3DTI communication. Compared with the case in end-to-end communication, the networking topology of multi-party communication is much more complicated. Therefore, as a trade-off the main focus of ViewCast is to optimize the transmission of multiple streams with full stream content instead of applying stream-level bandwidth allocation as in the end-to-end case. In this chapter, we first present an overview of ViewCast and the multi-party 3DTI session management for the ground of discussion. Then we introduce the basic problem of ViewCast followed by proposed solutions and the evaluation.

5.1 Overview

The view concept reflects the user interest at a higher-level, which distinguishes the ViewCast from any other content delivery schemes at the level of per stream. The ViewCast scheme basically specifies that when the user retrieves content from a multi-party/multi-stream system as in the case of 3DTI environments, only the user's view interest is required. The ViewCast scheme then controls the stream selection dynamically according to the view requirement and the status of resources with the ultimate goal of sustaining the QoS for the rendered view. Therefore, ViewCast has the advantages of improved flexibility, customization, adaptability, coordination and responsiveness under more dynamic and constrained resource environment. We envision the application of ViewCast in, for example, multi-

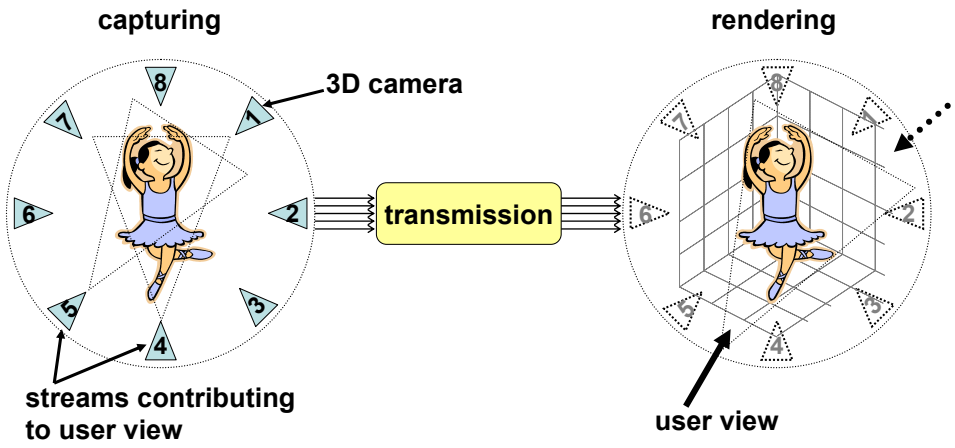


Figure 5.1: Stream Differentiation regarding to *View*

camera conferencing, surveillance system, 3D TV, and video sensor networks. From the 3DTI prospective, we point out several important properties of ViewCast as listed below. Most of the features may apply to other multi-party/multi-stream systems as well.

- *Multi-party/multi-stream Environment.* The basic assumption for ViewCast is that each content source supplies multiple correlated streams corresponding to one single view.

- *Stream Correlation.* Stream correlation is an important feature in multi-camera systems, where the concept of view has very intuitive definition.

- *Stream Differentiation.* Along with stream correlation is the feature of stream differentiation. That is, a given view should favor some of the streams over others. In 3DTI environment, the rendering process is view-dependent and the contribution of each stream to the rendering quality of the view could be different. As the angle of a 3D camera shifts away from the user view, its effective image resolution will decrease due to foreshortening and occlusion. As illustrated in Figure 5.1, given the user view (right part), cameras 4 and 5 are the most important ones. Cameras 3 and 6 are less important but will improve the visual quality if added. The rest cameras are the least important.

- *Inter-stream Coding Independency.* We assume the coding/decoding independency among streams. Since each stream can be independently transmitted and rendered, it is easier to perform the view to stream mapping and to select streams with different possible combinations. As illustrated in Figure 5.1, given the user view and the orientation of cameras ViewCast can select various subsets of streams (or cameras) such as $\{s_4\}$, $\{s_4, s_5\}$, $\{s_4, s_5, s_3\}$ or $\{s_4, s_5, s_3, s_6\}$... depending on the quality and resource constraints. Inter-stream coding independency provides more flexibility in stream selection and QoS adaptation. On the other hand, it also adds design challenge as there are more choices.

- *Open Model.* From an OSI layering point of view, the ViewCast scheme resides in the presentation layer. It maps between the semantics in the application layer (i.e., stream correlation/differentiation) and the stream manipulation in the session layer. More specifically, in 3DTI environments the ViewCast scheme only dictates desirable attributes of the application (e.g., the definition of view) and how these attributes affect the multi-streaming. However, the design choice of higher and lower layers is open depending on specific requirements. We are the first to identify a very core and ideal function in the presentation layer.

- *View Change.* As observed, the view change operation may occur frequently in 3DTI environments. There are two consequences of view change. First, stream differentiation varies with view change. For example, in Figure 5.1 when the user view changes to the position of dotted arrow (right part) cameras 1 and 2 will become the most important ones. This variance distinguishes ViewCast from other systems based on fixed stream differentiation such as layered coding and multi-description coding. Second, as soon as the view change is detected, the system must respond by switching streams accordingly. Stream switching may be costly. The direct impact to the user is the discontinuity of rendering. If multicast protocol is used in the underlying layer, then stream switching

at the parent vertex may influence the child vertices. The dynamics of view change presents a critical challenge for designing system based on ViewCast.

The key advantage of ViewCast is that by leveraging the high-level view semantics the visual quality, which is closely related to view, can be guaranteed while the low-level streaming regulation layer can have larger flexibility for topology construction and QoS adaptation so that the resource constraints can be satisfied adaptively.

5.2 Multi-party 3DTI Session Architecture

Figure 5.2 illustrates the 3DTI session architecture, which is managed at two levels. At the local level, each 3DTI environment is managed by its service gateway (SG), which consists of one or more processors. When a 3D camera initiates, it registers with the service gateway to save the meta-data of its stream. Due to the runtime cost of 3D reconstruction, once a 3D frame is generated it is forwarded to the service gateway through high-speed LAN for further processing of data compression and streaming control. The gateway manages rendering as well and retrieves streams on behalf of its local renderers. Thus, service gateway is an application level data aggregating point at each 3DTI site. We implement ViewCast on top of the end system multicast ([24]) which is becoming an appealing alternative to IP multicast due to the advantage of flexibility and easy deployment. At the application level, service gateways collect multiple streams either from local sources (i.e., cameras) or from remote sources (i.e., peer service gateways). The collected streams are then multicasted according to the current status of the overlay network and the forwarding topology managed by the session controller (explained next). Therefore, service gateways form an end system overlay network for content delivery.

After the bootstrapping of local environment is completed, the service gateway registers with the central session controller at the global level. The way session controller manages is similar to other proposed schemes (e.g., [23]). When a new service gateway joins the session,

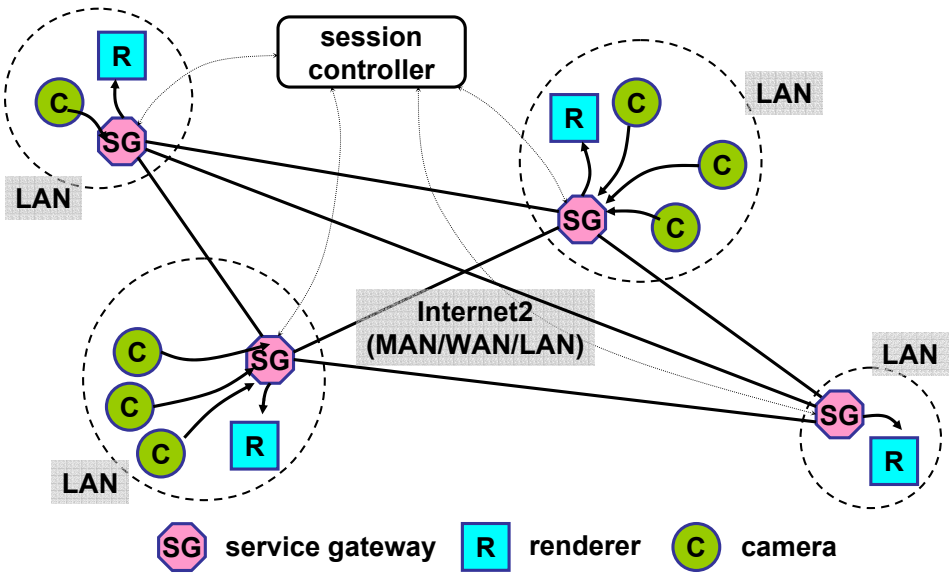


Figure 5.2: An overlay network of 3DTI session

the session controller informs other service gateways to let them connect with each other and form the initial overlay graph, $G = \langle V, E \rangle$. Then the session controller starts to receive and serve view requests for each vertex and update the delivery topology accordingly. For simplicity, it is assumed that all participating service gateways must register with the session controller before the live session can start.

During a live 3DTI session, the user can switch its viewing position of the virtual space at the renderer. If the view change cannot be resolved, the renderer will forward it to the service gateway. The service gateway checks whether it has the streams available for accommodating the view change. Otherwise, it sends a view request to the session controller to compute a new multicast topology for coordinating the multi-streaming.

We take a centralized approach at the global level because of its low messaging cost and responsiveness to the dynamics of 3DTI session. The approach is feasible in our situation where the number of service gateways is within a reasonable scale (≤ 20).

5.3 Multi-party 3DTI Session Initiation Protocol

The session protocol specifies in detail how the 3DTI session is initiated and managed. The steps of 3DTI session initialization are summarized below.

1. Start the session controller.
2. Start the service gateway. The service gateway registers its membership with the session controller.
3. The session controller updates the new service gateway with the membership of other service gateways which are currently active.
4. When 3D camera array is started, it registers with its local service gateway. The service gateway keeps the local resource information including the number of cameras and the orientation of each camera regarding to a global coordinate system. The resource information is sent to the session controller as well.
5. When the renderer is started, it contacts with the its local service gateway for sending view request.

After the session is started, the view request is served in the following steps.

1. The renderer captures the user view change and checks if it can be accommodated locally (i.e., if the view change is small).
2. If the view change cannot be accommodated, the renderer sends the request to its local service gateway.
3. The service gateway resolves the view request to streams and checks if it caches the relevant streams. If so, it serves the renderer by sending those streams.
4. In case needed, the service gateway forwards the view request to the session controller where the view dissemination topology is calculated globally.

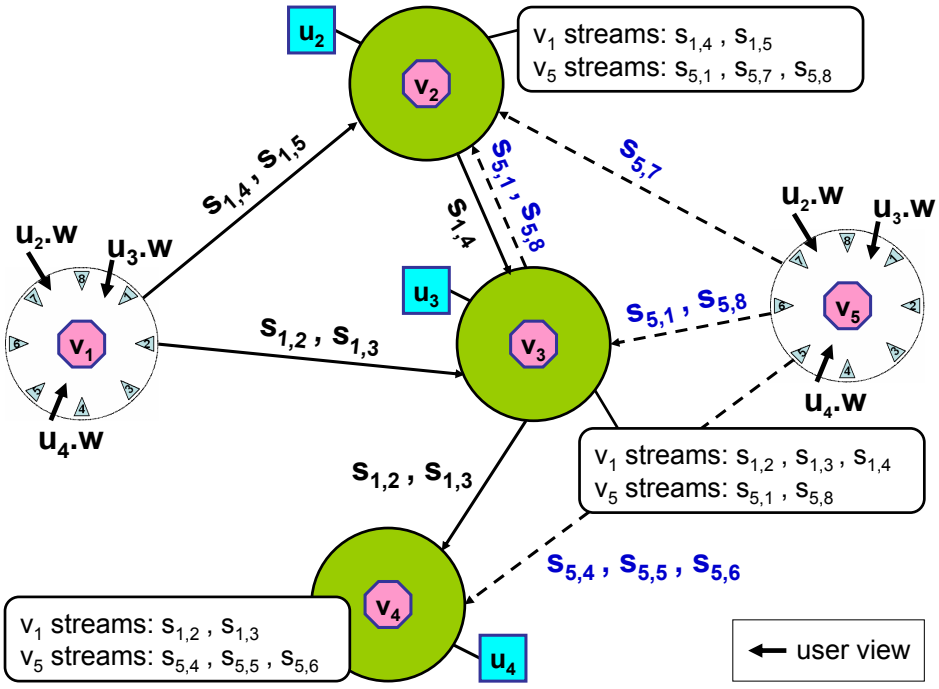


Figure 5.3: ViewCast streaming

5. After a new dissemination topology is calculated, the session controller broadcasts it to all active service gateways to complete the view request.

Figure 5.3 illustrates how ViewCast works. As the figure shows, each vertex represents a service gateway. Service gateways form an overlay network and cooperate for content delivery. In the figure, a vertex can request a view from a multi-stream source on behalf of its renderer. For example, renderer u_2 registers with vertex v_2 and v_2 requests a view (denoted as $u_2.w$) from vertex v_5. Depending on the view and available resources, each requesting vertex may get different subset of streams. As long as the quality and resource constraints are satisfied, vertices which have available streams can serve other vertices. Furthermore, a vertex can retrieve streams from multiple vertices in parallel.

5.4 Problem Formulation

We introduce the basic components and problems of ViewCast based on the initial graph of the overlay network $G = \langle V, E \rangle$.

5.4.1 Definition of ViewCast Components

Streams. In the multi-party/multi-stream system, a vertex v_i generates a set of streams. We denote S_i as the set of streams originated from vertex v_i (i.e., $S_i = \{s_{i,1}, s_{i,2}, ..., s_{i,|Si|}\}$). Each stream $s_{i,j}$ $(1 \leq j \leq |Si|)$ has extra field, denoted as $s_{i,j}.w$, which represents the view information of that stream. Note that, it is possible to have $S_i = \emptyset$ as the vertex may only serve rendering and viewing (i.e., no content generated at the vertex v_i). The complete stream space is denoted as S with $S = \cup_{\forall v_i \in V} S_i$.

Vertices: $V = \{v_1, v_2, ..., v_{N_e}\}$. There are N_e vertices. In 3DTI environments, a vertex v_i represents a service gateway which manages its local multiple streams. It can retrieve streams from other vertices as well by sending view requests. Vertex v_i is characterized by the following parameters.

- Inbound and outbound bandwidth constraints, denoted as I_i and O_i, respectively. For simplicity, we assume all streams have the same data rate. Then I_i and O_i are degrees measured as the number of streams v_i can receive and send. The inbound (and outbound) capacity is partitioned into *reserved* bins, where each bin hosts one 3D video stream.

- Set of inbound streams R_i where $R_i \subseteq S - S_i$. We denote $R_i(v_j)$ as the set of streams received which are originated from vertex v_j (i.e., $R_i(v_j) \subseteq S_j$).

- Set of outbound streams F_i where $F_i \subseteq R_i \cup S_i$. For stream $s \in R_i$, s is called a *relay stream*. Otherwise, stream $s \in S_i$ is called an *original stream*. We define a relay

function $rf(v_i, v_j, s)$. If vertex v_i transmits stream s to v_j, then $rf(v_i, v_j, s)$ is true. Otherwise it is false.

- Cost of stream ($cs : S \times V \rightarrow \Re$). Consider a stream s at vertex v_i (i.e., $s \in R_i \cup S_i$). If $s \in S_i$ then $cs(s, v_i) = 0$. Otherwise, there must exist a forwarding vertex v_j such that v_j transmits s to v_i. In that case, $cs(s, v_i) = cs(s, v_j) + ce(v_j, v_i)$, where $ce(v_j, v_i)$ denotes the cost of edge $\langle v_j, v_i \rangle$ (introduced below). The cost of stream reflects the delay from the source to the destination.

Edges: $E = \{\langle v_i, v_j \rangle | v_i \in V \wedge v_j \in V\}$. We define a cost function of edge $ce : E \rightarrow \Re$, which maps an edge to a real number. The cost function indicates the delay along the edge.

System Constraints. At any given time, the following conditions must always be satisfied for all vertices.

1. *Bandwidth Constraint*

$$\forall v_i \in V,\ |F_i| \leq O_i \wedge |R_i| \leq I_i \tag{5.1}$$

2. *Relay Constraint*

$$\forall v_i \in V,\ R_i \subseteq S - S_i \wedge F_i \subseteq R_i \cup S_i \wedge (\forall s \in R_i,\ \exists v_j \in V\ s.t.\ rf(v_j, v_i, s) = true) \tag{5.2}$$

3. *Delay Constraints*

$$\forall v_i \in V,\ max_{\forall s \in R_i}\left(cs(s, v_i)\right) \leq T_{delay} \tag{5.3}$$

where T_{delay} is the delay bound for interactive communication. We name conditions (5.1), (5.2) and (5.3) as the *system constraints*.

User. We denote the set of users as U with $U = \{u_1, u_2, ..., u_{N_r}\}$. Each user represents a renderer. The abstract concept of view reflects the user interest in retrieving the content.

We denote $u_i.w$ as the view information of the user. For simplicity, we assume each vertex can host one renderer. Therefore, we attach the view attribute to each vertex and use the notation of vertex $v_i.w$ instead, and use the set of V to represent U.

Differentiation Function. The differentiation function is denoted as df with $df : S \times V \rightarrow \Re$, which gives the importance of a stream regarding to a given user view. Depending on specific application, the definition of differentiation function could be different. For the case of 3DTI environments, the differentiation function is actually the function of contribution factor as described in Chapter 3, i.e., $df(s, v) = cf(s, v) = s.\vec{w} \cdot v.\vec{w}$.

