Supporting Scalable and Resilient Video Streaming Applications in Evolving Networks

A Thesis
Presented to
The Academic Faculty

by

Meng Guo

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

College of Computing
Georgia Institute of Technology
December 2005

Supporting Scalable and Resilient Video Streaming Applications in Evolving Networks

Date Approved: August 20, 2005

To my parents,

for their support, patience, and encouragement.

To my wife, Jing Zhou,

for her love, support and sacrifice.

ACKNOWLEDGEMENTS

My greatest gratitude goes to my advisor Dr. Mostafa Ammar for his guidance and consistent support. His knowledge, perceptiveness, and inspiring discussions has guided me throughout my Ph.D. study. I also want to express my sincere gratitude to my advisor Dr. Ellen Zegura. She provided encouragement, sound advice, good company, and lots of good ideas. It was such a pleasure to work with them for the past few years, they are always there to support me and lead me to the right direction. Without their guidance and encouragement, this thesis would not have been possible.

I would also like to thank other members of my committee, Dr. Umakishore Ramachandran, Dr. Jun (Jim) Xu, Dr. George Riley for their interests in my work. Their insightful comments have significantly affected the substance and presentation of my work.

My fellow students made my life at Georgia Tech an enjoyable experience. I wish to thank Sanjeev Dwivedi, Zongming Fei, Ruomei Gao, Christos Gkantsidis, Minaxi Gupta, Fang Hao, Qi He, Hyewon Jun, Pradnya Karbhari, Taehyun Kim, Abhishek Kumar, Richard Liston, Shashidhar Merugu, Matt Sanders, Sridhar Srinivasan, Min-ho Sung, Donghua Xu, Wenrui Zhao, Qi Zhao, Yong Zhu, Li Zou, for their friendship, encouragement, helpful discussions and all the good times we spent together.

Finally, I am forever indebted to my parents for their endless support, patience, and encouragement when it was most required. I owe special thanks to my wife Jing Zhou for her love, sacrifice and support in these years. Without their support, this thesis would not have been possible.

TABLE OF CONTENTS

DEDICATION . . . . **iii**

ACKNOWLEDGEMENTS . . . . **iv**

LIST OF TABLES . . . . **ix**

LIST OF FIGURES . . . . **x**

SUMMARY . . . . **xii**

I INTRODUCTION **1**

1.1 Server selection in replicated batching video on demand (VoD) systems . . 2

1.2 Scalable video streaming with time shifting and video patching: 5

1.3 A Cooperative patching architecture in overlay networks: 6

1.4 V3: a vehicle to vehicle video streaming architecture: 8

1.5 Outline 12

II RELATED WORK **13**

2.1 Scalable video on demand (VoD) services 13

2.2 Video streaming schemes in cooperative overlay networks 15

2.2.1 Application layer multicast protocols 15

2.2.2 Live video streaming using application layer multicast 16

2.2.3 On-demand video streaming using application layer multicast . . . 17

2.3 Video streaming in mobile ad hoc networks 18

III SELECTING AMONG REPLICATED BATCHING VOD SERVERS **20**

3.1 Batching VoD Servers 21

3.2 Server selection algorithms 23

3.2.1 Basic selection algorithms 23

3.2.2 Considering the server state information 24

3.2.3 Considering the channel allocation schemes 25

3.2.4 Dual server reception 26

3.3 Simulation setup 27

3.3.1 Simulation Environment 27

3.3.2 Performance Metrics 28

3.4 Performance Evaluation . . . 29
3.4.1 User Perceived Latency . . . 29
3.4.2 Channel Merge Rate . . . 31
3.4.3 Network Bandwidth Consumption . . . 31
3.4.4 Influence of Number of Servers . . . 33
3.4.5 Overhead of the server selection algorithms . . . 35
3.4.6 Comparison with a “Super” VoD Server . . . 37
3.4.7 Influence of Dual Server Reception . . . 38
3.5 Implementation Issues . . . 39
3.6 Summary . . . 41

IV SCALABLE LIVE VIDEO STREAMING TO COOPERATIVE CLIENTS USING TIME SHIFTING AND VIDEO PATCHING . . . 43

4.1 Design Overview . . . 44
4.1.1 Basic Operations . . . 44
4.1.2 Video Patching in Live Streaming . . . 46
4.1.3 Continuous Video Streaming . . . 49
4.1.4 An Example of System Operations . . . 51
4.2 Design Details . . . 52
4.2.1 Time Shifting Video Server . . . 53
4.2.2 Channel Allocation: . . . 54
4.2.3 Patching Stream Selection . . . 57
4.2.4 Node join algorithm . . . 57
4.2.5 Node Leave . . . 60
4.2.6 Viewing time and buffer requirement . . . 62
4.3 Performance Evaluation . . . 64
4.3.1 Simulation Environment Setup . . . 64
4.3.2 Evaluation of client video reception performance . . . 65
4.3.3 The effect of video patching on tree structure . . . 68
4.3.4 Server Channel Allocation Scheme . . . 69
4.3.5 Protocol Overhead Analysis . . . 72
4.4 Summary . . . 72

V COOPERATIVE PATCHING: A CLIENT BASED P2P ARCHITECTURE FOR SUPPORTING CONTINUOUS LIVE VIDEO STREAMING . . . 74

5.1 Cooperative patching architecture . . . 74

5.1.1 An overview . . . 75

5.1.2 Peer Design . . . 76

5.1.3 Selecting patching parents . . . 81

5.2 Performance Evaluation . . . 83

5.2.1 Parent selection algorithms . . . 84

5.2.2 Client video reception performance . . . 85

5.2.3 Comparison with time shifting server approach . . . 86

5.3 Summary . . . 87

VI V3: A VEHICLE TO VEHICLE VIDEO STREAMING ARCHITECTURE . . . 89

6.1 Problem Description . . . 90

6.2 V3: A Single Receiver Scenario . . . 92

6.2.1 Design overview . . . 92

6.2.2 Triggering the video source . . . 93

6.2.3 Transmitting data to the receiver . . . 96

6.3 V3: Streaming video to multiple receivers . . . 98

6.3.1 The impact of multiple receivers . . . 98

6.3.2 Integrated trigger message forwarding . . . 99

6.3.3 Multicast routing framework . . . 101

6.3.4 Streaming video from multiple destination regions . . . 104

6.4 Simulation environment setup . . . 104

6.5 Performance evaluation for the single receiver scenario . . . 106

6.5.1 Evaluation of trigger message forwarding . . . 106

6.5.2 Evaluation of video transmission . . . 108

6.6 Performance evaluation for multiple receivers scenario . . . 111

6.6.1 Evaluation of trigger message forwarding . . . 111

6.6.2 Evaluation of multicast routing framework . . . 113

6.6.3 Impact of multiple destination regions . . . 114

6.7 Summary 115

VII CONCLUSIONS AND FUTURE WORK 117

7.1 Summary of Contributions 117

7.2 Future Work 119

REFERENCES 122

VITA 128

LIST OF TABLES

1 A list of general cases 11

2 Node Join Algorithm 58

3 Bandwidth First Node Join Algorithm 59

4 Level Join Algorithm 60

5 Trigger message format 94

6 TMFZone Calculation 95

7 DFZone Calculation 96

8 Integrated trigger message 99

9 TMFZone calculation algorithm 100

10 DFZone Calculation Algorithm 103

LIST OF FIGURES

1 VoD Server Structure 21
2 User Perceived Latency 30
3 Channel Merge Rate Comparison 31
4 Network bandwidth consumption 32
5 User Access Latency Under Different Number of Servers 34
6 Network Bandwidth Consumption Under Different Number of Servers 36
7 Server selection overhead 37
8 Super Server Vs. Replicated Server 38
9 Influence of Dual Server Reception 39
10 Replicated VoD Service Architecture 40
11 An example of indefinite shifting 45
12 Video patching in live streaming 48
13 Continuous video streaming with initial delay 50
14 The example of system operations 51
15 Video server Design 53
16 1:1 channel allocation 55
17 Static multiplex channel allocation 56
18 Failed node discovery process 61
19 Client Side Buffer Size Under General Case 63
20 Video patching on client video reception 65
21 Client video playback performance 66
22 Video Patching on viewing performance 67
23 Viewing delay distribution 68
24 Influence of video patching on tree structure 69
25 Video Server Channel Utilization 70
26 Client Requests Queuing Delay 71
27 Peer Structure 76
28 An example for client control plane 78
29 Case 1: $r < d$ 78

30 Case 2: $d < r < 2d$ 79
31 Case 3: $r > 2d$ 79
32 Peer State Transition Diagram 80
33 Node correlation 81
34 Hit ratio comparison 84
35 List maintenance overhead comparison 85
36 Video Patching on viewing performance 86
37 Comparison with the server based approach 87
38 A sample scenario 91
39 A typical service process 92
40 Multicast Routing Approach 101
41 Highway structure for simulation experiments 105
42 Video trigger delay 106
43 Video trigger overhead 107
44 Effect of data forwarding schemes 108
45 Effect of traffic conditions 109
46 Effect of market penetration ratio 110
47 Effect of number of forwarders 110
48 Effect of trigger message integration scheme 111
49 Viewing offset comparison 112
50 Data delivery ratio comparison 114
51 Data delivery ratio 115

SUMMARY

While the demand for video streaming services has risen rapidly in recent years, supporting video streaming service to a large number of receivers still remains a challenging task. Issues of video streaming in the Internet, such as scalability, reliability are still under extensive research. Recently proposed network contexts such as overlay networks, mobile ad hoc networks pose even tougher challenges. This thesis focuses on supporting scalable video streaming applications under various network environments. This study explores the major challenges of streaming video applications under different circumstances, and provides solutions on how to improve the performance of such applications. More specifically, this thesis investigates the following problems:

- **Server selection in replicated batching video on demand (VoD) systems:** We investigate the design of server selection techniques for a system of replicated batching VoD servers. We find out that, to optimize the user perceived latency, it is vital to consider the server state information and channel allocation schemes when making server selection decisions. We develop and evaluate a set of server selection algorithms that use increasingly more information.

- **Scalable live video streaming with time shifting and video patching:** We consider the problem of how to enable continuous live video streaming to a large group of clients in cooperative but unreliable overlay networks. We design a server-based architecture which uses a combined technique of time-shifting video server and P2P video patching. This approach maintains video continuity in spite of tree disruption caused by departing clients.

- **A Cooperative patching architecture in overlay networks:** The server-based architecture can cause potential performance degradation due to server channel limitation. We design a client-based solution - cooperative patching architecture. In

this design, video patching responsibility is shifted completely to the client side, an end-host retrieves lost data from other end-hosts within the same multicast group.

- **V3: a vehicle to vehicle video streaming architecture:** We propose *V3*, an architecture to provide live video streaming service to driving vehicles through vehicle-to-vehicle networks. V3 incorporates a novel signaling mechanism to continuously trigger video sources to send video data back to the receiver. It also adopts a *store-carry-and-forward* approach to transmit video data in a partitioned network environment.

- **Multicasting in the vehicle-to-vehicle video streaming (V3) architecture:** We develop a multicasting framework that enables live video streaming applications from multiple sources to multiple receivers in vehicle-to-vehicle (V2V) networks. In this framework, a message integration scheme is used to suppress the signaling overhead, as to the video data forwarding, a two-level tree-based routing approach is adopted. A set of multicast routing algorithms based on different knowledge oracles are used. To support video streaming from multiple destination regions, we design a set of scheduling schemes that use increasing amount of knowledge.

CHAPTER I

INTRODUCTION

While the demand for video streaming services under various circumstances has risen rapidly in recent years, supporting video streaming service to a large number of receivers still remains a challenging task. Due to the bandwidth-intensive nature of video streams, the straightforward solution of setting up a connection for every single request is obviously not scalable. The server-side link can be easily congested as the client requests accumulate. Server replication is an approach that allows a streaming video service to handle a large number of clients, albeit at the additional cost of providing more servers. In such a system, servers maintaining the same videos are placed in multiple locations in the network. A client's request can be satisfied by the server which is closer to this client and has the available bandwidth resource. For example, commercial content delivery networks (CDNs) such as Akamai use dedicated infrastructure to support scalable delivery of live streams [1]. Another approach that has been investigated extensively is to serve clients requesting the same video using one multicast stream. This approach has the advantage that it can save server resources as well as server access and network bandwidth, thus allowing the server to handle a large number of customers [59].

One recently proposed approach relies on the cooperation of video clients in forming an overlay network over which the video is propagated [46, 63, 73]. This approach is scalable in the sense that the forwarding capability of the overlay network can grow incrementally. New clients joining the network also bring in extra bandwidth capacity to the system. End-hosts in the overlay network are *cooperative* in the sense that they share data and exchange group control information. On the other hand, they are also *unreliable.* First, these nodes are potentially more vulnerable subject to attacks than the dedicated routers. Furthermore, they are less stable than routers since overlay nodes can join/leave the group at any time. For video streaming applications over an overlay network, the major problem caused by the unreliability of end-hosts is *video discontinuity*. When an interior node leaves the multicast group without informing others, its children nodes will be disconnected and their video reception will be interrupted. Although the disconnected clients can resume the service by rejoining the multicast tree, this process can take time and can result in loss of video content and interruption of video playback.

With the advance of wireless communication and mobile computing techniques, research work on streaming video applications over mobile ad hoc networks (MANETs) has drawn a lot of attentions. Wireless ad hoc networks pose new challenges to video streaming applications. 1) Frequent topology change: due to the mobility of wireless nodes, the topology of an ad hoc network may change often. Thus the established connection route between a source and a destination is likely to be broken during the period of the transmission, which may cause interruption, or pause in the received video packets. 2) High transmission error: the availability and quality of a link further fluctuates due to channel fading and interference from other transmitting users [62]. This problem is aggravated when there are multiple wireless hops between the source and the destination. 3) Partitioned network structure: in a sparse ad hoc network, the network is more likely to be partitioned and this partition can last a significant period of time. In a partitioned network, classic MANET routing schemes that require an end-to-end path between the source and the destination will fail to deliver packets.

This research seeks to investigate the major challenges of streaming video in evolving and challenging network environments. Our objective is to design appropriate video streaming techniques according to different network structures, and to develop schemes to compare and evaluate these video streaming mechanisms. Specifically, this research focuses on the following issues.