Optimal Function. The optimal function is denoted as of with $of : 2^S \times V \rightarrow \Re$, which dictates the goal for optimization. For example, in 3DTI environments it could be the rendering quality of view regarding to the set of streams received. Since it is quite complicated to derive an exact form of optimal function as in our case. A simple linear approach is taken as in Equation (5.4).

$$of(S', v) = \sum_{s \in S'} df(s, v) \tag{5.4}$$

5.4.2 Problems of ViewCast

Given above notations and definitions, there is the maximum quality problem of ViewCast as formalized below.

Maximum Quality Problem

1. to maximize $\Sigma_{v_i \in V}\, of(R_i, v_i)$
2. subject to system constraints (5.1), (5.2) and (5.3)

The maximum quality problem is NP-complete, which can be proved based on the layered peer-to-peer streaming problem proposed in [16]. Another related problem of ViewCast is the minimum quality problem as given below.

Minimum Quality Problem

1. to satisfy $\forall v_i \in V, of(R_i, v_i) \geq \Delta$

2. subject to system constraints (5.1), (5.2) and (5.3)

where Δ is a given lowest bound on acceptable quality. The minimum quality problem is also NP-complete as shown by studies on finding minimum-cost degree-constrained multicast trees ([12, 43]).

In summary, we explain the basic idea of ViewCast under the 3DTI scenario. To simplify the explanation, certain technical restraints are imposed on the model. However, those restraints are not inherent in the general ViewCast concept. For example, it is not required that all streams have similar data rate or quality. The essence of ViewCast includes the definition of view and whether it can be used to differentiate streams.

5.5 Solution

There are two major goals for the ViewCast-based solution.

- *Minimum quality guarantee*: each vertex should receive a minimum set of streams to have some quality guarantee of every other vertex inside its view. For 3DTI environments, it implies the consistent presence of all participants in the virtual space, which is critical for collaborative work.

- *View change resilience*: when a vertex changes its view, the impact on other affected vertices should be minimized for the continuity of group interaction.

5.5.1 Minimum Quality Guarantee

Because the minimum quality problem is hard, we propose heuristics using the approach based on priority ([59]) and preemption ([58]) with the following steps.

Step 1. Given view request $v.w$, the importance of the stream is calculated using the differentiation function $df(s, v)$. In 3DTI environment, it is the same as the function of contribution factor.

Step 2. The streams are selected against a threshold, for example, in 3DTI environment we choose streams such that $df(s, v) \geq 0$, reflecting a 180° total view range.

Step 3. The selected streams are further differentiated into several priorities according to their importance. In 3DTI environments each vertex has around 8 streams. The stream selection in Step 2 produces a subset of 3 to 4 streams. We then define the set of priorities P as $\{p_1, p_2, p_3, p_4\}$, where p_4 is the highest priority. Next, we assign priorities to selected streams according to the differentiation function. In our case of 3DTI environments, the stream having the largest value of contribution factor is assigned the priority p_4 and so forth.

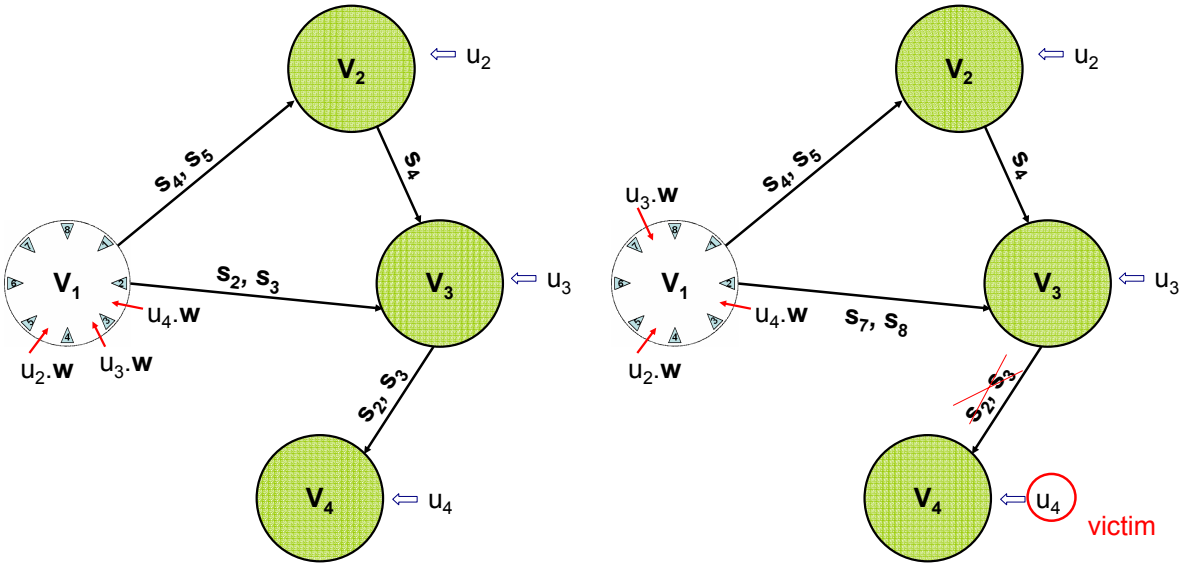
Step 4. As mentioned earlier, the inbound (and outbound) bandwidth resource is divided into bins with each bin hosting one stream. Suppose it is needed to forward stream s from vertex v_i to vertex v_j. If both vertices have available bins, it is straightforward to establish the streaming. Otherwise, the bin of lower priority stream can be preempted. For example, if stream s has p_4 priority based on the view, it can take the bin in either v_i or v_j occupied by streams of lower priority (i.e., $p_{1,2,3}$). When the preemption is needed, the bin of lowest priority stream will be taken first. The bin allocation of selected streams is performed in descending order of priority and terminated when the preemption is not possible.

Currently, we control the QoS of view rendering at the per stream granularity. When there is not enough resource to transmit a stream at its full content, the streaming will be dropped. However, it is an interesting problem to explore whether a quantitative improvement could be achieved at finer granularity (e.g., to transmit a stream with different data rate as in the end-to-end case) in ViewCast.

5.5.2 View Change Resilience

The negative impact of view change is illustrated in Figure 5.4. In the figure, vertex v_3 and v_4 have similar views and v_4 is streaming s_2 and s_3 from v_3. When vertex v_3 changes its view, streams needed by v_4 may temporarily become unavailable. In such case, v_4 becomes

a *victim*. The impact will grow as the size of dependent vertices increases. The view change operation is a frequent phenomenon in 3DTI environment and may cause large overhead if not treated properly.



(a) Before View Change (b) After View Change

Figure 5.4: Effect of View Change

Previous solutions rely on concepts of soft leave ([23]) and buffering ([16]). Soft leave requires the changing vertex to continuously serve old streams until affected vertices have found replacement. Although doable, under multi-stream scenario it would incur longer delay. Buffering let the intermediate vertices continue streaming from cache to absorb the propagation of quality degradation. However, it is not a feasible approach for live communication.

We apply the strategy of *dependency balancing* to improve the resilience of view change tolerance. There are three basic techniques in dependency balancing: (1) *source balancing*, (2) *priority balancing*, and (3) *load balancing*. We introduce them in the following.

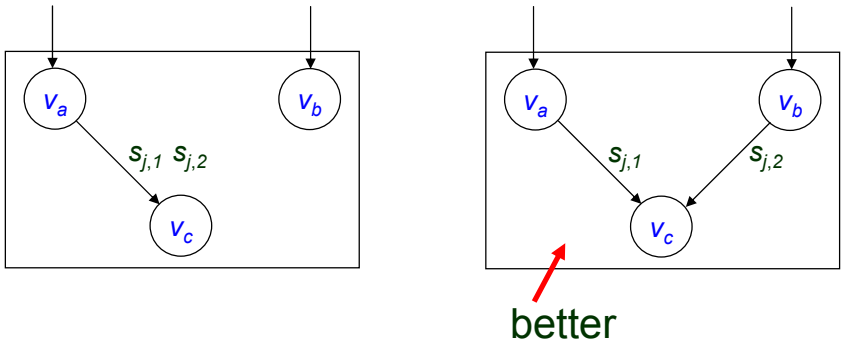


Figure 5.5: Source Balancing in ViewCast

Source Balancing

Source balancing attempts to diversify the supplying vertices to lower the dependency on each individual vertex. The basic idea is illustrated in Figure 5.5, where vertices v_a, v_b have the same set of streams (i.e., $\{s_{j,1}, s_{j,2}\}$) that can be relayed to vertex v_c. Under that, the streaming schedule on the right part is considered better since it provides a more evenly distribution of streaming among the sources.

Priority Balancing

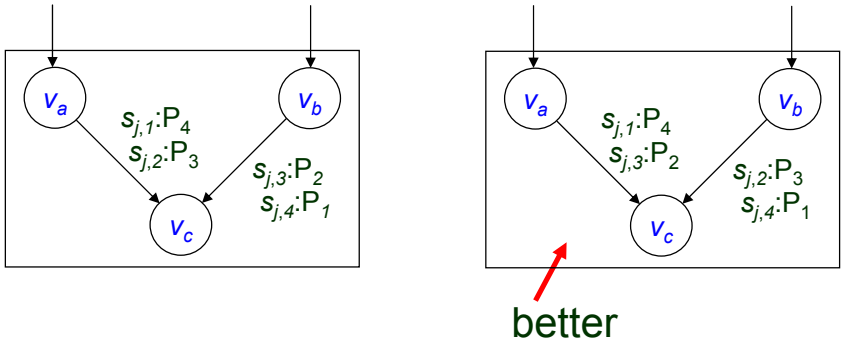


Figure 5.6: Priority Balancing in ViewCast

The basic idea of priority balancing is illustrated in Figure 5.6. In the figure, vertices v_a, v_b have the same set of streams (i.e., $\{s_{j,1}, s_{j,2}, s_{j,3}, s_{j,4}\}$) that can be relayed to vertex v_c. The stream priority of v_c as determined by its view is $s_{j,1} : p_4$, $s_{j,2} : p_3$, $s_{j,3} : p_2$ and $s_{j,4} : p_1$.

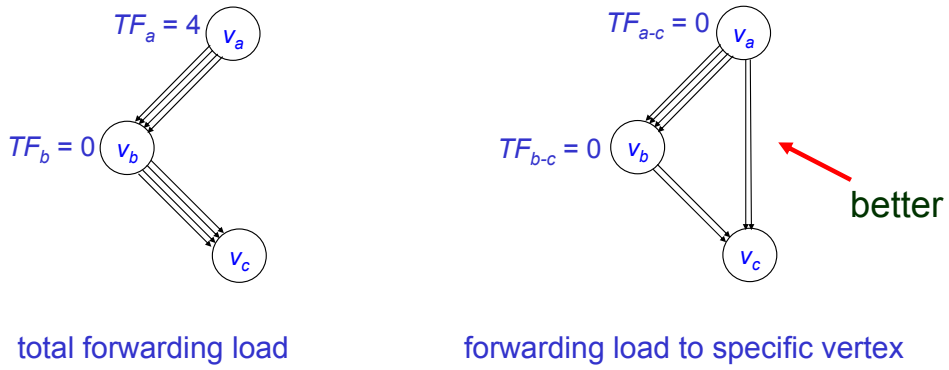


Figure 5.7: Load Balancing in ViewCast

Under that, the streaming schedule on the right part is considered better since it provides a more evenly distribution of streaming quality among the sources.

Load Balancing

Load balancing attempts to balance the forwarding load among all the vertices. There are two criteria in calculating the load. One is the *total forwarding load* as is often discussed in the literature (e.g., [14]). The other is the *forwarding load to specific vertex*. The ViewCast scheme considers the latter as more important. The justification is illustrated in Figure 5.7. In the figure, vertex v_b receives four streams from v_a. Next, v_c requests similar streams. Considering total forwarding load, v_b will be the forwarding vertex to v_c. However, that makes v_c solely depend on v_b. In the right figure, if we use the forwarding load to specific vertex as the criterion. Then both v_a and v_b can serve v_c which reduces forwarding dependency among the sources. Therefore, in ViewCast we choose the forwarding load to specific vertex as the first criterion over the total forwarding load.

Finally, possible techniques at the application level can also limit the overhead of view change effectively. For example, in 3DTI environment most of the view change occurs with small degree which can be tolerated by human perception. Thus, the rendering tier sends view request only when the view change is significantly big.

5.5.3 ViewCast Management

The main task of ViewCast management is to serve view request ($v_i.w$), which is performed by the session controller for each vertex (v_i). The main `serve_view` algorithm is sketched in Table 5.1.

The `get_streams` routine calculates the differentiation function $df(s, v_i)$ with the given view ($v_i.w$). The selected streams are assigned priority and saved in $\bar{S}$.

The `find_source` routine searches for a supplying vertex that can stream s to vertex v_i while obeying system constraints. It first scans the vertices which have available bins. Then it picks up the vertex that has the minimum forwarding load to vertex v_i for load and source balancing (break even with the minimum total forwarding load). If such vertex is not available, it looks for a vertex which has the bins that can be preempted. For priority balancing, each vertex maintains the sum of priorities (sp) for every other vertex. For example, if vertex v_i serves p_3 and p_2 streams for vertex v_j, then $sp_i(v_j)$ will be 5. The bigger this sum the higher the streaming quality that vertex v_i serves vertex v_j. Therefore, when there are several candidate vertices for relaying one stream to a destination vertex, the one with the smaller sum will be selected to achieve priority balancing.

The `find_out_bin` and `find_in_bin` routines are pretty straightforward. They return either an unused bin or a bin used by lower priority stream for preemption. For selecting an outbound bin to be preempted, we prefer to choose the lowest priority stream. For selecting an inbound bin to be preempted, one important consideration is to select a stream that is least used in forwarding to reduce the preemption cost, because once an inbound stream is preempted all child vertices that rely on it will not be streaming from the parent vertex any more.

The `serve_stream` maintains the bookkeeping of inbound and outbound bins of source and destination vertices. When preemption or view change is performed, the affected vertices are saved in the victim set. The `fix_victim` routine tries to fix the broken link. To reduce

Table 5.1: View Management Algorithm

```
serve_view(v_i)
    S̄ ← get_streams(S, v_i)
    for s ∈ S̄ in descending order of priority
        v ← find_source(s, v_i)
        if (v = null)
            report rejection
        end
        out ← find_out_bin(s, v, v_i)
        in ← find_in_bin(s, v, v_i)
        if (out = null or in = null)
            report rejection
            return
        end
        serve_stream(s, v, out, v_i, in)
    end
    fix_victim()
end

find_source(s, dst)
    for each v ∈ V other than dst
        if (v can stream s to dst under system constraints)
            if (v has extra outbound bins)
                V_1 ← V_1 ∪ v
            else
                V_2 ← V_2 ∪ v
            end
        end
    end
    if (V_1 ≠ ∅)
        return v ∈ V_1 where v has the least forwarding to dst
    else
        return v ∈ V_2 where v has preemptable bins
    end
end
```

the cost, this routine only fixes the broken link related to higher priority streams (e.g., p_4 and p_3). Otherwise, the affected vertex will simply ignore the lost stream and propagate the message to its child vertices. The propagation will terminate either when the preempted stream is not important for all child vertices or some child vertices have found new sources.

After the `serve_view` routine is completed, the session controller calculates the new topology and broadcasted to all vertices (i.e., service gateways).

5.6 Evaluation

We evaluate ViewCast in simulated 3DTI session. In this chapter, we introduce the experimental setup for the evaluation and then report the experimental results, including rejection ratio, overhead and QoS provisioning.

5.6.1 Experiment Setup

We implement a message-driven ViewCast simulator using C++ (on Windows and Linux operating systems). The simulator first generates application-level overlay network. We use mesh topology for the overlay network, where the connectivity between vertices follows the uniform distribution. The total number of vertices, defined as the *session size*, ranges from 5 to 10. The total number of edges is determined by the *connectivity ratio* (CR) which is the ratio of edges compared with the corresponding complete directed graph. For example, a directed graph of 8 vertices will have 42 edges, if the connectivity ratio is set to 75% (i.e., $56 \times 75\%$). We choose the connectivity ratio from 25% to 100%.

In the experiment, each vertex has 8 original streams which are evenly distributed in $360°$. For any view request to a vertex, at most 4 of its original streams are selected for an optimal coverage of $180°$. To determine the range of degree bound, in the maximal case of 10 vertices each vertex requires at least an in-degree ≥ 36 to get the optimal coverage from every other vertex and an out-degree ≥ 8 so that all its streams may be accessible for

serving any view. The estimation has not considered relaying overhead. For simplicity, we set the in-degree bound to be the same as the out-degree bound. The range of *degree bound* (DegB) is chosen between 12 to 36.

During the simulated 3DTI session, each vertex sends view change request to the central session controller. The interval of view change follows the normal distribution with a mean of 60 seconds. We use two patterns of view change: random walk and Zipf. In the random walk pattern, each vertex adds a view change degree to its current view. The view change degree follows the normal distribution with a standard deviation of 20°. In the Zipf pattern, the view is changed according to a Zipf distribution of 10 pre-selected view degrees (i.e., $n = 10$). The Zipf distribution is actually the power-law distribution in discrete form. In our 3DTI simulation, it dictates that the ith most popular view degree has the access frequency in proportion to ω/i^α, where α is a constant and ω is determined by n. The total running time of one simulation experiment is 200 minutes.

The propogation delay along each edge follows the normal distribution with a mean of 50 ms. We assume each stream has the same bandwidth. Since our main goal of simulation experiment is to study the Viewcast overhead and the impact of view change, the transmission delay is of less interest. The simulation parameters are summarized in Table 5.2.