1.1 Server selection in replicated batching video on demand (VoD) systems

A Video-on-Demand (VoD) service offers a large selection of videos from which customers can choose. The service is *on-demand* because customers expect to receive their videos *soon* after they issue their requests to the server. Designers of VoD systems, therefore, strive to achieve low *access latency* for customers. One of the main challenges in the design of a VoD system is how to allow the system to handle a large number of customers, maintain a low cost of operation and at the same time provide for acceptable access latency.

One approach that has been investigated allows the server to *batch* clients requesting the same video and to serve clients in the same batch with one multicast video stream [2, 3, 19, 34]. This approach has the advantage that it can save server resources as well as server access and network bandwidth, thus allowing the server to handle a large number of customers. VoD data delivery schemes can be broadly classified into two categories: *periodic broadcast* and *scheduled multicast* [21]. Periodic broadcast is also referred to as data centered approach. In this approach, the server channel is dedicated to video objects and is broadcasting repeatedly [21, 2, 75, 44]. Scheduled multicast, also called user centered approach, dedicates channels to individual users. When the server channel is free, the server selects a batch of clients to multicast according to some scheduling policy [10, 19, 23, 32, 33]. The server selection problem is more interesting for the scheduled multicast approach, since the server behavior is based on client demand. In this thesis, we focus on scheduled multicast schemes only.

VoD *server replication* is another approach that allows a VoD service to handle a large number of clients, albeit at the additional cost of providing more servers. In such a system, servers maintaining the same videos are placed in multiple locations in the network. The main question when server replication is employed is that of server selection, i.e., how to direct clients to the *best* server. There has been a significant amount of research recently regarding server selection issues [9, 26, 74, 83], with a focus on unicast services in which a client is served by a single point-to-point flow from a server. The only prior investigation of multicast server selection that we are aware of is the work reported in [27, 28, 29]. That work, however, considers servers that are continuously transmitting multicast data (as would be the case in a live video or audio broadcast). In such a system, the only important performance metric is overall network bandwidth consumption, and there is no consideration of client latency or server access bandwidth.

In this thesis, we study the design of a VoD service in which a set of batching and multicast servers are replicated. Our objective is to investigate the benefits of replication for such a service. Our work focuses on the design of server selection techniques for such a replicated service and the effect of the server selection approach on the performance of a replicated batching-server VoD system. Because the VoD servers use multicast, the issues inherent in the design of a server selection algorithm differ in important ways from unicast server selection work. For example, when deciding to allocate a new client to a multicast server, it is important to take into account other clients that were previously assigned to the same server and will ultimately be batched with the new client. Our work differs significantly from previous multicast server selection problem as well. User access latency is the most important performance metrics in VoD service, this in turn is determined by the server disk bandwidth, batching status and batch scheduling policy of the server. To achieve desirable user access latency, the state of VoD servers (defined by the video channel availability, and the batches it is currently maintaining) needs to be considered in making server selection decisions.

One of the main motivations for replicating a server is to provide a client with the requested video program in shortest time possible. At the same time, it is desirable to serve client's requests by relatively closer server, so as to reduce the total network bandwidth required, and possibly to improve the quality of the received video. We also want to distribute the VoD traffic across the network to prevent *hot spot* from generation. Our work considers selection algorithms as they would be applied to three batching approaches: Batching with Persistent Channel Allocation [2], Patching [45], and Hierarchical Multicast Stream Merging (HMSM) [23]. In this work, we explore a range of selection techniques that incorporate, to various degrees, knowledge of the batching and allocation schemes of the servers.

1.2 Scalable video streaming with time shifting and video patching:

The problem of how to enable *live streaming* of video content from a single server to a large number of clients has always been challenging. IP layer solutions had limited success in such an area. Unicast solution is not scalable since the server side link can be easily congested as the client requests accumulate. IP multicast is an ideal communication paradigm, but it is not widely deployed due to technical and non-technical reasons. Application layer multicast is a recently proposed approach to support scalable video streaming services without infrastructure support. In application layer multicast, video clients form an overlay network over which the video is propagated. In this approach, a client currently in the overlay network forwards the content it is receiving, and serves other client's request as a server. By distributing the transmission load to the clients all over the network, the video server is no longer the bottleneck. This approach is scalable in the sense that the forwarding capability of the overlay network is growing incrementally. New clients joining the network also bring in extra bandwidth capacity to the system.

One of the major problems for application layer multicast is the video discontinuity problem caused by the dynamics of membership. Since the clients in the group can leave at any time, other clients which are receiving video content from them have to suffer service disconnection. Although the disconnected clients can resume the service by rejoining the application layer multicast tree, this process can take time and can result in loss of video content and interruption of video reception. To make things worse, unsatisfied clients leaving the group can become a positive feedback process, causing more clients to leave, which ultimately makes the streaming media service unacceptable. In CoopNet [63], video continuity is maintained using multiple description coded (MDC) streams multicast over several application layer trees. CoopNet employs a server-based centralized control protocol, the server is responsible to make sure that multiple trees for different descriptions are uncorrelated with each other. A single client's departure can only disconnect a subset of descriptions for its children. While CoopNet maintains video continuity, it can cause video quality fluctuation as clients depart and trees are reconstructed around them. Recent work in CoopNet shows the PSNR variation with different number of descriptions [64].

Our solution tries to provide continuous streaming service without video quality fluctuation. In this thesis, we develop a scheme using the transmission of a *single description coded* video over an application layer multicast tree formed by cooperative clients. Clients in the tree always receive full quality video stream. Video continuity is maintained in spite of tree disruption caused by departing clients using a combination of two techniques: (1) providing time-shifted streams at the server and allowing clients that suffer service disconnection to join a video channel of the time-shifted stream, and (2) using video patching to allow a client to catch up with the progress of the live video program. We also need a buffering scheme that allows the client to store the video packets, and to playout when needed.

In our design, the client buffers the initial part of the video content for a certain time before play out. If the client is disconnected from the multicast tree, it can play out full quality video from this buffer while reconnecting to the group. The client rejoins the group by connecting to both the original stream and a time-shifted stream. The time-shifted stream is used to retrieve the missed video. The original stream is used to catch up with the progress of the live video. Received video packets that are not being played out are stored at the buffer for future playout. When the missed portion of the video content is fixed, the time-shifted stream is released, and the client receives from the original stream only. The use of video patching requires that a client should have enough bandwidth for two video streams: the original stream, and the time-shifted stream.

1.3 A Cooperative patching architecture in overlay networks:

We have developed a server-based scheme that achieves continuous video streaming under unreliable overlay network [38]. This scheme uses a combined approach of time-shifted video server and peer to peer video patching technique. There are two major limitations in this approach: i) potential performance degradation. Video server is the only point to provide the patching stream for a client, patching stream uses significant amount of server side bandwidth. In the case where multiple patching requests are received, the server might have to delay some of the requests if there is no free channel available; ii) server design complexity. The scheme requires a specially designed video server, it not only has to broadcast the live video, but also has to carry out the functionality of an on-demand video server. It has to process video patching requests, serve time-shifted video stream on demand, set up and tear down the transient trees of patching stream.