Table 5.2: Simulation Parameters

overlay topology	mesh topology
number of vertices	5, 6, ..., 10
connectivity ratio (CR)	25%, 50%, 75%, 100%
in-degree (out-degree) bound (DegB)	12, 24, 36
number of streams per vertex	8
number of streams for optimal view coverage	4
simulation time	200 minutes
view change interval	normal distribution ($\mu = 60$ seconds)
view change pattern	(1) random walk with view change degree in normal distribution ($\mu = 0^\circ$, $\sigma = 20^\circ$), (2) Zipf ($n = 10$, $\alpha = 1.0$)
propogation delay of each edge	normal distribution ($\mu = 50$ ms)

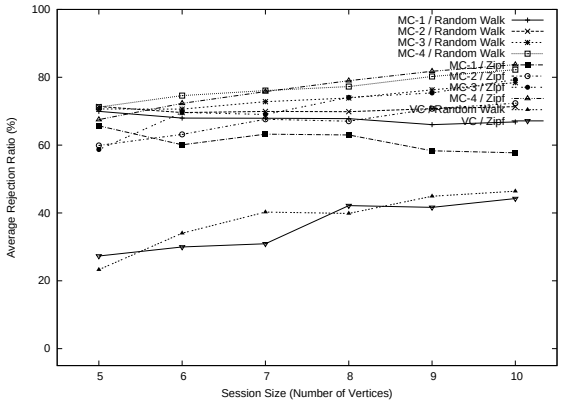
5.6.2 Rejection Ratio

We measure the *rejection ratio* of view change request. Recall that in ViewCast, the view change request is served with relevant streams in descending order of importance and the request is *rejected* if and only if there is no resource available to supply any stream. For comparison, we introduce another method of view dissemination based on multicast, where a view request is served by multicasting its relevant streams. However, the application needs to explicitly specify the set of streams needed for a view and the system must supply all the streams. Otherwise the request will be rejected. We use the notation of MC-n ($1 \leq n \leq 4$) to refer to the multicast-based method where the application always specifies a set of n streams to satisfy a view request. For example, MC-3 means that the system must deliver 3 streams to the application otherwise the view request will be rejected. The results of rejection ratio are plotted in Figure 5.8 and Figure 5.9. We use VC to denote the ViewCast method. Other notations are explained in Section 5.6.1.

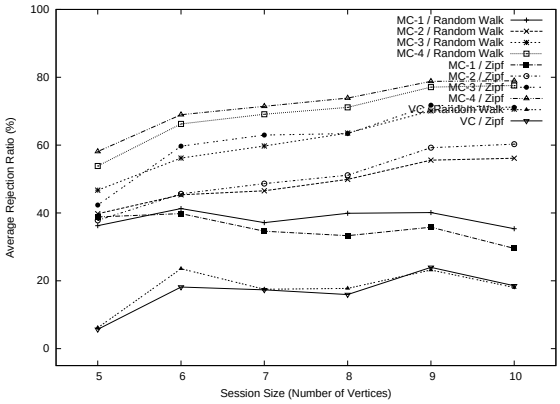
From Figure 5.8 and 5.9, we observe that the connectivity ratio has a very strong impact on the view dissemination capacity of the overlay network. In the most constrained cases (e.g., Figure 5.8(a,c,e)) where CR $\leq 25\%$, the rejection ratio is pretty high in either ViewCast or multicast-based methods. However, when the session size increases (≥ 8) ViewCast starts to perform much better. There is a clear-cut in performance at the medium region of CR $= 50\%$. When both the connectivity ratio and the degree bound are improved (CR ≥ 75, DegB ≥ 24), the rejection ratio of ViewCast becomes almost zero. The rejection ratio of multicast-based method drops as well. However, we show later that even in those cases ViewCast still performs better.

5.6.3 Streams Per View

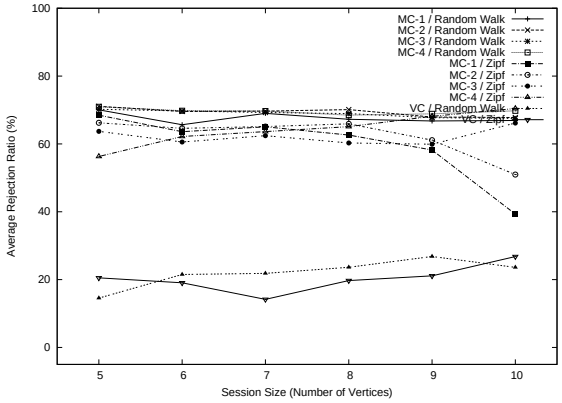
We measure the average number of streams for all successfully served view requests in ViewCast. The results of streams per view are plotted in Figure 5.10 and Figure 5.11.



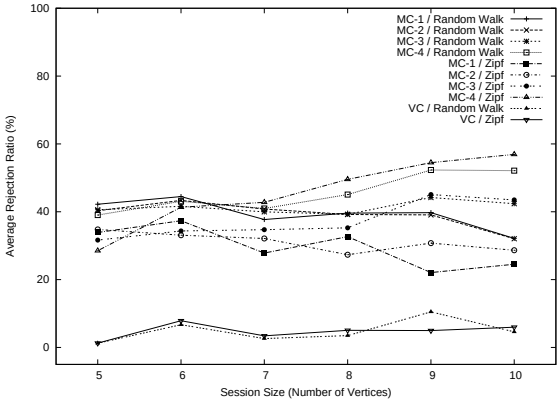
(a) CR = 25%, DegB = 12



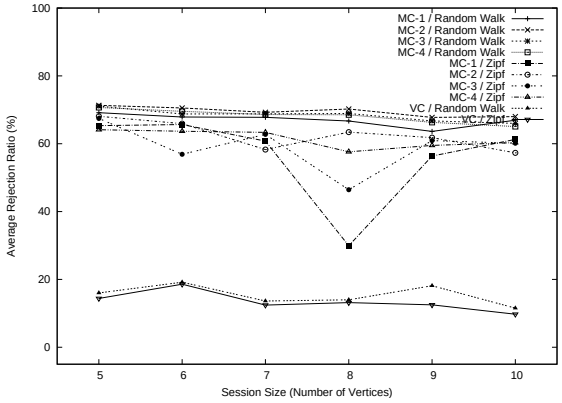
(b) CR = 50%, DegB = 12



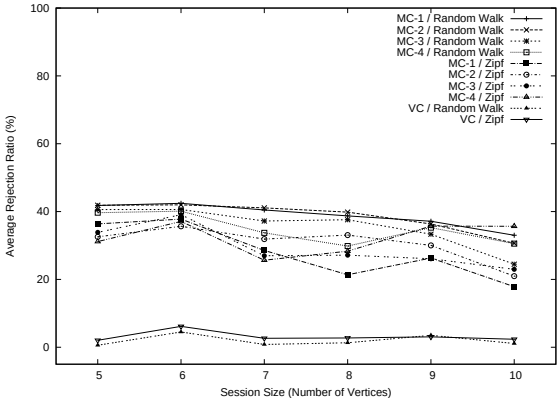
(c) CR = 25%, DegB = 24



(d) CR = 50%, DegB = 24



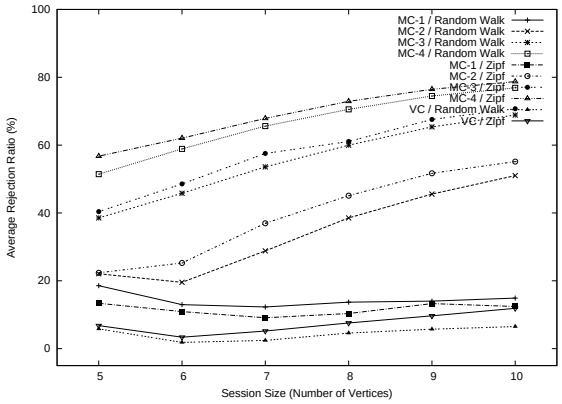
(e) CR = 25%, DegB = 36



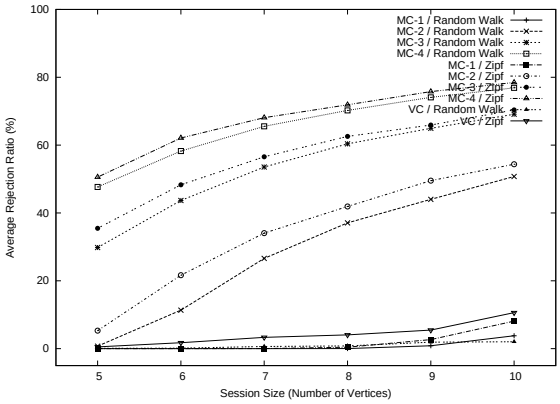
(f) CR = 50%, DegB = 36

Figure 5.8: Average Rejection Ratio

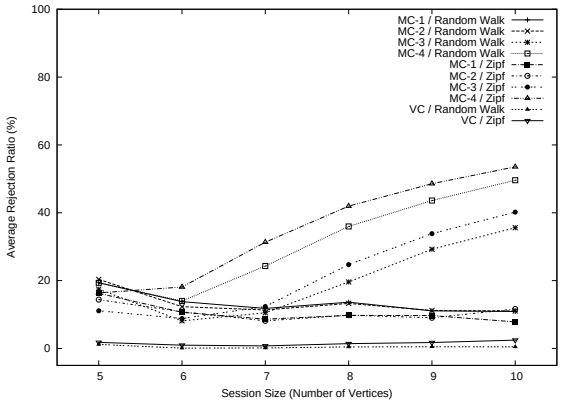
Those figures indicate that ViewCast serves each view request with an average of 2.5 to 3 streams. This number is close to what could be achieved by MC-2 and MC-3. However, if we examine Figure 5.10-5.11 along with Figure 5.8-5.9 and compare ViewCast with MC-2



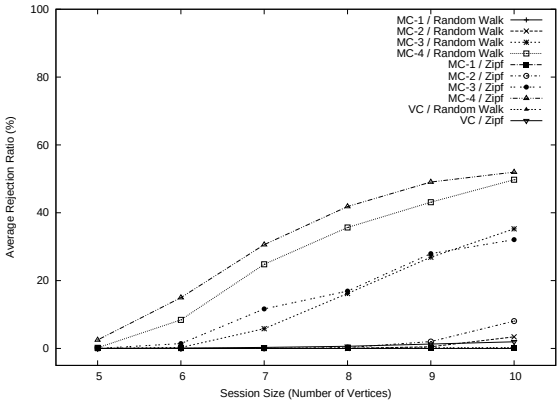
(g) CR = 75%, DegB = 12



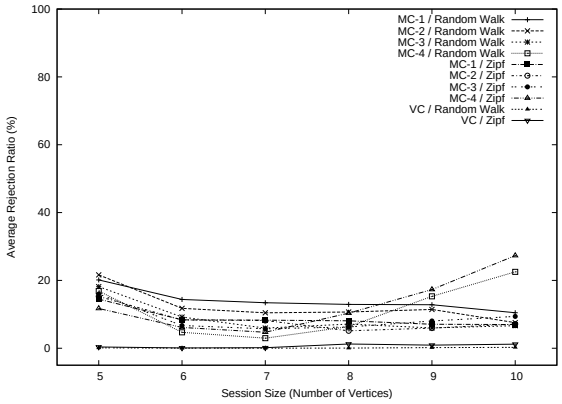
(h) CR = 100%, DegB = 12



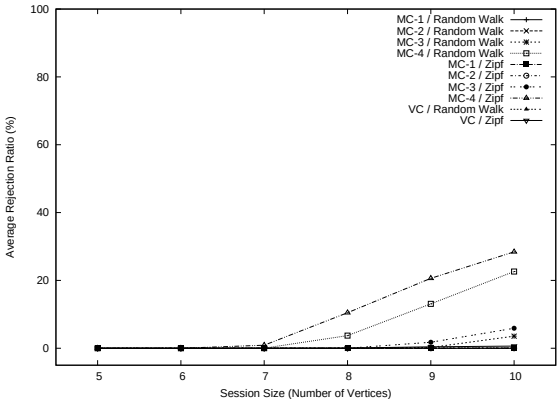
(i) CR = 75%, DegB = 24



(j) CR = 100%, DegB = 24



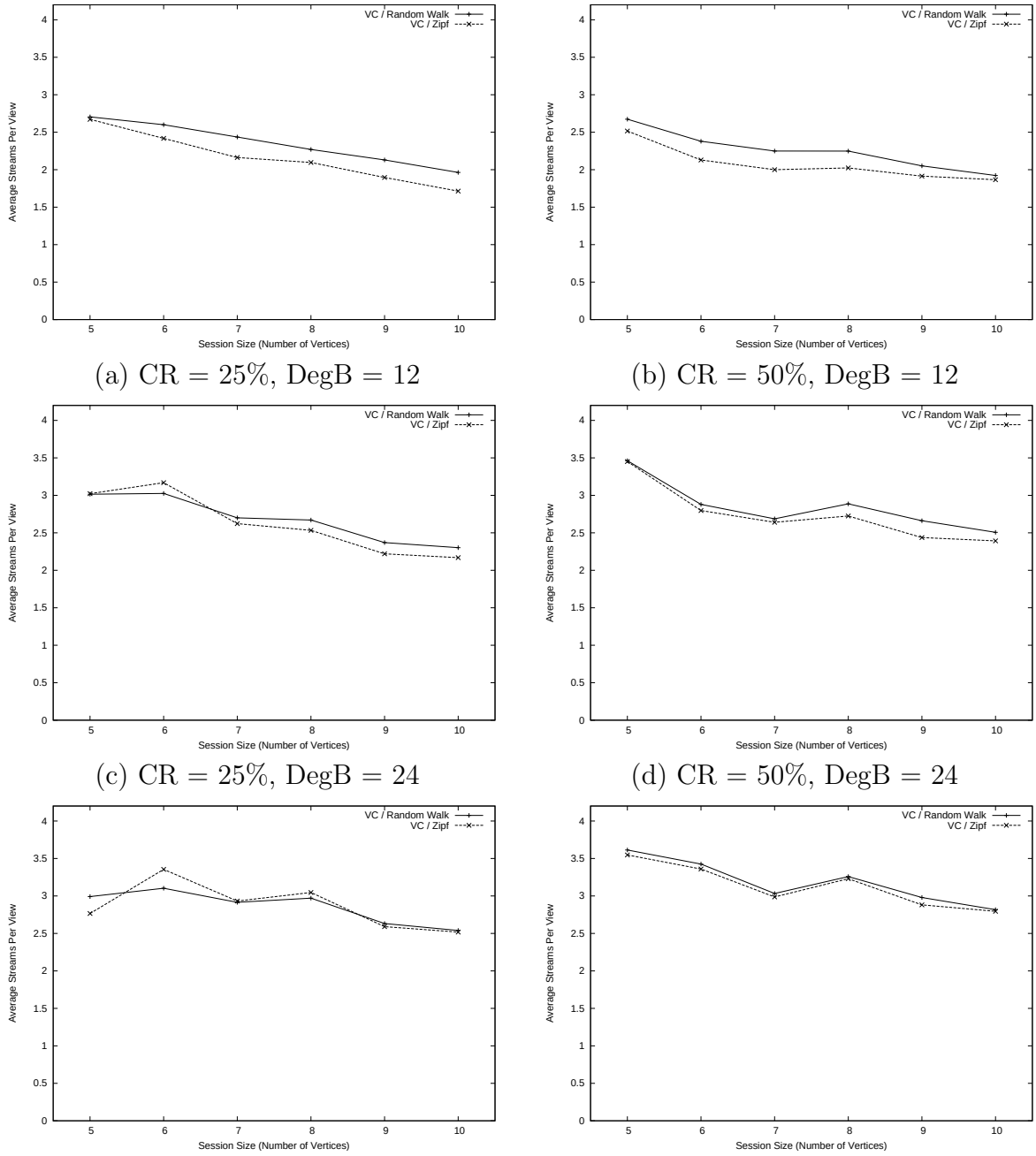
(k) CR = 75%, DegB = 36



(l) CR = 100%, DegB = 36

Figure 5.9: Average Rejection Ratio (*continued*)

and MC-3, we will see that ViewCast achieve much lower rejection ratio by a margin around 20% to 40%.



(a) CR = 25%, DegB = 12

(b) CR = 50%, DegB = 12

(c) CR = 25%, DegB = 24

(d) CR = 50%, DegB = 24

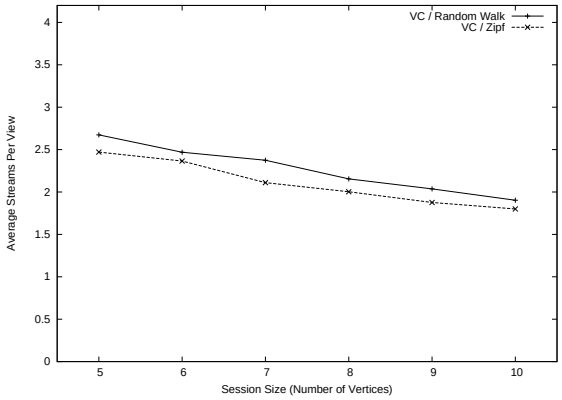
(e) CR = 25%, DegB = 36

(f) CR = 50%, DegB = 36

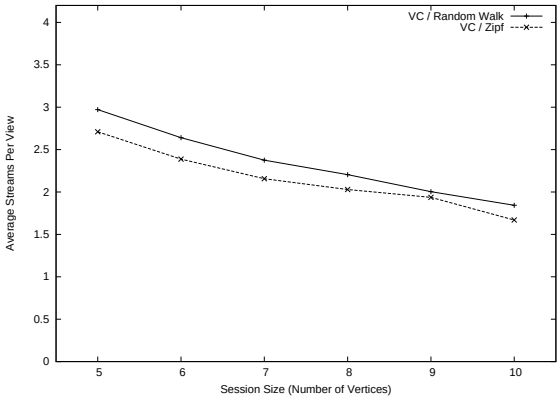
Figure 5.10: Average Number of Streams Per View

5.6.4 Workload

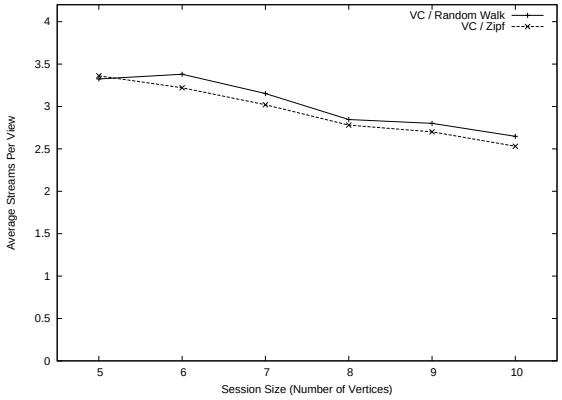
We measure the standard deviation of workload among vertices to investigate how evenly the forwarding load is divided. The workload is defined as the number of out-bound streams



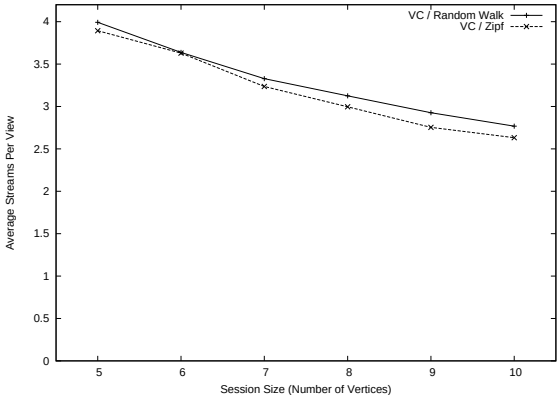
(g) CR = 75%, DegB = 12



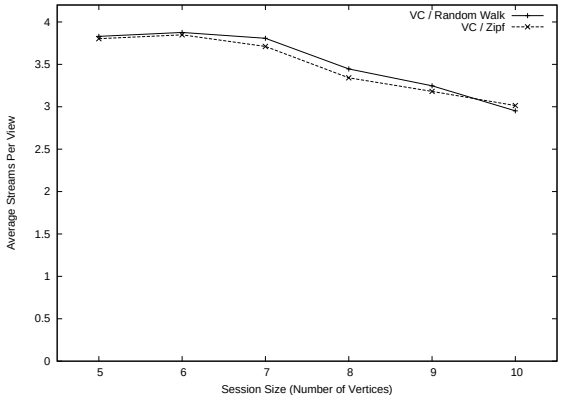
(h) CR = 100%, DegB = 12



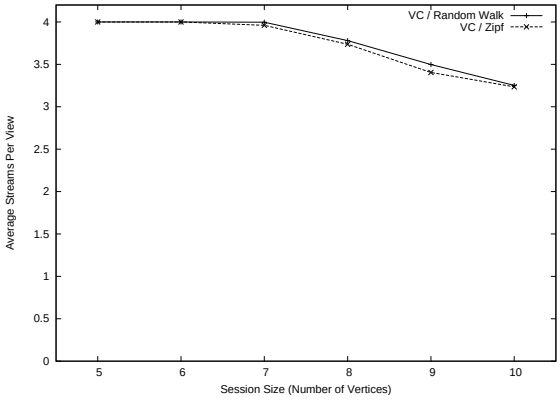
(i) CR = 75%, DegB = 24



(j) CR = 100%, DegB = 24



(k) CR = 75%, DegB = 36

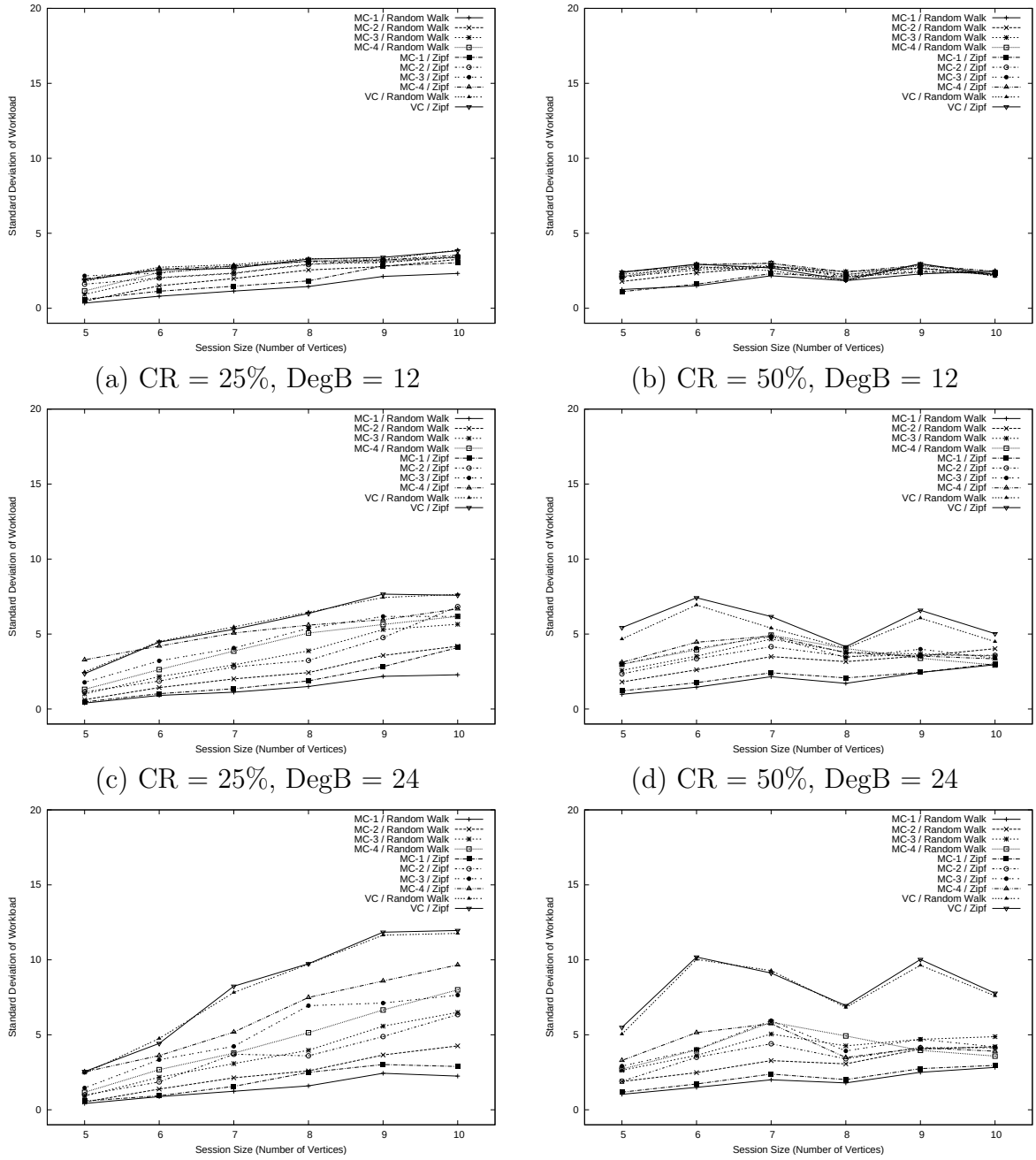


(l) CR = 100%, DegB = 36

Figure 5.11: Average Number of Streams Per View (*continued*)

served by a vertex. The results of standard deviation of workload are plotted in Figure 5.12 and Figure 5.13.

Those figures indicate several trends. First, the standard deviation grows bigger as the degree bound increases. Second, when the overall request of stream sharing increases (i.e.,



(a) CR = 25%, DegB = 12

(b) CR = 50%, DegB = 12

(c) CR = 25%, DegB = 24

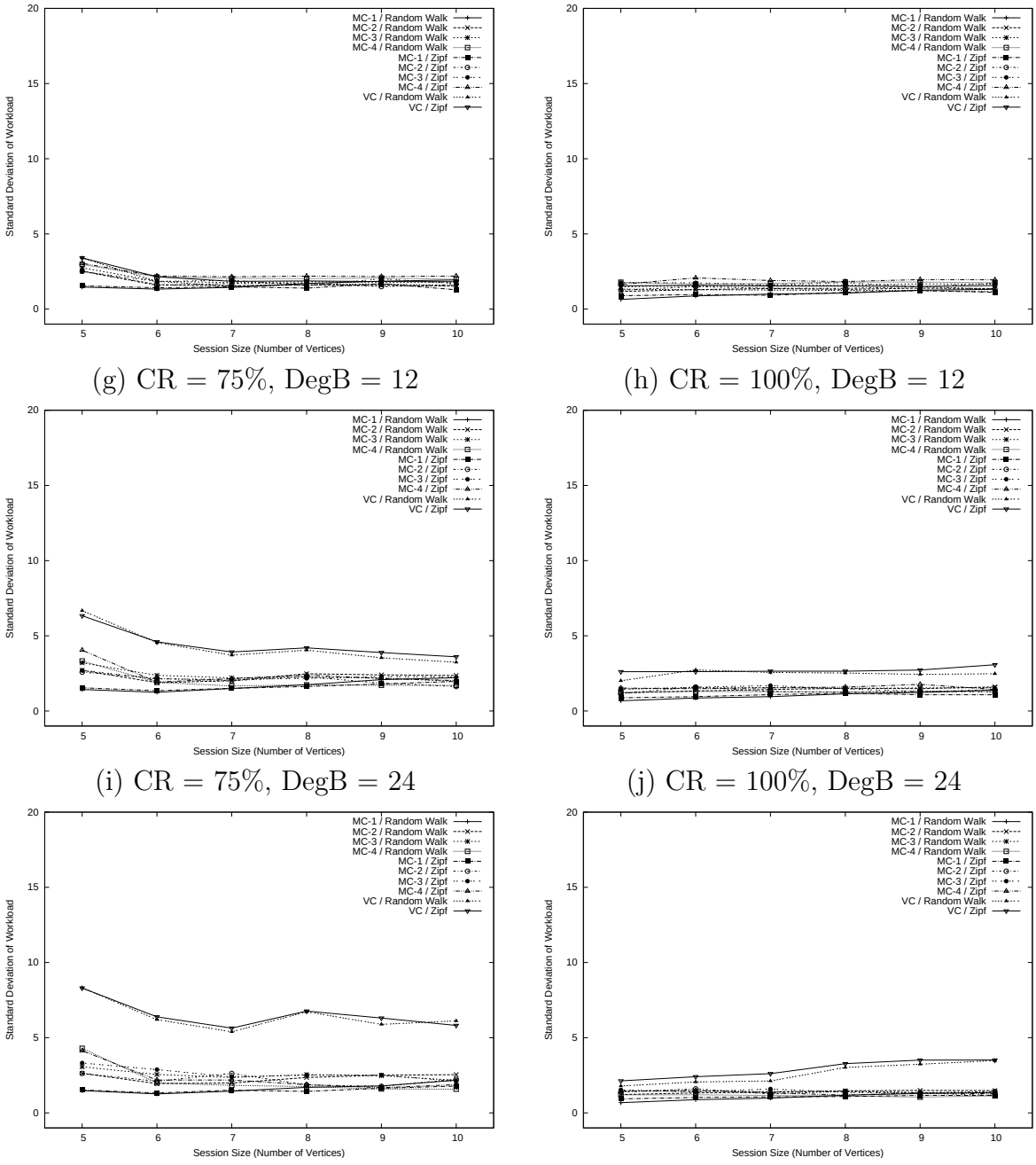
(d) CR = 50%, DegB = 24

(e) CR = 25%, DegB = 36

(f) CR = 50%, DegB = 36

Figure 5.12: Standard Deviation of Workload

MC-4 or MC-3) the workload tends to be more deviated. Third, we notice in terms of load sharing ViewCast tends to perform not as good as multicast-based schemes, which is expected as ViewCast prefers forwarding load to specific vertex as the major load balancing criterion.



(g) CR = 75%, DegB = 12

(h) CR = 100%, DegB = 12

(i) CR = 75%, DegB = 24

(j) CR = 100%, DegB = 24

(k) CR = 75%, DegB = 36

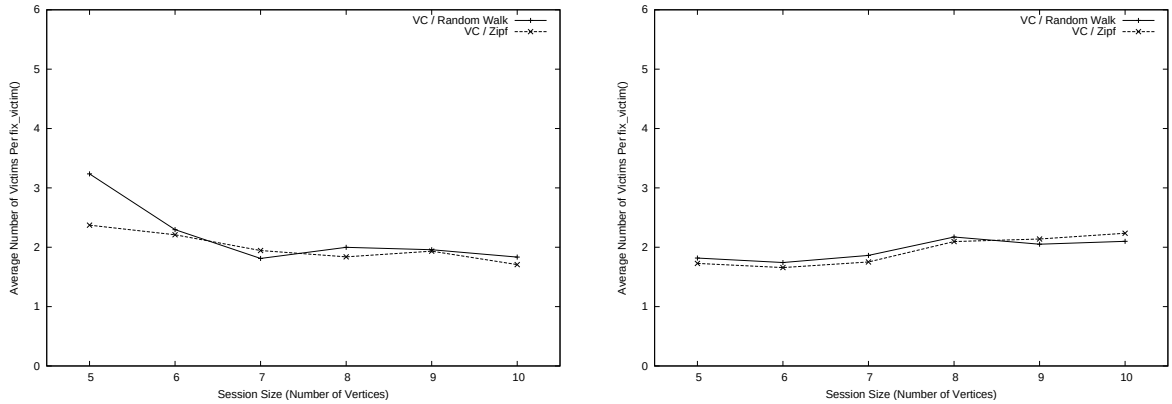
(l) CR = 100%, DegB = 36

Figure 5.13: Standard Deviation of Workload (*continued*)

5.6.5 Collateral Cost of ViewCast

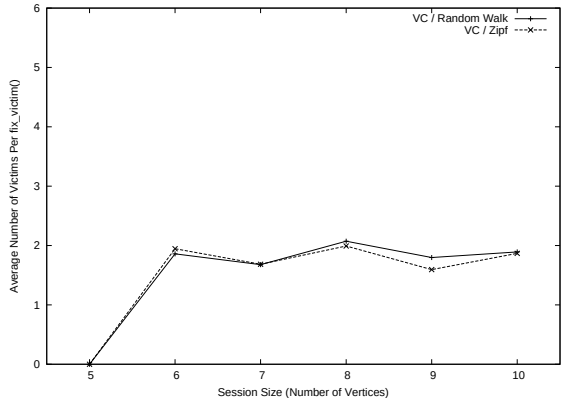
In ViewCast, due to the preemption of network resource and view change some vertices may have their streams involuntarily discontinued. We call those broken links the *victims*. More

precisely, a victim is defined as a pair of (v, s) where $v \in V$ and $s \in S$. After each view request is served, the routine of `fix_victims()` is called to fix any possible victims. The size of victim set indicates the scope of affected vertex/stream pairs. In the simulation, we measure the average number of victims. The results are plotted in Figure 5.14 and Figure 5.15.