In this thesis, we propose a *cooperative patching* architecture to recover lost video packets and to achieve continuous video playback at client side. In *cooperative patching* architecture, video server multicasts live video to receivers through an application layer multicast tree. When a client joins the multicast group, it caches the initial portion of the video that has been received. Instead of immediate playback, it delays the start of playback for a certain period. It also keeps the video that has been played out for another period before discarding it. In this design, each end host maintains a list of patching parents. A patching parent of a client refers to an end host which provides patching stream to this client. Any end host currently in the multicast group can be a potential patching parent. When a client is disconnected from the tree due to ancestor node failures, it rejoins the tree by sending two signals: it sends a "patching" request to one of its patching parent. The selected patching parent then streams the video to recover the lost content. It also sends a "rejoin" request to join the tree and receives the original video from a normal parent to keep up the progress of video reception. A normal parent of a client denotes the node which forwards original video to it. When the missed video content is fully recovered from its patching parent, the client is disconnected from its patching parent, and begins to receive from the original stream only.

This scheme relieves server load by completely shifting video patching responsibility to the client side. In *cooperative patching*, video contents are replicated in multiple locations across the overlay network to provide fast and timely data recovery. Client that suffers service disconnection relies on other clients in the same multicast tree to recover the missed video. A set of peer selection algorithms are proposed to locate appropriate end hosts for a client to recover the lost data. This scheme is especially advantageous for legacy video systems, since it does not require modification of the video server, the server is only responsible for broadcasting video.

1.4 V3: a vehicle to vehicle video streaming architecture:

Streaming live video content to driving vehicles can provide a very attractive service. Many applications can benefit from such a service:

1. Driver assistance and safety applications. In case of a car accident or road work, if vehicles that are driving towards this area can receive live video about this area, the drivers can then drive with alert or simply choose another route.

2. Business or entertainment applications. Road side businesses, such as hotels and restaurants, can use content-rich video streams to broadcast advertisements to drivers on the road.

3. Military or scientific applications. Vehicles on duty can broaden their view by receiving video from other vehicles' video cameras. Also, it is very useful to stream every vehicles' video back to a command center for information processing or decision making purposes.

A widely used approach to broadcast traffic conditions to driving vehicles is to set up bulletin boards on certain spots of a major highway, and display traffic conditions in several lines of text. This approach can provide traffic information to some extent, but it is far from satisfactory: i) it may suffer extensive delay, and sometimes fails to deliver valid information; ii) it is not customizable for different drivers on the road, every driver receives the same information; iii) within just several lines of text, it can only provide very brief information, which might not be enough for drivers to make appropriate decisions; iv) very limited coverage area, this service can only reach vehicles that drive past these bulletins. Imagine a scenario where a vehicle is equipped with an LCD screen at the front panel which displays video information from a remote region. Note that this screen serves similar functions as the rear view mirror. But, instead of displaying traffic conditions behind the vehicle, this screen can display traffic conditions from any spot on the road. With a glimpse of this screen, the driver can obtain desired information without interrupting normal driving. As to the media that carries the traffic information, we use video instead of plain text for the following reasons: i) video provides content rich information, while text can only provide very brief descriptions; ii) text needs human intervention as it often needs people to capture, organize and summarize to generate descriptive text information; iii) video needs shorter time to understand, a glimpse of the video screen is enough to grasp the road conditions, while reading information like *"Car accident on North Ave., 1 mile south of exit 250 on I-75, 3 left lanes blocked..."* takes significantly longer time; iv) Transferring text information needs reliable transmission protocols, while video data can tolerate a certain amount of packet loss.

Two basic approaches can be used to support video streaming to vehicles on the road: an *infrastructure-based* approach, and a *vehicle-to-vehicle (V2V)* approach. In the infrastructure-based approach, video cameras as well as video servers are deployed across the road network. These video servers are connected through a network of base stations or simply through the Internet. When a vehicle at location A wishes to receive video information about location B, it sends a request to a nearby server s_A. This request is then forwarded to video server s_B at location B through the infrastructure. Upon receiving the request, s_B then sends data back to s_A, the receiver can obtain data from s_A. As a vehicle is driving on the road, it reports its geographic location information to the base stations nearby, and the video server s_B can adjust the data destination accordingly. While this approach can provide streaming video service to certain vehicles, it suffers from high base station deployment and maintenance cost. Furthermore, it could be extremely difficult and sometimes unnecessary to cover every part of the entire road network.

In this work, we are interested in investigating the problem of supporting a video streaming service using a vehicle-to-vehicle (V2V) approach. A V2V network is a special mobile ad hoc network in which each mobile node is a vehicle with wireless capability. Vehicles in a V2V network are equipped with on-board computing and wireless communication devices. It is also assumed that the vehicles are equipped with GPS devices which enable the vehicles to track their location and trajectory. Some vehicles are equipped with video cameras, as well as enough storage to store recorded video for a certain time, and are therefore, able to capture and transmit live video from their surroundings. Other vehicles driving on the road can request to receive video from the camera-equipped vehicle.

Video streaming through a V2V network is practical with the advance of computing technique, wireless communication and video compression techniques:

1. Wireless communication techniques: the IEEE 802.11 standard supports data transfer rates at up to 54Mbps. Even between high speed driving vehicles, it's reasonable to expect a 1Mbps data rate [71].

2. Video compression techniques: if we assume a 320*240 screen with 65536 color, a frame rate of about 15fps (web-based video streaming quality), the widely-used MPEG compressed video stream rate is estimated to be about 100 to 150kbps [31]. So, there is enough bandwidth to support video streaming between vehicles.

3. Computing capability: each vehicle can support relatively large computational and storage resources. It is reasonable to assume that each mobile node has Laptop computing and storage capability.

4. Power support: ample power can be supplied from the engine of the vehicle. With enough power and large computational and storage capability, it is possible to explore data intensive streaming video service.

Video streaming in V2V networks is challenging for several reasons.

1. Network partitions: Since each vehicle's radio range is only about two hundred meters, a V2V network may not be connected at all times. Furthermore, due to the gradual market penetration, only a fraction of the vehicles on the road will be equipped with communication capability. We call the fraction of vehicles that are so equipped the *penetration ratio*. Only such vehicles can participate in the V2V system. A low penetration ratio can cause a V2V network to be constantly partitioned [78, 79], and consequently, traditional mobile ad-hoc routing schemes can not be used in such a network.

2. Dynamic conditions: Vehicles on a highway or a main road can drive at speeds of $40Mph$ to $60Mph$. If two vehicles are driving in opposite directions, their wireless

connection lifetime could be very short. It is also highly possible that two vehicles meet each other only once during the streaming video service process. Thus, a receiver vehicle can not rely on a fixed set of neighbor vehicles for data delivery.

3. Multiple mobile video sources: In many cases, video data is generated from multiple vehicles. Each vehicle is only responsible for a small portion of the data. These vehicles tend to collect data in an uncoordinated way, data redundancy and data discontinuity is inevitable. Also, because of this random data collection and network partitioning, it is highly possible that video data arrives at a receiver out of order. This requires even longer delays and more buffering at the receiver for continuous playback.