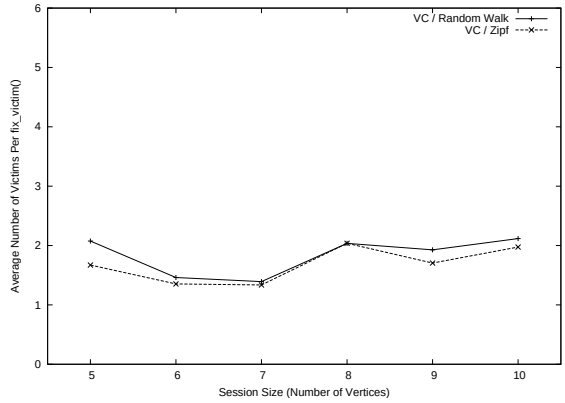


(a) CR = 25%, DegB = 12

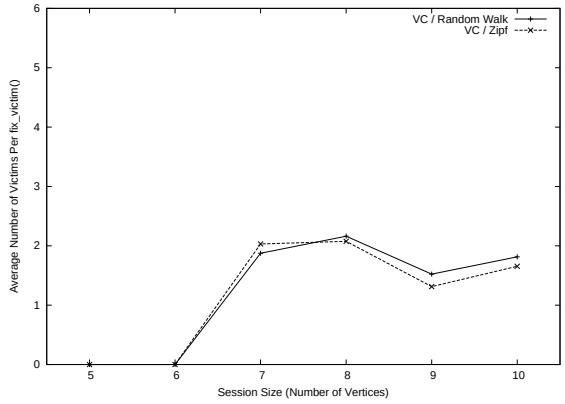
(b) CR = 50%, DegB = 12



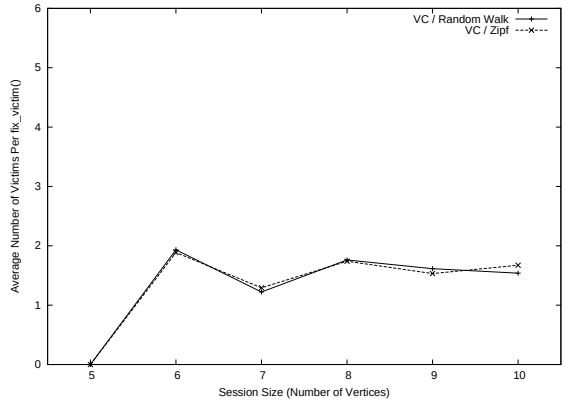
(c) CR = 25%, DegB = 24



(d) CR = 50%, DegB = 24

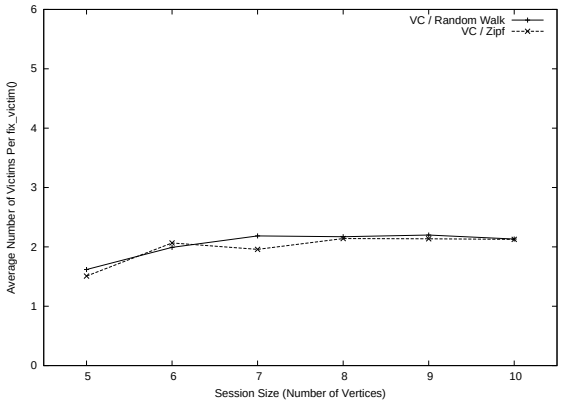


(e) CR = 25%, DegB = 36

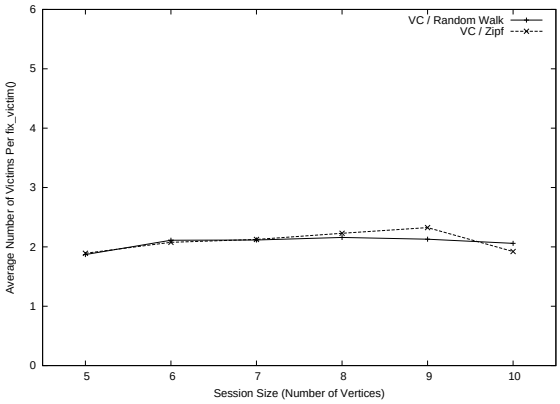


(f) CR = 50%, DegB = 36

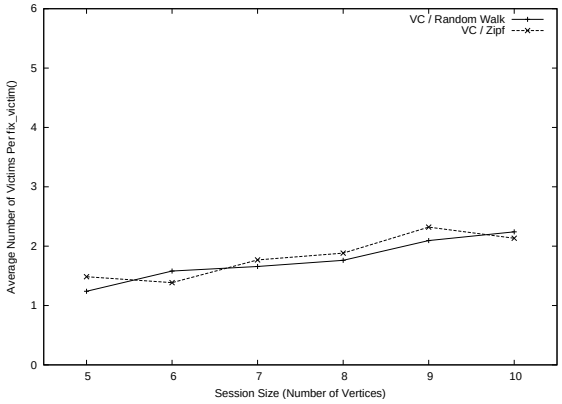
Figure 5.14: Average Number of Victims Per `fix_victim()`



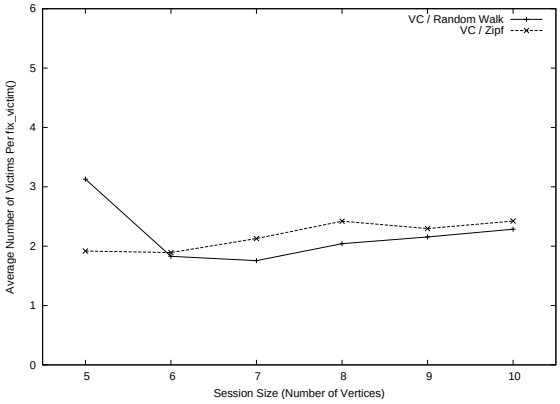
(g) CR = 75%, DegB = 12



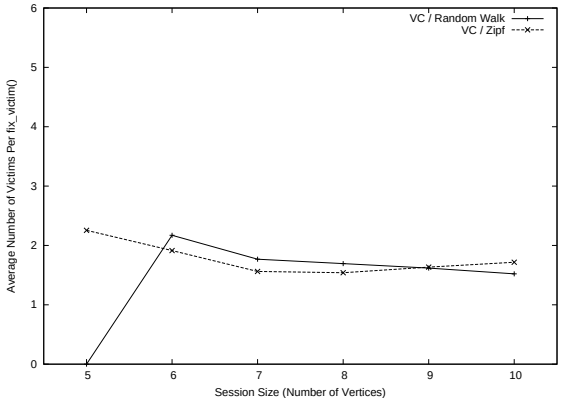
(h) CR = 100%, DegB = 12



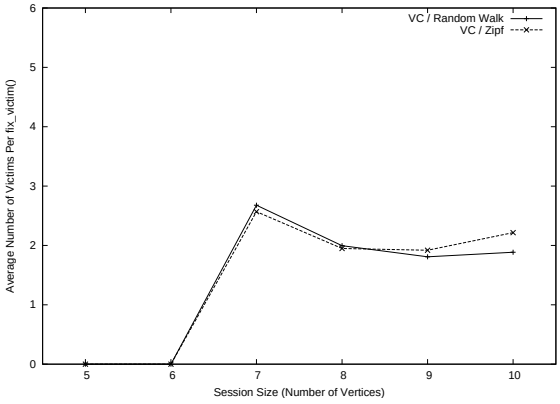
(i) CR = 75%, DegB = 24



(j) CR = 100%, DegB = 24



(k) CR = 75%, DegB = 36



(l) CR = 100%, DegB = 36

Figure 5.15: Average Number of Victims Per `fix_victim()` (*continued*)

The results indicate that in most cases the number of victims is small (around $2 \sim 3$), which is quite acceptable.

Chapter 6

Streaming Control

The purpose of streaming control is to ensure temporal quality, which is important for satisfactory live tele-communication. Particularly, we are interested in maintaining a fixed frame rate of rendering throughout the entire 3DTI session to reduce jerkiness. To achieve that, we use the video frame as the probing packet and feed the result into a PID controller to fine tune the frame size such that the arrival of each frame meets the deadline of rendering. The details are given in this chapter.

6.1 Problem Description

The streaming control layer is the bottom layer of the management framework. Below it, there is the transport layer. We choose TCP as the transport protocol due to its desirable properties including reliable and in-order delivery, congestion control, and the easiness of handling large size packet at the application layer. Those features are well suited for transmitting large volume data. Even though the backoff and retransmission mechanisms seem to make it unfit for real-time streaming, it has been shown that TCP is widely used in commercial systems ([52]). Note that, the streaming control method can be applied to UDP-based transport protocol as well, e.g., the tcp-friendly rate control protocol ([18]).

Figure 6.1 illustrates the basic problem of streaming control based on the timing model introduced in Chapter 2. At the capturing tier, macro-frames are generated at the fixed period. Accordingly, at the rendering tier we apply a timer which has the same period as the capturing. Thus, for each macro-frame F_t we have $T_{disp}(F_t)$ which is the time that F_t

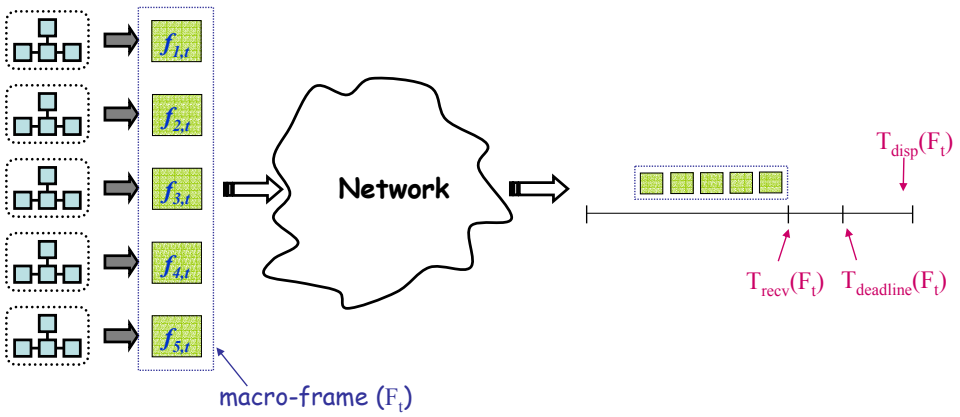


Figure 6.1: Streaming Control Problem

must be rendered. There are two other related time instants, $T_{deadline}(F_t)$ which is the time that F_t must be received and $T_{rcv}(F_t)$ which is the actual time that F_t is received. Ideally, we should guarantee $T_{rcv}(F_t) \leq T_{deadline}(F_t)$ for every macro-frame.

Suppose the network is stable. If we increase the size of macro-frame, then $T_{rcv}(F_t)$ will grow towards $T_{deadline}(F_t)$. Since it is generally preferrable to have larger frame size, the streaming control should choose a suitable frame size such that $T_{rcv}(F_t)$ is as close to $T_{deadline}(F_t)$ as possible. On the other hand, suppose the network status changes with the available bandwidth decreased. The direct consequence is that $T_{rcv}(F_t)$ will become bigger and may finally grow beyond $T_{deadline}(F_t)$ which indicates that the frame size is too big under the current network condition.

Therefore, we can monitor the difference of $T_{deadline}(F_t) - T_{rcv}(F_t)$ as the feedback for the control process. The goal of control is to keep the difference close to 0. If $T_{deadline}(F_t) - T_{rcv}(F_t) > 0$, the size of next frames should be increased. Otherwise, if $T_{deadline}(F_t) - T_{rcv}(F_t) < 0$, the size of next frames should be decreased.

In this chapter, we present the scheme of PID-based (proportional-integral-derivative) streaming control. We first introduce the PID controller [19]. Then, we describe how it is applied in the streaming control. Finally, we show the experimental results as evaluation.

6.2 PID Controller

Figure 6.2 illustrates an overall control system. In the figure, *plant* is the component whose output (Y) needs to be controlled. The controller performs the control function (U_f), based on the error (e) which is the difference between the desired value (R_v) and the output. The output of the control function is given to the plant to enforce the control.

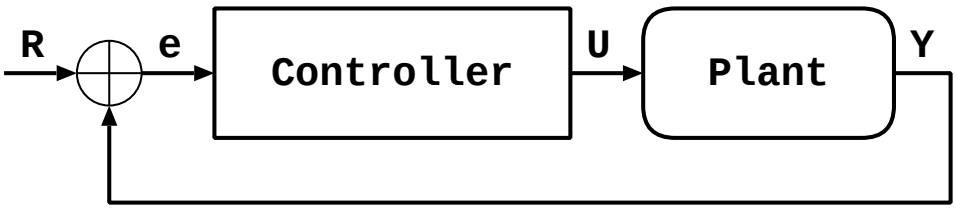


Figure 6.2: An Overall Control System

The basic PID control function is given in Equation 6.1, which is a linear combination of a proportional term, an integral term and a derivative term,

$$u_f(t) = K_p \left(e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{de(t)}{dt} \right) \tag{6.1}$$

where $e(t)$ is the difference between the desired value R_v and the actual output Y, K_p is the proportional gain, T_i is the integral factor and T_d is the derivative factor. For the discrete form, we could use the following approximation.

$$\int_0^t e(\tau) d\tau \approx \Delta t \sum_{j=0}^{n} e(j) \qquad \frac{de(t)}{dt} \approx \frac{e(n) - e(n-1)}{\Delta t}$$

where Δt is the sampling period. By introducing the integral gain K_i and the derivative gain K_d, the discrete case of PID function can be derived as in Equation 6.2.

$$u_f(n) = K_p e(n) + K_i \sum_{j=0}^{n} e(j) + K_d (e(n) - e(n-1)) \tag{6.2}$$

Table 6.1: Program of PID controller

update(error)
  $t \leftarrow$ get_time()
  $\Delta t \leftarrow t - last\_time$
  $d_{error} \leftarrow \frac{error - prev\_error}{\Delta t}$
  $i_{error} \leftarrow i_{error} + error \times \Delta t$
  $frame\_size \leftarrow K_p \times error + K_i \times i_{error} + K_d \times d_{error}$
  $prev\_error \leftarrow error$
  $last\_time \leftarrow t$
end

where,

$$K_i = \frac{K_p \Delta t}{T_i} \qquad K_d = \frac{K_p T_d}{\Delta t}$$

The design procedure is to tune the parameters of K_p, K_i and K_d. For this, we choose the Ziegler-Nichols method as a reference [60], which is a very popular online tuning strategy. First, we set K_i and K_d to zero and increase K_p until there is sustained and stable oscillation in the output. Once the critical gain (K_c) and the oscillation period (P_c) are obtained, the parameters can be set as in Equation 6.3. The tuning details are given in Section 6.4.

$$K_p = 0.65K_c \quad T_i = 0.5 \times P_c \quad T_d = 0.125 \times P_c \tag{6.3}$$

6.3 Implementation

The programming of PID function is pretty straightforward. The sketch of the program is listed in Table 6.1, where the `update` routine is executed at the sender side whenever an acknowledgement packet is received.

The main implementation issue is how to start the timer at the rendering tier. Figure 6.3 illustrates the problem. Suppose at the beginning of the first period, a macro-frame F_1 is pushed to the network by the capturing tier. After certain amount of time, the macro-frame is received by the rendering tier at time $T_{rcv}(F_1)$. The question is how to determine an

appropriate deadline for macro-frame F_1, which depends on when we start the rendering timer. Seemingly, the most appropriate time to start the rendering timer is the time instant when the first bit of F_1 is received (i.e., T_0). However, since it is not easy to acquire this value we use a simple method of approximation.

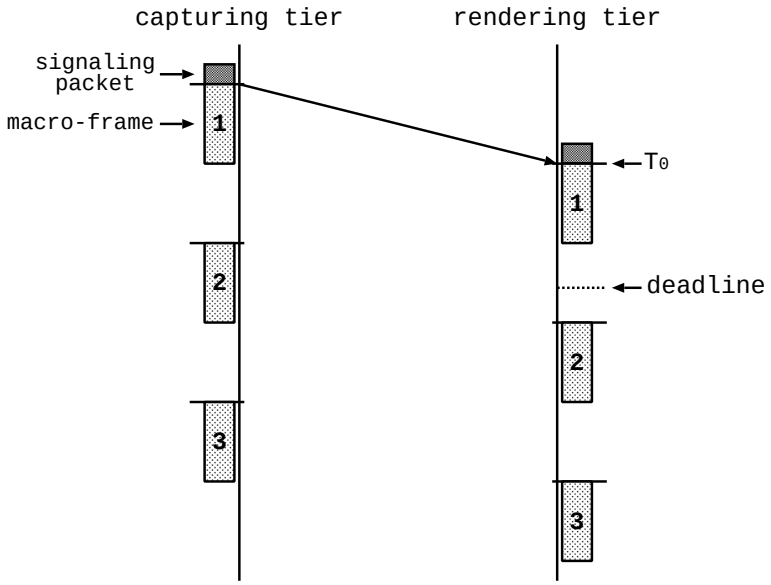


Figure 6.3: Initialization of the Rendering Timer

Before the capturing timer starts, a small packet is sent to the rendering tier. The rendering tier records the receipt time of the packet, which we use as the start time of the rendering timer, T_0.

6.4 Evaluation

6.4.1 Experiment Setup

We perform the experiment to evaluate the feasibility of the PID controller. For the data, we set the frame rate (fr) as 5 frames/second. Thus, the period is 200 ms. We set the deadline to be 20 ms ahead of the next period. For the networking, we use two different settings. In the first setting, we deploy one data sender at the University of California at

Berkeley and one data receiver at the University of Illinois at Urbana-Champaign. For the second setting, we deploy one data sender at the uplink of the broadband network and one data receiver at the University of Illinois at Urbana-Champaign.

6.4.2 Network Setting: UC Berkeley → UIUC

We use the Ziegler-Nichols method for tuning the control parameters. First we set $K_i = 0$ and $K_d = 0$, We find the critical gain K_c around 380000. We then set $K_p = 247000$. We choose $P_c = \frac{2}{fr}$ and $\Delta t = fr$ with $fr = 5$ frames/second. Thus, we start to probe K_i from 500000 and K_d from 15000 as suggested by the Ziegler-Nichols method. The results of one set of parameters are plotted in Figure 6.4.

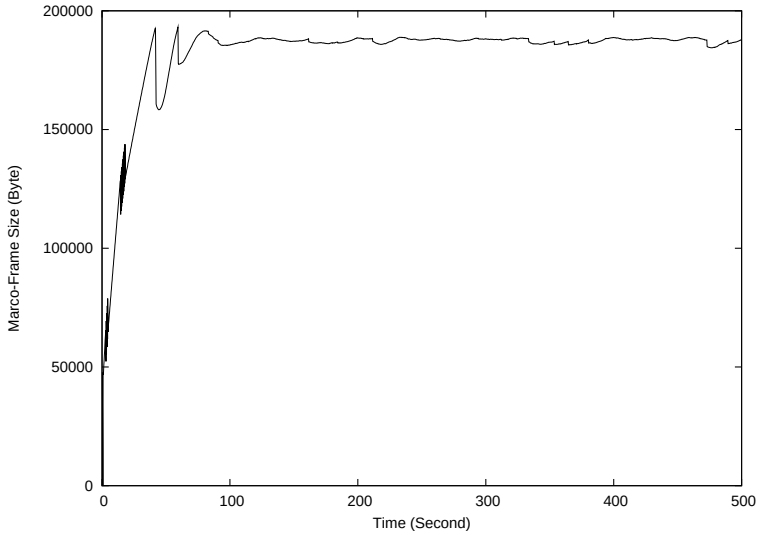


Figure 6.4: Macro-frame Size with $K_p = 247000$, $K_i = 60000$ and $K_d = 1000$

We run the test for 500 seconds. As shown in the figure, the frame size starts to converge at 110 seconds. In the stabilized area, the average frame size is 187 KBytes, indicating a throughput of 7.5 Mbps for one TCP connection. The standard deviation of frame size is 929 Bytes. We also plot the error in Figure 6.5. In the stabilized area, the average of absolute error is 0.001296 second and the standard deviation of error is 0.001324 second. Those results indicate that the converging is very fast, and the variation of frame size and

frame arrival is pretty small.