Table 1: A list of general cases

Number of Receivers	**Number of Destination Regions**
Single	Single
Multiple	Single
Multiple	Multiple

We propose *V3*, an architecture to support video streaming applications in V2V networks. This architecture is composed of two parts: a *video source trigger sub-system* and a *video data transfer sub-system*. The video source trigger sub-system uses a novel signaling mechanism to continuously trigger video sources to send video data back to receivers. The video data transfer sub-system adopts a *store-carry-and-forward* approach to transmit video data in a partitioned network environment [12, 24, 25]. Several algorithms are proposed to balance the video transmission delay and bandwidth overhead. In this thesis, we first describe our design of V3 by focusing on a special case where a single receiver is interested in a single destination region. A more general case involves both multiple receivers and multiple destination regions, as shown in Table 1. This thesis then studies the impact of multiple receivers on the system design, and proposes a multicasting framework to support a streaming video service from multiple destination regions to multiple receivers.

1.5 Outline

The rest of the thesis is organized as follows. Chapter 2 discusses related work in the area of focus. Chapter 3 presents a set of server selection schemes for replicated batching VoD systems. In chapter 4, we propose a combined approach of time-shifted server and video patching to support continuous video streaming in overlay networks. In chapter 5, we then propose the design of cooperative patching which achieves continuous video streaming without the support of a specially designed video server. The techniques that enable a live video streaming service in V2V networks is discussed in Chapter 6. We finally summarize this thesis in Chapter 7.

CHAPTER II

RELATED WORK

This chapter provides an overview of related work in the areas of scalable video on demand services, video streaming in peer to peer and overlay networks, and video streaming in mobile ad hoc networks.

2.1 *Scalable video on demand (VoD) services*

A typical Video-on-Demand (VoD) service allows remote users to play back any one of a large collection of videos at any time. Upon receiving a client's request, a VoD server delivers the video to the client in a single video stream. For each video stream, sufficient storage, I/O bandwidth as well as network bandwidth must be available for continuous transfer of data from the server to the clients. One major performance metrics is *access latency*, since customers expect to receive their videos *soon* after they issue their requests to the server. In the case where a large group of clients requesting a same VoD service, using unicast to deliver video stream is apparently unacceptable. The server side bandwidth can be easily congested, newly generated requests have to wait until the previous requests are fully served. Thus, one of the main challenges in the design of a VoD system is how to allow the system to handle a large number of customers, maintain a low cost of operation and at the same time provide for acceptable access latency.

One widely used scheme for providing scalable VoD service is to use batching VoD servers and to serve client requests for the same video with one multicast stream. There are two approaches to achieve video batching: periodic broadcast, and scheduled multicast. In this section, we briefly overview the related research work in these two approaches.

• **Periodic broadcast:** the videos are broadcast periodically, a new video stream starts for every B minutes for a given video [44]. Thus, the worst service latency experienced by any subscriber is guaranteed to be less than B minutes independent of the current number of pending requests. In periodic broadcast, the key problem is *data segmentation*, i.e., how to segment a video file so that the average client access latency can be minimized. One of the earlier periodic broadcast schemes was proposed by Dan, Sitaram and Shahabuddin [19], this approach broadcasts a given video every batching period, the service latency can only be improved linearly with the increases in the server bandwidth. To significantly reduce the service latency, Pyramid Broadcasting (PB) technique was introduced [75]. In this scheme, each video file is partitioned into segments of geometrically increasing size; and the server capacity is evenly divided into K logical channels. The i_{th} channel is used to broadcast the i_{th} segments of all the videos in a sequential manner, Since the first segments are very small, they can be broadcast a lot more frequently through the first channel, This ensures a smaller wait time for every video. Aggarwal, Wolf and Yu proposed a technique called Permutation-Based Pyramid Broadcasting (PPB) [2]. PPB is similar to PB except that each channel multiplexes among its own segments (instead of transmitting them serially). Hua and shen proposed a Skyscraper Broadcasting (SB) [44] technique which limits the maximum segment size, and in turn achieves a better user access latency and better storage usage efficiency.

• **Scheduled multicast:** A batching VoD server that uses scheduled multicast generally operates in two phases: *batch scheduling* and *channel allocation*. Batch scheduling decides which batch should be selected when there is a free channel available. Channel allocation deals with whether the channel is used to transmit the whole video program or only a portion of it. For the batch scheduling phase, many scheduling policies have been proposed [20, 19, 2]. The *first come first serve (FCFS)* approach schedules the batch whose first client comes the earliest; The *maximum queue length first (MQL)* approach schedules the batch with the largest batch size; The *maximum factored queue length (MFQ)* approach [2] selects the batch with the maximum value of $q_i/\sqrt{f}_i$, where q_i is the queue length (number of clients in the batch) for video i, and f_i is the relative frequency of the arrivals of video i. For the channel allocation phase, video patching is the most widely adopted channel allocation technique [10, 23, 34, 45]. In video patching, video channels of the VoD server are classified as regular channels and patching channels. Regular channels transmit the entire video stream, while patching channels only transmit the initial portion of a video as needed. Clients that request the video can join the patching channel for low access latency, and at the same time, joins one on-going regular channel to catch up the video progress. When the missing portion is completely received, the patching channel is released, and the clients only receive packets from the regular channel. Video patching technique requires client bandwidth at least two times of the video streaming rate, and an extra client side buffer.

2.2 *Video streaming schemes in cooperative overlay networks*

Due to the bandwidth-intensive nature of video streams, deploying scalable streaming service with acceptable quality has always been a challenging task. IP Multicast is the most efficient way to perform group data distribution, as it is able to reduce packet replication on the wide-area network to the minimum necessary. However, IP Multicast is not widely deployed due to a variety of technical and non-technical reasons. Recently, the idea of application layer multicast has been proposed as a new approach to implement wide-area multicast services. In this approach multicast functionality is implemented at the end-hosts instead of network routers. Unlike network layer multicast, application layer multicast requires no infrastructure support and can be easily deployed in the Internet. In this section, we first describe a set of application layer multicast protocols that have been proposed recently, we then describe some video streaming solutions based on application layer multicast.

2.2.1 Application layer multicast protocols

In application layer multicast, the group members, i.e. the end hosts, are organized as an overlay network. Generally, there are two topologies in this overlay network: control topology and data topology. Control topology is used for group members to exchange group control information, and to adapt the network structure upon client join/leave. Data topology is used to transmit the real data traffic among group members. Based on how this topology is constructed, we classify the different application layer multicast protocols as:

mesh-based protocols, and tree-based protocols.

- **Mesh-based protocol:** In the mesh-based approach, group members first organize themselves into an overlay mesh, then a data delivery tree rooted from at the source node can be created over the mesh. Protocols such as NARADA [13], NICE [6], LARK [51], etc.

- **Tree-based protocol:** protocols based on the tree-first approach first construct a shared data delivery tree directly. For robustness considerations, additional control links can then be added to this tree. Such protocols include YOID [30], Overcast [48], AOM [77], HMTP [80], etc.