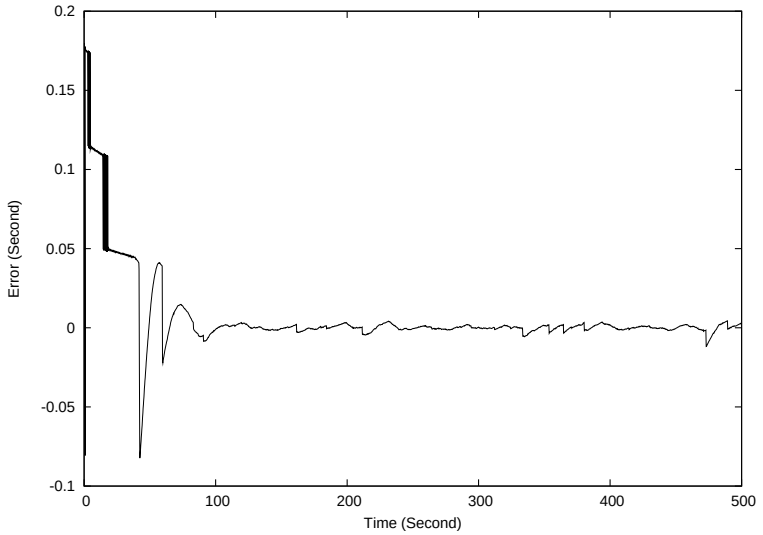


Figure 6.5: Error with $K_p = 247000$, $K_i = 60000$ and $K_d = 1000$

6.4.3 Network Setting: Broadband User → UIUC

We take similar approach for tuning the control parameters. We first find the critical gain K_c around 15000 and 20000. We still choose $P_c = \frac{2}{fr}$ and $\Delta t = fr$ with $fr = 5$ frames/second. Therefore, we derive $K_p = 10000$, $K_i = 40000$ and $K_d = 625$. We use above parameters and run the experiment for 300 seconds. The results are plotted in Figure 6.6.

As shown in the figure, the frame size starts to converge around 20 seconds. In the stabilized area, the average frame size is 24 KBytes, indicating an uplink throughput of 960 Kbps. The error is plotted in Figure 6.7. The average of absolute error in the stabilized area is 0.003803 second and the standard deviation is 0.003572 second.

We also try different settings of parameters. Figure 6.8 shows the result with $K_p = 10000$, $K_i = 20000$ and $K_d = 625$. We reduce the gain of K_i and find that the time of convergence grows to around 35 seconds. However, the variation becomes much less.

In the following tests, we try two more settings with (1) $K_p = 10000$, $K_i = 20000$ and $K_d = 300$; and (2) $K_p = 10000$, $K_i = 20000$ and $K_d = 900$. The results are plotted in

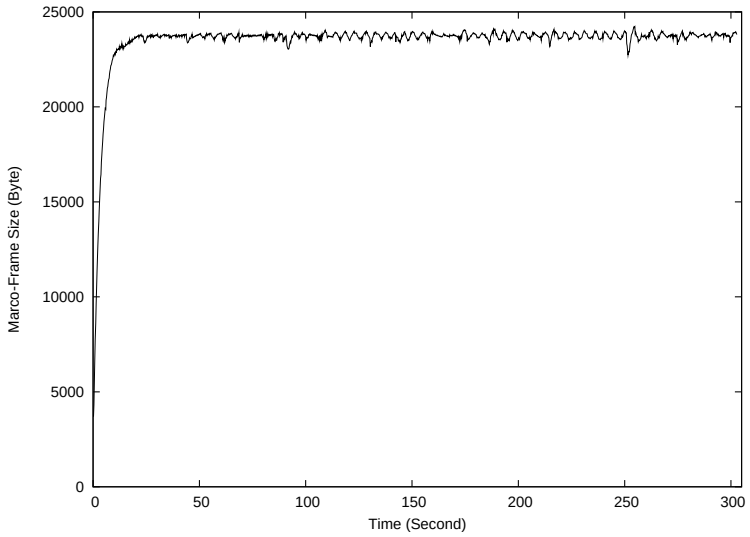


Figure 6.6: Macro-frame size with $K_p = 10000$, $K_i = 40000$ and $K_d = 625$

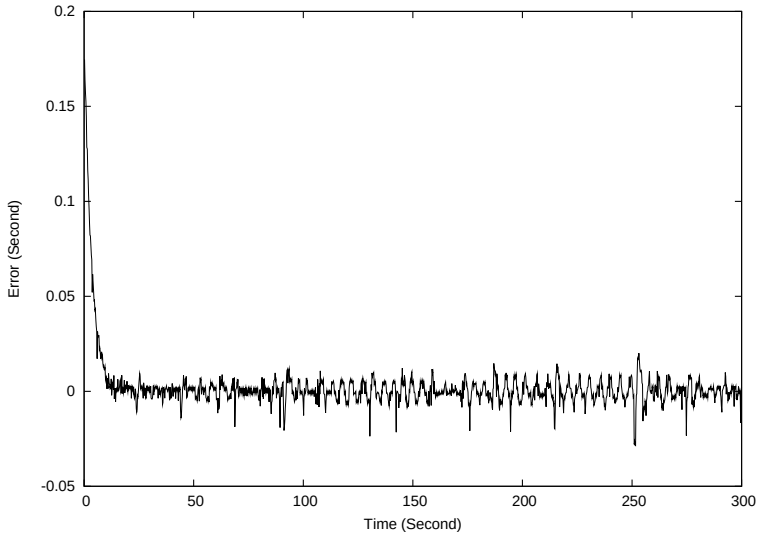


Figure 6.7: Error with $K_p = 10000$, $K_i = 40000$ and $K_d = 625$

Figure 6.9(a) and Figure 6.9(b) respectively.

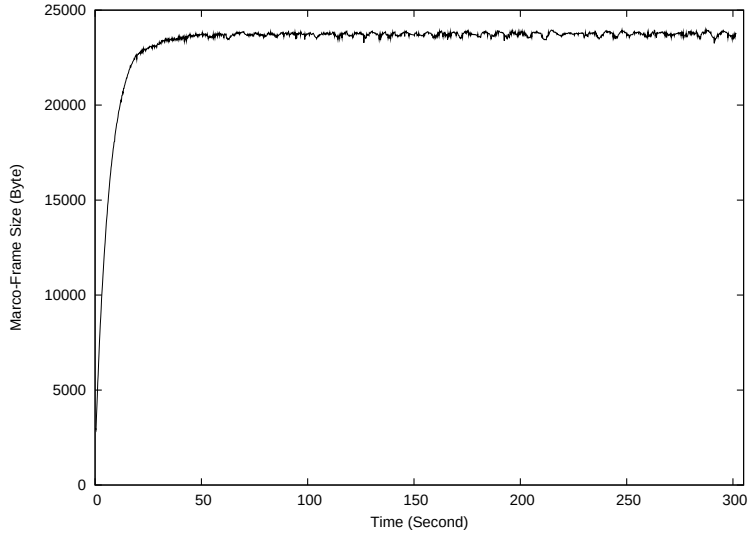
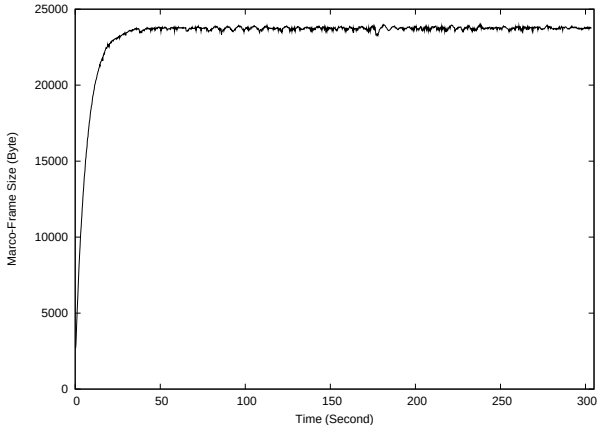
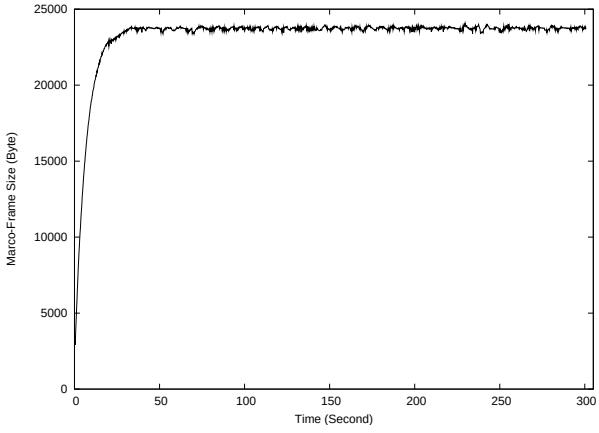


Figure 6.8: Macro-frame size with $K_p = 10000$, $K_i = 20000$ and $K_d = 625$



(a) $K_p = 10000$, $K_i = 20000$ and $K_d = 300$



(b) $K_p = 10000$, $K_i = 20000$ and $K_d = 900$

Figure 6.9: Streaming Control Performance with Different Settings of PID Gains

Chapter 7

Real-Time 3D Video Compression

For the service middleware layer, one major challenge is to accommodate the huge data rate from the application layer. This is especially true at higher frame resolutions and rates. Real-time 3D video compression is needed to alleviate this bandwidth requirement. Another challenge for 3D video compression lies in the diversity of data. Existing image/video compression algorithms such as JPEG, MPEG, and H.263 cannot be applied directly to 3D video compression because the video frame includes not only color information, but also depth information. Thus, we need to design a compression scheme that can handle both color and depth information.

7.1 Design Methodology

Because of their different properties, the color and depth data must be considered differently. The human visual system is relatively insensitive to variations in color. 2D compression algorithms such as MPEG and JPEG take advantage of this insensitivity by using lossy compression methods that still perform well on color video images. However, any loss of depth information may distort the rendered volumetric image. Therefore, we decided to use a lossless algorithm to compress the depth information. In addition, the compression scheme must meet the following goals.

- The compression and decompression must be performed fast enough for real-time interactive communication.

- There must be sufficient compression to accommodate the underlying bandwidth limitation.
- The compression algorithm should balance information loss against time and space cost to allow effective real-time transmission of 3D images with acceptable visual quality.
- The intra-stream compression algorithm must take into account particular 3D representations used in tele-immersive environments and should be compatible with inter-stream compression algorithms.

In the following sections, we present one intra-stream compression scheme for 3D depth images motivated by the design considerations given above.

7.2 Intra-stream Compression Scheme

We apply *color reduction* to compress the color information and then uses *zlib* to further compress all the data, including the depth information. The scheme is denoted as *CR-zlib* scheme. Color reduction is a popular image compression technique [2, 1] that reduces the number of bits used to represent colors (e.g. reducing from 24 bits to 8 bits per pixel). Color reduction can maintain high visual quality in most cases. In addition, there are several reasons for using color reduction in the context of 3DTI environments.

- Color reduction is suitable in the case of simple color composition, which is typical in a tele-conferencing setup. For example, Figure 7.1 shows the distribution of colors in RGB space from 612 depth images captured in our 3DTI experiments, featuring a person moving against a blank background.
- As our experiments show, if the color composition is relatively constant, then the runtime overhead of color compression can be kept small. This property also fits well with 3DTI environments.

- Color reduction is a pixel-level operation that can be performed on a *partial image* (i.e. image with some of its pixels removed). Therefore, it can be easily combined with the rendering independency property.

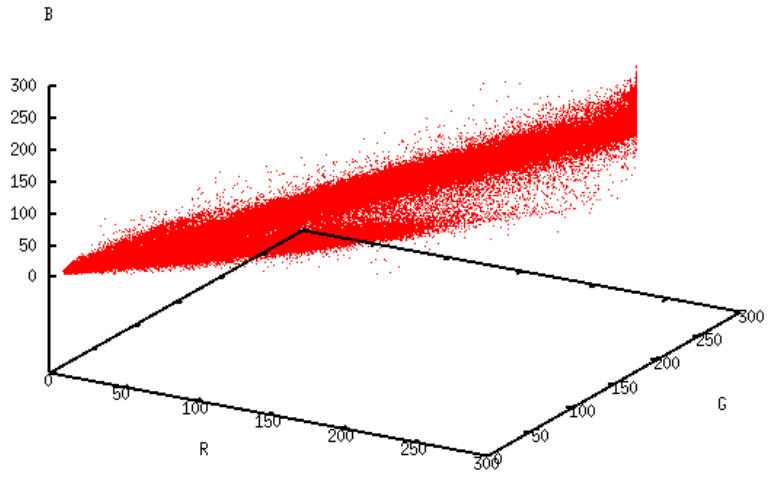


Figure 7.1: Color Distribution of 3DTI Video

7.2.1 Color Reduction

We briefly describe one color reduction algorithm tailored for 3DTI environments. The algorithm is based on ImageMagick [2], which contains three phases: *classification*, *reduction*, and *assignment*.

Color classification builds a color description tree for the image in RGB color space. In Figure 7.2, the root of the tree represents the entire space from $[0, 0, 0]$ to $[C_{max}, C_{max}, C_{max}]$ (usually $C_{max} = 255$). Each lower level is generated by subdividing the cube of one node into eight smaller cubes of equal size. For each pixel in the input image, classification scans downward from the root of the color description tree. At each level of the tree, it identifies the single node that represents a cube containing the color of the pixel and updates the statistics including the quantization error in that node.

Color reduction collapses the color description tree until the number it represents is at most the number of colors desired in the output image. The goal is to minimize the

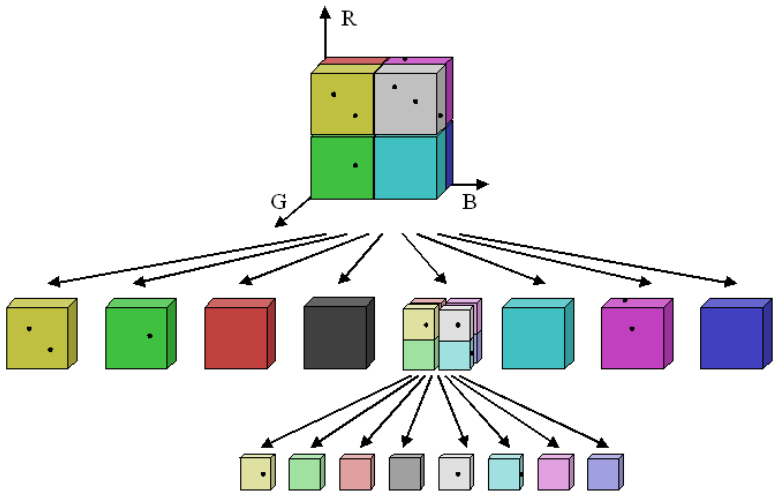


Figure 7.2: Color Description Tree

numerical discrepancies between the original colors and quantized colors. For this, color reduction repeatedly prunes the tree. On any given iteration over the tree, it selects those nodes whose quantization error is minimal for pruning and merges their color statistics upward.

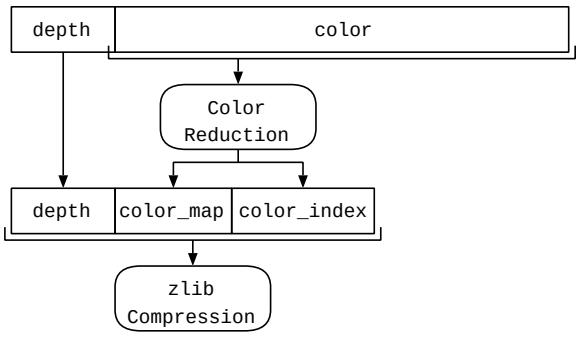
Finally, color assignment generates the output image from the pruned tree. It defines the *color map* of the output image and sets the color of each pixel via indexing into the color map. For the example of 24-bit to 8-bit color reduction, the output image contains the color map of 768 bytes and an index array of pixels that is one third the size of the original pixel array, achieving a total compression ratio close to 3.

7.2.2 zlib Compression

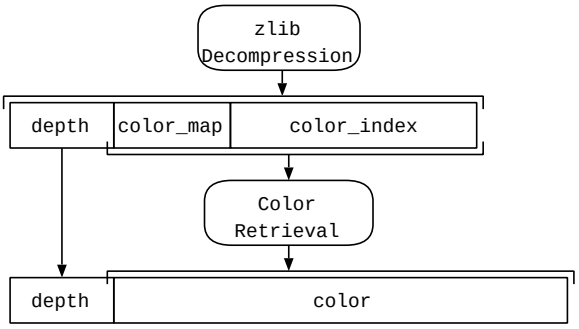
After color reduction, *zlib* [5] compression, a powerful, generic and openly available compression tool, is applied to the depth information along with the reduced color information to further improve the compression ratio.

The decompression follows similar steps in reverse except that color reduction is much simpler. Once the color map and the color index array are restored, the original color information can be easily recovered. As a summary, the overall compression and decompression

procedures are illustrated in Figure 7.2.2.



(a) Compression Diagram



(b) Decompression Diagram

7.2.3 Practical Issues

Color reduction is regarded as an expensive operation, which may affect the performance of real-time image transmission. However, because the basic color layout of the image frames in a tele-immersion session does not change dramatically (the colors of people's hair, face and clothes do not change), we need not perform the full-fledged color reduction algorithm for each frame. Once the color description tree has been set up and properly pruned (via the classification and reduction phases), the output color-reduced image can be generated during the assignment phase. While the first frame of the session has to go through all three phases, the follow-up frames can simply reuse the tree generated in the first frame to generate the output image directly; this involves only the assignment phase. Since most compute-intensive operations of the algorithm happen during the classification and reduction phases, much computing overhead is saved.

Furthermore, a runtime monitoring feedback loop can be introduced at the sender side to periodically calculate the color reduction error. If the errors become consistently larger than a threshold, the original pruned tree has become less accurate at representing the color layout of the current scene. In this case, we rerun the entire algorithm on the current image frame to generate a new tree, and then continue the same procedure as described above. The larger computing cost can be amortized over n frames. For tele-immersive environments, in

which the color layout is assumed to be very stable, the average value of n could be large.

As shown in the experimental results below, this compression scheme is very well suited to our problem domain, giving good compression ratios as well as good real-time performance and visual quality. Although color reduction is usually considered a time-consuming operation, we show that under the context of tele-immersive environments most of the computing overhead can be eliminated to achieve very high performance for 3D video compression.

7.3 Evaluation

For comparison, we introduce another scheme, *JPEG-RH*, which uses motion JPEG for color compression and run-length coding (RLE) plus Huffman coding for depth compression. For implementation, we use the Independent JPEG Group's library software. For the depth compression, we use the Basic Compression Library by Marcus Geelnard.

7.3.1 Evaluation Metrics

According to the design goals, we evaluate and compare the performance of the two compression schemes in terms of the *time cost*, *compression ratio*, and *visual fidelity*.

- *Time Cost.* The time cost represents the compression and decompression time for one image frame. For compression, we are also interested in measuring the time cost of the individual components: (1) for the CR-zlib Scheme, the time cost of each phase of color reduction and zlib compression, (2) for the JPEG-RH Scheme, the cost of color and depth compression.

- *Compression Ratio* (CR). The compression ratio is defined as: $CR = \frac{size\ of\ original\ data}{size\ of\ compressed\ data}$.

- *Visual Fidelity.* The video fidelity is measured in terms of the *peak signal-to-noise ratio* (PSNR), which is commonly used as a measure of reconstruction quality in image compression.

7.3.2 Environment

We have implemented both compression schemes in C/C++ on the Linux operating system and tested our code on Dell Precision 450 (Dual Xeon processor with 1 GBytes memory) computers running Fedora Core 2. We decided to integrate our compression algorithm with the UIUC/UC Berkeley tele-immersion system so that we could (1) transmit the compressed frames and ensure that our compression and decompression were correct and (2) evaluate the performance of the system with our compression installed as compared with the base system. For testing and evaluation, we used a 3D video scene pre-recorded in the tele-immersive environment that shows a person and his physical movement (Figure **??**). Currently, our system uses an image resolution of 320×240. Each uncompressed video stream in our test tele-immersive recording contains 612 depth frames with a total per-stream size of 235 MBytes ($612 \times 320 \times 240 \times 5$).

In establishing our testing and evaluation environment, we wanted to reuse the existing system as much as possible. Therefore, we integrated the compression and decompression code into the service middleware layer. This integration made the compression and decompression transparent to the capturing and rendering tiers and allowed us to deploy the code with minimal system modification. The experimentation consisted of several runs, each involving the compression and decompression of individual video frames as follows. The service gateway at the sender end compressed a frame and transmitted it to the receiver, where it was decompressed and compared with a local copy of the original frame to measure degradation of visual fidelity.

7.3.3 Compression Time

Table 7.1 shows the compression time of both schemes. The time unit is one millisecond (ms). For the CR-zlib scheme, the most expensive operation in terms of time taken is color classification and reduction with an average around 35 ms. However, as mentioned earlier,

it is reasonable to assume a stable color layout in a tele-conferencing environment such that the classification and reduction phases need to be performed only for the first frame of the session or every n frames based on a monitoring mechanism. In between, frames may omit the classification and reduction steps. For a stable color layout, n could be large (e.g. $n = 100$). Therefore, the average compression time can be taken as around 10 ms, as we assume an average recalculation period of 10 seconds and a frame rate of 10 frames per second.

For the JPEG-RH scheme, the average compression time is around 13 ms. The color (JPEG) compression performed better than the depth compression (even though the color contains more data). This is probably because (1) the depth compression uses two different types of compression (run-length encoding and Huffman coding), (2) the JPEG implementation we use may be more efficient than the implementations of RLE and Huffman coding, and (3) the Huffman coding implementation recomputes the Huffman table for each frame, whereas JPEG uses a fixed Huffman table.

The timing results of both compression schemes are quite good, with even the worst time being around 20 ms and well below 100 msec, which is the basic requirement for processing 10 3D frames per second. These results show that our compression schemes are feasible for the 3D video processing demanded by the 3DTI environments.

Table 7.1: Compression Time of Two Schemes

(a) CR-zlib Scheme (unit: ms)

	min	avg	max
classification	0.416	7.11	14.4
reduction	0.001	27.6	80.9
assignment	0.878	4.19	9.29
zlib compression	2.66	4.93	11.1

(b) JPEG-RH Scheme (unit: ms)

	min	avg	max
color compression	3.59	5.99	10.2
depth compression	3.68	7.02	10.5

7.3.4 Decompression Time

Table 7.2 shows the decompression times for both schemes. The decompression time of the CR-zlib scheme is better than that of the JPEG-RH scheme (with average 1.7 ms versus 9.2 ms). For the JPEG-RH scheme, the color decoding (JPEG) is on average faster than encoding, but depth decoding is significantly slower than depth encoding. This agrees with the manual for the Basic Compression Library we use for Huffman coding, which states that decoding is slower than encoding.

Table 7.2: Decompression Time of Two Schemes

(a) CR-zlib Scheme (unit: ms)

	min	avg	max
decompression	1.36	1.72	3.99

(b) JPEG-RH Scheme (unit: ms)

	min	avg	max
decompression	2.87	9.23	16.6

7.3.5 Compression Ratio

Table 7.3 gives the compression ratio achieved for the 612 depth frames. The performance of compression is very impressive (with a compression ratio of 25.9 for the CR-zlib scheme and 15.4 for the JPEG-RH scheme), which implies that on average the overall frame size (both color and depth) can be shrunk from 384 KBytes ($5 \times 320 \times 240$) to below 15 KBytes. This excellent compression reflects the large amount of spatial redundancy in the 3D depth image, which consists of a person moving against an empty background (Figure **??**). For color compression, color reduction and JPEG have comparable average compression performance. On the other hand, the depth compression performed by zlib is better than RLE plus Huffman coding.

7.3.6 Visual Fidelity

The visual fidelity of the color information after decompression is measured using PSNR (in dB). The results are given in Table 7.4, which indicates that both schemes have high

Table 7.3: Compression Ratio of Two Schemes

(a) CR-zlib Scheme

	min	avg	max
color ratio	2.97	2.97	2.97
overall ratio	14.6	25.9	338

(b) JPEG-RH Scheme

	min	avg	max
color ratio	9.57	15.8	126
depth ratio	7.69	14.9	272
overall ratio	8.69	15.4	200

compression quality, with the CR-zlib scheme achieving slightly better fidelity to the original image.

Table 7.4: PSNR of Two Schemes

(a) CR-zlib Scheme (unit: dB)

	min	avg	max
color	45.9	51.5	74.3

(b) JPEG-RH Scheme (unit: dB)

	min	avg	max
color	41.5	45.7	68.3

We also visually judge the image quality by comparing two similar frames before and after data compression as in Figure 7.3 and Figure 7.4. No subjective quality degradation is observed. We note that in many tele-immersion scenarios such as the one shown, the color layout of the image is rather simple, which yields unnoticeable degradation.

Figure 7.3: Visual Quality before Compression

Figure 7.4: Visual Quality after Compression

7.3.7 Lessons Learned

For color compression, the performance data indicates that color reduction could be a better choice than JPEG in terms of compression time, compression ratio, and visual fidelity. However, the time cost of color reduction depends largely on the particular tele-immersive video being processed. If the color change of the scene occurs very frequently and dramatically, then JPEG may give a much better performance. Hence as an extension, we propose a periodic monitoring and switching mechanism which operates at two levels. In the lower level, a monitoring thread decides whether a full-fledged execution of color reduction is necessary by measuring quality after decompression. In the higher level, if the most recent window of history shows a higher overhead for color reduction, the monitor may (depending on other factors such as bandwidth estimation) decide to switch to JPEG compression. The switching between the two schemes is very flexible as the compression method can be indicated in the frame header by using one extra bit. For depth compression, the performance of zlib is better than run length coding plus Huffman coding, though this is primarily an implementation issue of the different compression libraries.

Chapter 8

Related Work

We summarize previous work in five main aspects: (1) existing systems, (2) 3D compression, (3) multi-stream coordination, (4) view-based camera/stream selection, and (5) multicast-based content dissemination.

8.1 Existing Systems

Existing tele-conferencing and tele-immersive systems can be classified into three categories: (1) tele-conferencing systems over COTS computing platforms and existing Internet, (2) tele-presence systems relying on augmented OS/middleware support and, (3) tele-immersive systems over advanced networking service.

8.1.1 Tele-conferencing Systems over COTS Components

Most video conferencing systems widely in use today fall into the first category. Examples include PolyCom, Microsoft NetMeeting, WebEx, etc. These systems aim to provide point-to-point or limited multi-point communication for desktop users. Only a single view is available for each user through a 2D web camera. Since such systems produce medium or low quality video and audio stream compressed using standardized media format (H.263, MPEG4, etc.), the bandwidth requirement is largely compatible with a variety of networking environments including DSL and ISDN connections.

8.1.2 Tele-presence Systems over Augmented Components

Solutions in the second category aim to provide users the tele-presence experience beyond the single 2D view. In [15] and [20], participants of video conference are grouped and tiled into synthetic environments such as virtual auditorium or digital amphitheater. Coliseum [10] is a desktop-based immersive conferencing system. Here, the 3D view of a user is captured by multiple cameras, extracted from background, then reconstructed and embedded into the virtual environment. Finally, 2D video is created by locally rendering the 3D scene from the user-defined viewpoint, then streamed to the network. A commonality shared by these applications is that the transport services of current commodity operating systems (e.g., Windows) do not satisfy their demands for the media streaming purpose. Therefore, advanced software modules or middleware framework are developed to enhance the existing transport service to effectively support these applications. In [15] and [20], frame tiling and stream merging functions are provided to produce synthetic media stream. [10] created Nizza, a middleware framework to simplify the development of the media streaming subsystem.

8.1.3 Tele-immersive Systems over Advanced Networking Service

Solutions in the last category include Virtue [47], MetaVerse [41, 48], and the National Tele-Immersion Initiative [31, 39, 51]. These systems aim to provide teleimmersive realism to users. Similar to Coliseum [10], they rely on multiple cameras to capture the 3D scene. However, after reconstruction, the 3D data stream, instead of the 2D rendered view, is transmitted to the network. The advantage of this choice is to give users maximum freedom to watch the 3D scene from any viewpoint and change it anytime he/she wishes to do so. In case more than one user are present at a local site such as a conferencing room with multiple displays, they can also share the same 3D video stream by watching it from different

viewpoints. Due to the huge volume of 3D data stream, it poses significant challenges to the current transport service and the capacity of the traditional Internet infrastructure itself. The Tele-Immersion system developed by UNC Chapel Hill is deployed over the Internet2, in order to cope with its considerable traffic demand. The entire architecture of the network transport service also needs complete innovation. [52] proposed a cluster-to-cluster architecture. Here, gateways are inserted at both ends of the path, which aggregate and regulate all data flows through them. This solution was shown to well address the synchronous arrival and racing condition among multiple camera streams. In [48], an overlay-based multi-path routing service was introduced for efficient delivery of multiple streams. It is shown that by utilizing redundant paths provided by the overlay routing service, the aggregate bandwidth is significantly augmented, in comparison to the case of single end-to-end path. Other than attempts to modify the network transport service, many efforts are also made to design novel 3D data compression techniques. In [31], a 3D data compression scheme was proposed, which exploits the fact that many pixels in the 3D space are captured by multiple cameras. In order to remove these redundant points, one reference stream is chosen, which is used by other streams to remove the redundant pixels already appearing in the reference stream.

8.2 3D Compression

The 3D data model has a direct impact on the design of compression algorithms. Therefore, we divide the related work into three categories based on the underlying data models and present them in the order of relevance to our work including: (1) compression based on *depth images*, (2) compression based on *volumetric data*, and (3) compression based on *triangular meshes*.

8.2.1 Depth Image Compression

The data model of the first category is used in 3DTI environments as described in Chapter 2. Under this model, a 3D image (i.e., *macro-frame*) is represented with multiple 2D depth images captured by individual 3D cameras from different viewpoints at the same time instant. In contrast to an ordinary 2D image, the depth image has extra depth information for every pixel. There are two major compression methods: *inter-stream* and *intra-stream* compression. In inter-stream compression [31, 29], the 2D image closest to the viewpoint of the user is selected as the reference image, which is not compressed. Other images are compared with the reference image to remove redundant pixels within a certain threshold of depth distance. The method decreases the total number of pixels that need to be rendered as the non-reference images are reduced to differential images. The problems of inter-stream compression include the considerable communication overhead involved in broadcasting the reference image and the diminishing redundancy between images of larger viewpoint difference. To alleviate these problems, a two-level referencing and grouping scheme is applied. In addition, the performance of compression ratio is highly associated with the density of cameras. In one experiment ([31]), 22 3D cameras are deployed with a horizontal field of 42° to achieve a five to one compression ratio.

Intra-stream compression schemes ([56, 30]) process each stream individually without cross-stream comparison. In [56], color and depth components are compressed using different algorithms. The color is compressed using lossy compression while the depth using lossless compression. In [30], video stream encoding is extended to encode color and depth. Results indicate that a better compression ratio is achieved when color and depth are encoded using separate motion vectors. Intra-stream compression has several advantages. First, it achieves better scalability, since the overhead related to the number of streams is almost trivial as compared with inter-stream compression. Second, the compression ratio of intra-stream compression is consistently better than that of inter-stream compression with comparable

quality. Unlike the latter, the performance is not affected by the density of cameras as each stream is compressed separately.

The coding of multiple correlated video streams or *multi-view video coding* (MVC) has recently become an active topic including, for example, a multiview transcoder [8] and ISO survey of MVC algorithms [7]. The common idea is to augment MPEG encoding scheme with cross-stream prediction to exploit spatial redundancy among different streams. However, as pointed out earlier cross-stream compression could involve very high overhead. Most implemented systems we have seen so far still encode each stream independently such as a multi-view video system [34] and a 3D TV prototype [35].

8.2.2 Volumetric Data Compression

The volumetric data model in the second category refers to a regular 3D grid whose *voxels* contain RGB or grayscale information. The 3D data are derived from a discrete collection of samples generated by scientific simulations or by volumetric imaging scanners such as computerized tomography (CT) scanners. The typically large size of the resulting voxel datasets has been a driving factor in research targeting compression of volumetric images. For example, one 3D image from the male dataset of the National Library of Medicine (NLM) contains 1,878 cross-sectional (2D) images (*slices*) taken at 1mm intervals. The slice has a resolution of 512×512, and each voxel needs 16 bits to store grayscale information. The *3D wavelet transform* [53, 25, 9] is the most important compression method for this data model; it is an extension of 2D wavelet compression techniques for 2D images. To perform a 3D wavelet transform, a 3D image is first divided into *unit blocks* of size $16 \times 16 \times 16$ to take advantage of the spatial coherence in the cube. The wavelet coefficients are computed for each unit block. Non-zero coefficients are then further processed using quantization and entropy encoding. Although the 3D wavelet transform achieves a very high compression ratio, its application to the 3D video of tele-immersive environments is not straightforward and the overhead of the algorithm may offset its advantages. For example, it is not beneficial to

apply a 3D wavelet transform on a sparse dataset such as the contour of the subject. To make a wavelet transform moreefficient in a tele-immersive context, a transformation function is needed to first pack the data into a more dense volumetric form while maintaining a high spatial coherence within unit blocks. This function may involve significant time cost.

8.2.3 Triangular Meshes Compression

The last category of triangular meshes represents the most widely used 3D geometric model in computer graphics for purposes such as manufacturing, scientific computation, and gaming due to the fact that polygonal surfaces can be efficiently triangulated. The triangular representation of 3D geometry has two major components: vertex coordinates and triangles, and their associated properties including color. As complex scene representations may contain millions of triangles, the amount of data required to store such scenes may be quite large. On average, the number of triangles is twice the number of vertices, and each type of data may exhibit different kinds of data redundancy. Hence, the popular mesh compression algorithms treat vertices and triangles differently. The vertex data can be compressed using the *vertex spanning tree* as in the Topological Surgery approach [49] adopted by the MPEG-4 standard. The spanning tree is constructed to exploit spatial coherence, or the observation that proximity of the nodes in the tree often implies geometric proximity of the corresponding vertices. The ancestors of the tree can be used as predictors and only the differences between the predicted and actual values encoded. The differences are further processed using quantization and entropy coding. The property information can be compressed in a similar way. For triangle data, the basic unit of compression is the *triangle strip* which defines a particular order of traversal such that each new triangle can be represented by adding one vertex. Among triangle-based compression techniques proposed, *Edgebreaker* [45, 46] is the most advanced scheme; it encodes the traversal using an alphabet of five symbols with an overall performance of less than 2 bits per triangle. Recent efforts [32, 33, 46] aim to reduce the number of bits per triangle and extend the basic scheme to more arbitrary topologies. 3D

compression based on triangular meshes achieves a very high compression ratio. However, the challenge of applying this technique lies in the real-time and automatic acquisition of 3D models, which is currently an active research area.