2.2.2 Live video streaming using application layer multicast

In live video streaming applications using application layer multicast, there is a reliable video source, the video server, which broadcasts live video content to the network. A video client wishes to receive the video content from this server then joins the application layer multicast tree rooted at the video server. Video content is then propagated along this application layer multicast tree to the video clients [13, 14, 42, 46, 53, 73]. One major problem for live video streaming using application layer multicast is the video discontinuity caused by the dynamics of membership. Since the clients in the group can leave at any time, other clients that are receiving video content from them have to suffer service disconnection. Although the disconnected clients can resume the service by rejoining the application layer multicast tree, this process can take time and can result in loss of video content and interruption of video reception. Although approaches like adding redundant links, adding hierarchies with additional group leaders, can reduce the impact of group dynamics, this problem is not solved.

One solution to address video discontinuity problem in overlay video streaming is to encode video stream into multiple sub streams and to feed a receiver from multiple senders. In CoopNet [63, 64], video continuity is maintained using multiple description coded (MDC) streams multicast over multiple application layer trees. These trees are constructed such that each end-host appears as an interior node in exactly one tree [11]. A single client's departure can only disrupt one tree. While CoopNet maintains video continuity, it can cause video quality fluctuation as clients depart and trees are reconstructed around them. Recent work in CoopNet shows this quality fluctuation effect with varying number of descriptions [64]. PALS [68, 69] is a receiver-driven approach for quality adaptive playback of layer encoded streaming media from a group of congestion controlled sender peers to a single receiver peer. In PALS, the receiver adaptively determines a subset of senders that maximize overall throughput, and number of layers that can be delivered from these senders. Cui, et.al [18] has proposed a peer-to-peer streaming solution to address the on-demand media distribution problem, in which they claim their solution is optimal at maximizing the streaming quality of heterogeneous peers, scalable at saving server bandwidth. and efficient at utilizing bandwidth resource of supplying peers.

2.2.3 On-demand video streaming using application layer multicast

For on-demand video streaming applications, a video client wishes to receive complete video content from the video source. Thus, different client joining the same application layer multicast tree in different time slots might receive data asynchronously.

Jin et. al. proposed a cache-and-relay streaming media delivery for asynchronous clients [49]. In his work, clients cache data received from the server, and future requests are satisfied by the client already has the data and is willing to share. His work focuses on scalability of on-demand streaming applications, while our work tries to support resilient live video streaming in application layer multicast. End hosts in his work are responsible for distributing the data, while in our work they are used to recover the lost data. BitTorrent [15] is another example in which clients cooperate to facilitate large scale file downloading. With BitTorrent, end hosts that are downloading the same file also upload pieces of the file to each other. BitTorrent is used for file downloading with no requirement on latency, while our scheme is used for real time video streaming. What's more, BitTorrent downloads pieces of file in random order, our scheme keeps strict ordering.

P2Cast is another peer-to-peer approach to support scalable VoD service [42]. In P2Cast, clients arriving close in time (within a threshold) form a session. For each session, the server, together with the P2Cast clients, form an application layer multicast tree. The entire video is streamed over the application layer multicast tree. For clients who arrive later than the first client in the session and thus miss the initial part of the video, the missing part can be retrieved from the server or other clients that have already cached the initial part. A Best Fit (BF) algorithm is developed to construct the application layer multicast tree, as well as select the patch server to serve the missing part of the video.

2.3 Video streaming in mobile ad hoc networks

Streaming video applications that operate over wireless links pose a lot of new challenges to the networking community [62, 76]. 1) Reduced video packet transmission reliability due to higher bit error rate of a wireless link. 2) Unreliable end-to-end data path since mobile end-hosts can move out of each other's radio range at any time. 3) Potentially longer delay due to high propagation delay. 4) Limited energy for mobile hosts, mobile hosts rely on battery for power supply, the data intensive feature of video streaming applications poses a critical challenge for such hosts. Efforts that have been made to make video streaming over wireless networks possible includes:

1. Improving the data transmission infrastructure: IEEE 802.11 was originally developed as a replacement of Ethernet by providing up to 2 Mbps transmission rates. The state-of-the-art 802.11 now supports up to 54 Mbps rates at much lower cost. Enhancements in this wireless Ethernet technology also include QoS, security, and mobility supports. On the other hand, Internet is well-developed and provides reliable data transmission service. Strategically deployed base stations can take this advantage, and improve the video transmission performance;

2. Advanced video encoding techniques: If we assume a 320*240 screen with 65536 color, a frame rate of about 15fps, the widely-used MPEG compressed video stream rate is estimated to be about 100 to 150kbps [31]. More efficient source coder provides more scalable in both time and space, and low complexity [43, 52, 57, 72]. Error-erasure channel coding techniques are also used to recover the bit rate error [60, 70]. At last,

to handle the path instability, video contents are encoded into multiple descriptions and be sent to the receiver from diversified paths are also used [4, 35, 62].

3. Routing protocol suits for wireless communication: In ad hoc networks all nodes are mobile and can be connected dynamically in an arbitrary manner. All nodes of these networks behave as routers and take part in discovery and maintenance of routes to other nodes in the network. Many routing algorithms have been developed for ad hoc networks [47, 50, 65, 66, 67]. These routing protocols can be divided into two categories: table-driven and on-demand routing. In table driven routing protocols consistent and up-to-date routing information to all nodes is maintained at each node whereas in on-demand routing the routes are created only when desired by the source host. These routing protocols assume the ad hoc networks are fully connected, and are vulnerable to network partitioning. New routing protocols, which are based on store-carry-and-forward paradigms, are designed for such partitioned networks [24, 25, 81, 82].

CHAPTER III

SELECTING AMONG REPLICATED BATCHING VOD SERVERS

To deploy scalable VoD service, two approaches are generally used: server replication and multicasting. In this thesis, we study the design of a VoD service in which a set of batching and multicast servers are replicated. Our objective is to investigate the benefits of replication for such a service. Our work focuses on the design of server selection techniques for such a replicated service and the effect of the server selection approach on the performance of a replicated batching-server VoD system.

Besides server load distribution and server client distance reduction, two factors are vital to achieve desirable performance for the server selection scheme: server state information, and channel allocation scheme. The state of a VoD server is defined by the video channel availability, and the batches it currently maintains. Selecting a server who has the most number of free channels or whose batch ranks highest in MFQ queue of this server almost always leads to minimized user access latency. When different channel allocation schemes are used, i.e. persistent channel allocation, video patching, and hierarchical multicast stream merging, server selection algorithms have to take this factor into consider to achieve minimal user access latency. For example, a server allocate a *patching channel* whose lifetime is only a portion of a whole video program can expect to reuse this channel much earlier than a *regular channel*.

In this chapter, we first describe a typical VoD server structure, and describe the batch scheduling and channel allocation schemes in detail. We then propose a set of server selection algorithms which incorporates increasing amount of knowledge. We evaluate our design by simulation and discuss some implementation issues at last.

3.1 Batching VoD Servers

A VoD server receives requests from clients and schedules the transmissions of videos over a set of channels. For a *batching* VoD server, a video transmitted on a particular channel is multicast to the clients in the particular batch. There is typically limit on how many videos can be transmitted simultaneously from a single video server, for example, due to limited server disk access bandwidth.

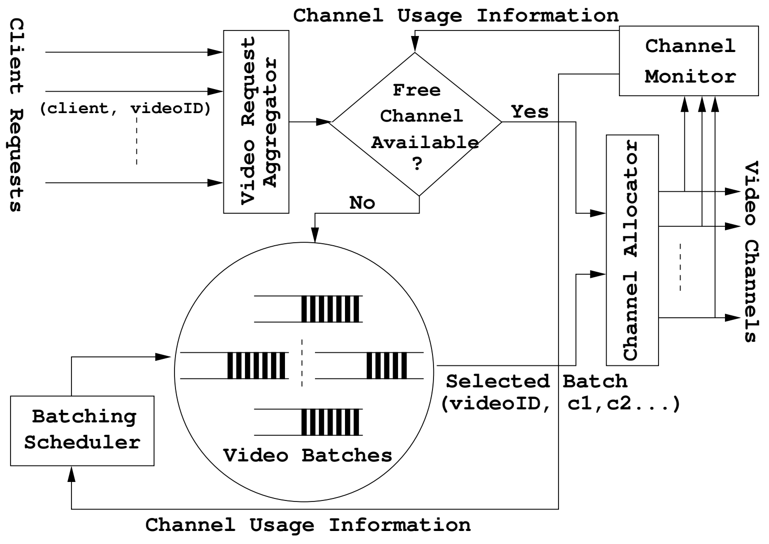


Figure 1: VoD Server Structure

Figure 1 illustrates a typical batching-VoD server structure. When the server receives a client request, it will schedule the client request immediately if there is a free channel available. Otherwise, the client request will be put into a video batch, indexed by the video stream ID. Whenever a channel becomes available, the server will select a batch and allocate the channel to this batch for transmission. Video transmitted over a particular channel is multicast to the clients in the particular batch.

A batching VoD server operates in two phases: *batch scheduling* and *channel allocation.* Batch scheduling decides which batch should be selected when there is a free channel available. Channel allocation deals with whether the channel is used to transmit the whole video program or only a portion of it. For the batch scheduling phase, we assume that all the

VoD servers use MFQ as the batch scheduling policy. Since MFQ approach yields excellent empirical results in terms of latency, throughput and fairness. As to the channel allocation phase, we study three channel allocation policies in this thesis:

- The *persistent approach* [19, 2]: Once the channel is allocated to a batch of clients, it will continue to be used for multicasting the entire video to the batch. This channel allocation scheme is simple and straightforward, it only requires the bandwidth of a single video stream, and it does not need extra client side buffer to store temporary video content.

- The *video patching approach* [45, 23, 10, 34]: This channel allocation technique is designed to provide for better server channel utilization and thus lower user-perceived access latency than the persistent allocation scheme. In video patching, video channels are classified in two categories: regular channels and patching channels. Regular channels transmit the entire video stream, while patching channels only transmit a portion of a video as needed. When the server allocates a free channel to a batch of requests (according to some batch scheduling policy), the server first scans all the on-going channels. If there is no regular channel distributing the desired video, or the starting time of the latest regular channel which is transmitting the desired video is too early for the client to patch, this channel is allocated as a regular channel. Otherwise, the channel is allocated as a patching channel. In this case, the clients first receive data from both the regular channel and the patching channel. The data from the patching channel is used to make up for the data the clients are missing from the regular (already on-going) channel. While receiving from the patching channel, the data being received from the regular channel is buffered by the clients for playout when needed. When the missing portion is completely received, the patching channel is released, and the clients will be merged into the regular channel.

- The *hierarchical multicast stream merging (HMSM) approach* [23]: This approach attempts to capture the advantages of dynamic skyscraper [21], as well as the strengths of stream patching [45]. In HMSM, clients that request the same file repeatedly merge into larger and larger groups, leading to a hierarchical merging structure. In this way, the required server bandwidth can be further reduced by repeatedly merging channels. In ordinary video patching, only regular channels can be merged into, while in HMSM, there is no discrimination between regular and patching channels. All the previous channels that are within a "patchable" range can be merged into. When there are multiple previous channels within the patchable range, we use closest target (CT) scheme to decide which earlier channel to merge into. This scheme simply chooses the closest earlier stream still in the system as the next merge target [22].

The first channel allocation policy requires client access bandwidth to be equal or larger than the video streaming rate. The latter two channel allocation approaches require client bandwidth at least two times of the video streaming rate, and an extra client side buffer.

3.2 Server selection algorithms

We are concerned in this paper with a scenario in which a set of replicated batching VoD servers are distributed across the network. We assume that all the servers carry the same set of videos and employ the same batch scheduling and channel allocation policies. A large set of clients make requests for video and we are concerned with the question of how to best direct a client's request to one of the replicated servers.

Two types of information are potentially useful in designing a server selection algorithm: information about *topology* (e.g., the number of hops from a client to a VoD server) and information about the *servers* (e.g., size of the MFQ queue for the requested video). We propose six server selection algorithms that use increasingly more information to make a decision.

3.2.1 Basic selection algorithms

In this section, the server selection algorithm selects a video server based on the topology information only.

- The first option, the *Closest-server-first* algorithm, selects the server which is closest to the client using the network hop count measure. This provides a baseline illustrating the performance in the absence of any information about the servers.

- The second option adds a minimal amount of information about the servers. Specifically, the *Optimized closest-server-first* algorithm selects the closest server among those with free channels. If no servers have free channels, the closest server is selected.

3.2.2 Considering the server state information

We next turn to methods that are aware that the VoD servers are using MFQ scheduling. In each of these methods, the client always prefers a server with free channels. However, if there is no such server, these methods aim to select a server that will schedule the client request earliest.

- The *Register-all* algorithm will put the client request into the corresponding queue at all of the video servers. When the client request is satisfied at any one server, the request is withdrawn from the other server queues. This method loads all of the servers with the request and requires the ability to withdraw a request from a server. On the other hand, it is quite simple from the client viewpoint.

- As an alternative, the *Maximum-MFQ-rank-first* algorithm computes the destination queue *rank* at each server and sends the client request to the queue with the best MFQ rank. That is, best MFQ rank is used as a heuristic to select a server that will reduce the latency to serve the client request. This method uses more information than Register-all, but avoids sending the request to all servers and then withdrawing.

- Our *Minimum Expected Cost (MEC)* algorithm is more sophisticated. It makes use of the number of free channels per server, the hop count to each server, and the MFQ value at each server. Shorter access latency and less bandwidth consumption lead to a smaller expected cost. The *expected cost* measure is a weighted sum of three terms, and the client selects the server with smallest expected cost. The first term is proportional to the inverse of the number of channels; the second term is proportional to the MFQ value; the third term is proportional to the hop count. Specifically, the expected cost [1] at server i is

$$\frac{W_1}{a+c_i} + \frac{W_2}{m_{i,j}+1} + W_3 \times d_i$$

In this formula, i is the server number, j is the videoID. $m_{i,j}$ means the MFQ value of video j at server i, c_i is the number of free channels at server i, a is the adjustment parameter which is the load balancing threshold controller, and d_i is the hop count from the client to server i. The W_k's are the weight associated with the various terms. These weights are assigned so that $W_1 >> W_2 > W_3$.