8.3 Multi-stream Coordination

Coordination Protocol (CP) proposed in [39, 40] is a transport layer mechanism used for coordinating multiple streams in cluster-to-cluster applications. The protocol resides on the stack of the *aggregation point* (AP) which is a special host where all streams converge before an out-going shared path and must be under the local administrative control. For one cluster-to-cluster connection, a pair of AP's are needed to cooperate. To use the protocol, an application first registers its streams with the AP. Then the AP monitors the link status of each stream (e.g., round trip time, loss rate and available bandwidth) by exchanging the information in the protocol header. The per stream information aggregated by the AP can be retrieved by the application.

The advantage of CP is that it provides a general means where per stream information can be collected and disseminated for high level multi-stream management. The changes needed are restricted to the end-to-end scope instead of intermediate routers. However, the real question of how to coordinate multiple correlated streams is not answered.

8.4 View-based Camera/Stream Selection

View-based multi-stream selection has been used in 3DTI environments for two reasons. First, image-based vision techniques are shown to be a feasible solution for real-time 3D video systems ([6, 8]), where multiple cameras are distributed to reconstruct the 3D model from images taken in real scenes. However, image-based techniques require tremendous computational power and network resource if a large number of video streams need to be

processed in full scale. Second, as mentioned earlier the interactivity of dynamic view selection is the key feature of 3D video application. Therefore, the user view information provides a natural hint for multi-stream selection.

In the 3D video pipeline ([38]) implemented for the telepresence project ([22]), the video system installs 16 CCD cameras covering 360°. During the runtime, 3 cameras are selected for the texture and 5 cameras for reconstruction based on the user view. The concern of adaptation is more focused on the 3D video processing and encoding part to make it affordable within resource limitations. However, the issue of how to adapt the data according to the bandwidth and user requirement, and the related spatial and temporal quality loss has not been addressed.

In other cases, the limitation of human's viewpoint is exploited to facilitate the design and implementation of 3D video systems. For example, a prototype of group video teleconferencing system [55] is implemented which uses a linear array of cameras mounted horizontally at eye level to capture a compact light field as an approximation for light field rendering.

8.5 Multicast-based Content Dissemination

The most important application of the ViewCast concept is in the multi-party/multi-stream environment for QoS management, which distinguishes it from available protocols and techniques in several aspects.

Multicast protocols including application level multicast (e.g., [26, 11, 24]) are mostly concerned with efficient transmission of particular stream(s) for a group of receivers. In contrast, ViewCast is a higher-level concept which is focused on the coordination of multi-streaming among multiple groups.

The related awareness driven model has been applied in collaborative virtual environment ([21, 44, 23]) for quality of service management. Given the awareness information of the user, the model dynamically selects the set of sources and the quality. Usually, each source

represents one audio/video stream with multiple levels. However, the limitation of the model is its incapability of handling multiple correlated streams at each source and among sources as required in multi-party/multi-stream systems. For example, in [23] a multi-sender 3D videoconferencing application is implemented where certain 3D effect is created by placing the 2D image of each participant in the virtual space. So each user is represented by one 2D stream. In their work, stream selection is used to reduce the downlink traffic of each user based on the orientation of the view and the *visibility*. Streams that are not considered as visible will not be sent to a particular user. The application of stream selection is much simpler than the case of 3DTI environments where each user is represented by multiple streams.

Chapter 9

Conclusion and Future Work

This thesis addresses the problem of supporting 3DTI tele-immersive environments over general computing and networking infrastructure for broader audience. Our work will enable future generation of tele-immersive collaboration. We have presented a distributed service middleware framework to support view-aware multi-stream coordination, which includes the view-based stream differentiation layer, the stream coordination layer, and the streaming control layer. The organization of this chapter is as follows. First, we summarize our major contributions. Then, we briefly discuss possible future research directions.

9.1 Contributions

In an attempt to design and implement a distributed service middleware management framework for QoS-provisioning and for supporting the Internet killer-application of 3DTI environments, this thesis makes the following special contributions.

- **Integrated service middleware framework.** We propose a novel cross-layer framework to integrate various design concerns at different layers. From taking the view-oriented approach, accommodating stream coordination, down to manipulating compression and transmission at the lower layer, the framework manages every aspect of the 3DTI content dissemination. Thus, it can deliver high-quality QoS management for 3DTI application by properly controlling different service components.

- **View-oriented QoS provisioning.** Unlike previous systems which apply stream-level QoS management, we are making the breakthrough by the argument that QoS management in the multi-stream scenario should focus on the user view. The view-oriented approach not only provides more flexibility and adaptability but also a much more clear guidance to reach high-quality QoS management.

- **View-based content dissemination.** Traditional content dissemination structures are focused on delivering a set of streams. There is almost none or very little coordination in the dissemination. For the first time, we claim that it is to the common benefit of all content consumer that coordination should be performed. Using the *view* as the high-level quality criterion, we can prioritize the delivery topology to make it more efficient.

9.2 Future Work

The request for the next generation of communication will never end. Open questions still exist in realizing fully automatic QoS management system for 3DTI environments. We discuss several of the most interesting and challenging ones as follows.

- **Smart view management.** The theme of 3DTI environments revolves around the view idea. During the 3DTI session, the end user arbitrarily selects his/her view to get desirable visual effect. The next generation 3DTI system should provide automatic view management service. The advantage is to let the user fully enjoy the immersive environment with less intrusiveness of tracking devices. The view management will also have better cooperation with the underlying view-based content dissemination infrastructure.

- **Finer granularity multi-party viewCast.** Currently, ViewCast operates at the stream granularity which either selects or drops the whole stream. A more interesting

study would be to investigate view dissemination at finer granularity.

- **Multi-stream transcoding.** In the current ViewCast structure, each stream is transmitted independently. When user views are distributed in a relatively concentrated area range, a possible approach is to perform 3D multi-stream transcoding to generate a new set of streams which would improve the quality than using the original streams.

References

[1] Color reduction, http://www.catenary.com/appnotes/colred.html.

[2] Imagemagick, http://www.imagemagick.org/script/quantize.php.

[3] Opengl, http://www.opengl.org.

[4] Teeve project, http://cairo.cs.uiuc.edu/teleimmersion, http://tele-immersion.citris-uc.org.

[5] Zlib 1.2.2, http://www.zlib.net.

[6] Report on 3dav exploration. *International Organisation for Standardisation, ISO/IEC JTC1/SC29/WG11 N5878*, July 2003.

[7] Survey of algorithms used for multi-view video coding. *International Organisation for Standardisation, ISO/IEC JTC1/SC29/WG11 N6909*, January 2005.

[8] B. Bai and J. Harms. A multiview video transcoder. In *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*, pages 503–506, New York, NY, USA, 2005. ACM Press.

[9] C. Bajaj, I. Ihm, and S. Park. 3d rgb image compression for interactive applications. *ACM Trans. Graph.*, 20(1):10–38, 2001.

[10] H. Baker, N. Bhatti, D. Tanguay, I. Sobel, D. Gelb, M. Goss, W. Culbertson, and T. Malzbender. Understanding performance in coliseum, an immersive videoconferencing system. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 1, 2005.

[11] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proceedings of ACM Special Interest Group on Data Communication (SIGCOMM'02)*, 2002.

[12] F. Bauer and A. Varma. Degree-constrained multicasting in point-to-point networks. In *Proceedings of IEEE conference on computer communications*, 1995.

[13] R. Bellman. A markovian decision process. *Mathematics and Mechanics*, 6, 1957.

[14] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: high-bandwidth multicast in cooperative environments. In *Proceedings of ACM symposium on Operating systems principles*, 2003.

[15] M. Chen. Design of a virtual auditorium. *MULTIMEDIA '01: Proceedings of the 9th annual ACM international conference on Multimedia*, 2001.

[16] Y. Cui and K. Nahrstedt. Layered peer-to-peer streaming. In *International workshop on network and operating systems support for digital audio an video*, 2001.

[17] K. Daniilidis, J. Mulligan, R. McKendall, D. Schmid, G. Kamberova, and R. Bajcsy. Real-time 3d-teleimmersion. In *Confluence of Computer Vision and Computer Graphics*, pages 253–265, 2000.

[18] S. Floyd, M. Handley, J. Padhye, and J. Widmer. Equation-based congestion control for unicast applications. In *SIGCOMM '00: Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 43–56, New York, NY, USA, 2000. ACM Press.

[19] G. Franklin and J. Powell. Digital control of dynamic systems. *Addison-Wesley*, 1981.

[20] L. Gharai, C. Perkins, C. Riley, and A. Mankin. Large scale video conferencing: A digital amphitheater. In *8th International Conference on Distributed Multimedia Systems*, 2002.

[21] C. Greenhalgh and S. Benford. Massive: A collaborative virtual environment for teleconferencing. In *ACM Transactions on Computer Human Interactions*, 1995.

[22] M. Gross, S. Würmlin, M. Naef, E. Lamboray, C. Spagno, A. Kunz, E. Koller-Meier, T. Svoboda, L. V. Gool, S. Lang, K. Strehlke, A. V. Moere, and O. Staadt. blue-c: a spatially immersive display and 3d video portal for telepresence. *ACM Trans. Graph.*, 22(3):819–827, 2003.

[23] M. Hosseini and N. D. Georganas. Design of a multi-sender 3d videoconferencing application over an end system multicast protocol. In *MULTIMEDIA '03: Proceedings of the eleventh ACM international conference on Multimedia*, pages 480–489, New York, NY, USA, 2003. ACM Press.

[24] Y. hua Chu and H. Zhang. A case for end system multicast. In *Proceedings of ACM Sigmetrics*, 2000.

[25] I. Ihm and S. Park. Wavelet-based 3d compression scheme for very large volume data. *Graphics Interface*, pages 107–116, June 1998.

[26] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O'Toole. Overcast: Reliable multicasting with an overlay network. In *Proceedings of ACM symposium on Operating Systems Design and Implementation (OSDI'00)*, 2000.

[27] L. P. Kaelbling, L. L. Michael, and M. W. Andrew. Reinforcement learning: A survey. *Artificial Intelligence Research*, 4:237–285, 1996.

[28] P. Kauff and O. Schreer. An immersive 3d video-conferencing system using shared virtual team user environments. In *CVE '02: Proceedings of the 4th international conference on Collaborative virtual environments*, pages 105–112, New York, NY, USA, 2002. ACM Press.

[29] S.-U. Kum and K. Mayer-Patel. Real-time multidepth stream compression. *ACM Trans. Multimedia Comput. Commun. Appl.*, 1(2):128–150, 2005.

[30] S.-U. Kum and K. Mayer-Patel. Intra-stream encoding for multiple depth streams. In *NOSSDAV '06: Proceedings of the international workshop on Network and operating systems support for digital audio and video*, 2006.

[31] S.-U. Kum, K. Mayer-Patel, and H. Fuchs. Real-time compression for dynamic 3d environments. In *MULTIMEDIA '03: Proceedings of the eleventh ACM international conference on Multimedia*, pages 185–194, New York, NY, USA, 2003. ACM Press.

[32] T. Lewiner, H. Lopes, J. Rossignac, and A. Vieira. Efficient edgebreaker for surfaces of arbitrary topology. In *Proceedings of 17th Brazilian Symposium on Computer Graphics and Image Processing*, pages 218–225, Oct. 2004.

[33] H. Lopes, J. Rossignac, A. Safanova, A. Szymczak, and G. Tavares. Edgebreaker: A simple compression algorithms for surfaces with handles. *Computers and Graphics International Journal*, 27(4):553–567, 2003.

[34] J.-G. Lou, H. Cai, and J. Li. A real-time interactive multi-view video system. In *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*, pages 161–170, New York, NY, USA, 2005. ACM Press.

[35] W. Matusik and H. Pfister. 3d tv: a scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes. *ACM Trans. Graph.*, 23(3):814–824, 2004.

[36] J. Mulligan and K. Daniilidis. Real time trinocular stereo for tele-immersion. In *International Conference on Image Processing*, pages III: 959–962, 2001.

[37] J. Mulligan, V. Isler, and K. Daniilidis. Trinocular stereo: A real-time algorithm and its evaluation. *International Journal of Computer Vision*, 47(1-3):51–61, April 2002.

[38] S. Mürmlin, E. Lamboray, and M. Gross. 3d video fragmens: dynamic point samples for real-time free-viewpoint video. In *Technical Report No. 397*, Institute of Scientific Computing, ETH, Zurich, 2003.

[39] D. E. Ott and K. Mayer-Patel. Coordinated multi-streaming for 3d tele-immersion. In *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, pages 596–603, New York, NY, USA, 2004. ACM Press.

[40] D. E. Ott, T. Sparks, and K. Mayer-Patel. Aggregate congestion control for distributed multimedia applications. In *Proceedings of IEEE INFOCOM '04*, March 2004.

[41] T. M. Project. http://www.netlab.uky.edu/theme.html. 2001.

[42] R. Raskar, G. Welch, M. Cutts, A. Lake, L. Stesin, and H. Fuchs. The office of the future: a unified approach to image-based modeling and spatially immersive displays. In *SIGGRAPH '98: Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 179–188, New York, NY, USA, 1998. ACM Press.

[43] R. Ravi, M. Marathe, S. Ravi, D. Rosenkrantz, and H. Hunt. Approximation algorithms for degree-constrained minimum-cost network-design problems. In *Algorithmica*, 2001.

[44] G. Reynard, S. Benford, ChrisGreenhalgh, and C. Heath. Awareness driven video quality of service in collaborative virtual environment. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1998.

[45] J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, 1999.

[46] J. Rossignac, A. Safanova, and A. Szymczak. 3d compression made simple: Edgebreaker with zip&wrap on a corner table. In *Shape Modeling International Conference*, Genova, Italy, May 2001.

[47] O. Schreer, N. Brandenburg, S. Askar, and E. Trucco. A virtual 3d video-conferencing system providing semi-immersive telepresence: A real-time solution in hardware and software. In *International Conference on eWork and eBusiness*, 2001.

[48] S. Shi, L. Wang, K. Calvert, and J. Griffioen. A multi-path routing service for immersive environments. In *Workshop on Grids and Advanced Networks, in conjunction with CCGrid 2004*, 2004.

[49] G. Taubin and J. Rossignac. Geometric compression through topological surgery. *ACM transactions on Graphics*, 17(2):26–34, 1998.

[50] H. Towles, W.-C. Chen, R. Yang, S.-U. Kum, and H. F. et al. 3d tele-collaboration over internet2. In *International Workshop on Immersive Telepresence (ITP 2002)*, December 2002.

[51] H. Towles, S.-U. Kum, T. Sparks, S. Sinha, S. Larsen, and N. Beddes. Transport and rendering challenges of multi-stream, 3d tele-immersion data. In *NSF Lake Tahoe Workshop on Collaborative Virtual Reality and Visualization*, Octomber 2003.

[52] B. Wang, J. Kurose, P. Shenoy, and D. Towsley. Multimedia streaming via tcp: an analytic performance study. In *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, pages 908–915, New York, NY, USA, 2004. ACM Press.

[53] J. Wang and H. K. Huang. Medical image compression by using three-dimensional wavelet transformation. *IEEE Tran. on Medical Imaging*, 15(4):547–554, August 1996.

[54] C. Watkins. Learning from delayed rewards. *Ph.D. thesis*.

[55] R. Yang, C. Kurashima, A. Nashel, H. Towles, A. Lastra, and H. Fuchs. Creating adaptive views for group video teleconferencing - an image-based approach. In *International Workshop on Immersive Telepresence (ITP 2002)*, 2002.

[56] Z. Yang, Y. Cui, Z. Anwar, R. Bocchino, N. Kiyanclar, K. Nahrstedt, R. H. Campbell, and W. Yurcik. Real-time 3d video compression for tele-immersive environments. In *SPIE Multimedia Computing and Networking (MMCN 2006), San Jose, CA*, January 2006.

[57] Z. Yang, K. Nahrstedt, Y. Cui, B. Yu, J. Liang, S. hack Jung, and R. Bajscy. Teeve: The next generation architecture for tele-immersive environments. In *IEEE International Symposium on Multimedia (ISM2005)*, Irvine, CA, USA, 2005.

[58] Z. Yang, W. Wu, K. Nahrstedt, G. Kurillo, and R. Bajscy. Viewcast: View dissemination and management for multi-party 3d tele-immersive environments. In *ACM Multimedia (MM2007)*, Augsburg, Germany, September 23-29, 2007.

[59] Z. Yang, B. Yu, K. Nahrstedt, and R. Bajscy. A multistream adaptation framework for bandwidth management in 3d teleimmersion. In *NOSSDAV '06: Proceedings of the international workshop on Network and operating systems support for digital audio and video*, 2006.

[60] J. G. Ziegler and N. B. Nichols. Optimal settings for automatic controllers. *Transaction ASME*, 64:759–768, 1942.

Author's Biography

Zhenyu Yang was born in September, 1972. He received his Bachelor of Engineering from the Department of Computer Science and Engineering in Shanghai Jiao Tong University, 1994. He received his Master of Science from the Department of Computer Science in the University of Illinois at Urbana-Champaign, 2002. Since Spring 2004, he has joined the MONET (Multimedia Operating Systems and Networking) group as a Ph.D. student under the supervision of Professor Klara Nahrstedt, and worked on the TEEVE (Tele-immersive Environments for Everybody) project. After graduation, he will continue working with Professor Klara Nahrstedt as a postdoctoral fellow. His research interests are operating systems, networking, and distributed multimedia systems.