How this expected cost value determines the server selection decision can be explained under two cases: some servers have free channel, no server has free channels available. For the first case where some servers have free channels, under above weight allocation, these servers have lower cost value than those servers with no free channels. Furthermore, when the number of free channels at each server is small, the difference of c_i at server i dominates the whole cost value. Secondly, when all the servers are busy, the first term is same for all servers. Under such a situation, we assign $W_2 > W_3$, which means a high MFQ value is preferred over a low hop count, since user access latency has higher priority than bandwidth consumption.

3.2.3 Considering the channel allocation schemes

- Our last algorithm, *Merging-Aware Minimum Expected Cost (MAMEC)* is appropriate when the servers are using patching, either of the basic style or HMSM. Specifically, it adds one additional term to the server cost described above. This term reflects the preference for a server that can start a patching channel soon. To evaluate whether a patching channel can be used, one must examine the difference in the expected time for this batch to be served, as compared to the most recently scheduled regular channel. If the expected difference is too large, the request will not be served with a

[1] One can envision other possible composite cost functions. This particular measure gives good performance in simulations.

patching channel. If the expected difference is not too large, then servers with smaller expected difference are preferred since they will be able to merge sooner. We define a function $e(i, j)$ to represent this term.

$$e(i,j) = \begin{cases} r_{i,j} - s_{i,l} & r_{i,j} - s_{i,l} + avg_j < B \\ \frac{N}{2} & r_{i,j} - s_{i,l} + avg_j > B \\ N & r_{i,j} - s_{i,l} > B \end{cases}$$

Here, $r_{i,j}$ denotes the time when the client requests video j at server i, $s_{i,l}$ means the starting time of the latest regular channel l which is broadcasting video j, avg_j is the average latency when requesting video j, B is the client side buffer size in terms of video playback time, and N is some large number.

We then use the following formula to describe the merging aware expected cost measure. We allocate the weights such that $W_1 >> W_4 >> W_2 > W_3$.

$$\frac{W_1}{a+c_i} + \frac{W_2}{m_{i,j}+1} + W_3 \times d_i + W_4 \times e(i,j)$$

The client selects the server whose merging aware expected cost value is smallest. The MAMEC algorithm works similar to the MEC algorithm described above, but with a new term that indicates the channel merging probability. Note that the new term is given weight W_4 much larger than W_2 and much smaller than W_1 above, so that this term will not influence the client's preference for a server with free channels. When all the servers are busy, the client will select the server whose batch is most likely be merged into other channels.

3.2.4 Dual server reception

We also explore the option of *dual server reception* in which a client is allowed to receive video stream from channels of different VoD servers. Dual server reception only makes sense under video patching and HMSM schemes. Dual server reception allows a much wider channel patching range, and can achieve more efficient channel merging.

On the other hand, batch scheduling and channel allocation as well as server selection under this scenario is more complex. Whenever the server is ready to schedule a batch on a free channel, it has to contact other servers to decide whether this channel should be a patching channel or a regular channel. If it decides the newly allocated channel is a patching channel, it has to inform other servers to start sending videos to the clients in this channel.

Server selection now actually involves two separate steps. For the first step, the client selects the server using the algorithms described above. In the second step, there might be more than one server whose channels are within the patchable range, there is another server selection process required to decide which server the channel should be merged into. Currently, we only explore an algorithm which aims to reduce the client side buffer size, which means we always patch to the server whose patchable channel's starting time is closest to this newly allocated channel.

3.3 Simulation setup

In this section, we set up the simulation environment and describe the performance metrics for the proposed server selection algorithms.

3.3.1 Simulation Environment

We use the GT-ITM transit-stub model to generate a network topology [8] composed of about 1400 routers. The average degree of the graph is 2.5, and core routers have a much higher degree than edge routers. VoD servers and clients are associated with the edge routers.

In our current simulation, all the VoD servers have the same configuration. Each video server has $M = 100$ video programs, and has a capacity of transmitting $C = 1000$ video streams simultaneously. Other than the constraint on the video server, we assume infinite link bandwidth. We further assume that each video program has a play time of 100 minutes. MFQ is used in all the servers, and an alternative technique is used to calculate the factored queue length. We use $\sqrt{q_j \times \Delta t_j}$ as the factored queue length, where for video j, Δt_j is the time interval since the last time this video is scheduled.

Each client's request can be represented by a three tuple $(requesttime, clientID, videoID)$. The client request arrival conforms to a Poisson process with an average rate of λ ($20 \sim 110$ per minute). The client which sends out request is randomly distributed all over the network. The video selection probability conforms to a Zipf distribution. For λ requests per minute, $0.7 \times \lambda$ requests are for the 7 hot movies; $0.225 \times \lambda$ requests are for 18 normal videos; $0.075 \times \lambda$ requests are for the 75 unpopular movies. In our experiment, the total simulation time is one day.

3.3.2 Performance Metrics

We are interested in three performance metrics: user perceived latency, network bandwidth consumption and channel merge rate. We describe these three performance metrics in detail.

User perceived latency L is measured by the total amount of latency experienced by the clients over the total number of client requests. We define R to be the set of all client requests; L_r represents the access latency for request r.

$$L = \frac{\sum_{\forall r \in R} L_r}{|R|}$$

Network bandwidth consumption B is measured by the total amount of bandwidth used by all the multicast channels throughout the simulation time. Each video channel serves client requests through a multicast tree t; all the multicast trees used forms set T. $|t|$ denotes the number of edges in tree t. Thus, the network bandwidth consumption can be represented as:

$$B = \sum_{\forall t \in T} |t|$$

We also define channel merge rate as the number of clients that are merged over the total number of clients.

$$r = \frac{\sum_{\forall p \in P} |p|}{\sum_{\forall a \in A} |a|}$$

In this formula, P denotes the set of patching channels, and A denotes the set of all the scheduled channels. $|p|$ and $|a|$ means the number of clients in the channel. Channel merge rate represents the effectiveness of batching in the system.

3.4 Performance Evaluation

We conduct simulation experiments on three channel allocation schemes: persistent channel allocation, video patching, and HMSM. First we compare the user-perceived latency under different server selection algorithms. We then study the influence of the number of servers on the overall performance. We also The overhead of server selection schemes are also evaluated. Finally, we consider the effect of dual server reception on the overall performance.

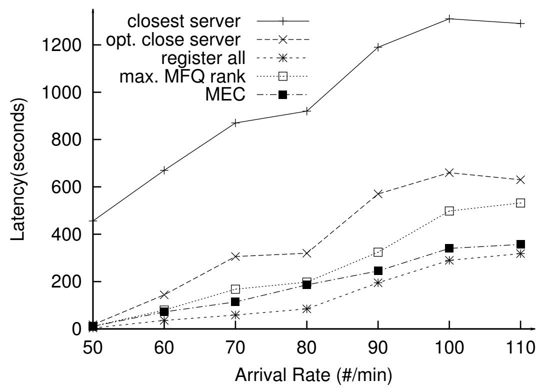
3.4.1 User Perceived Latency

In the first experiment, we evaluate the user perceived latency under persistent allocation, video patching, and HMSM channel allocation schemes. Note that the video request distribution and client viewing behavior are the same throughout the simulations. Figure 2 shows the experimental results.

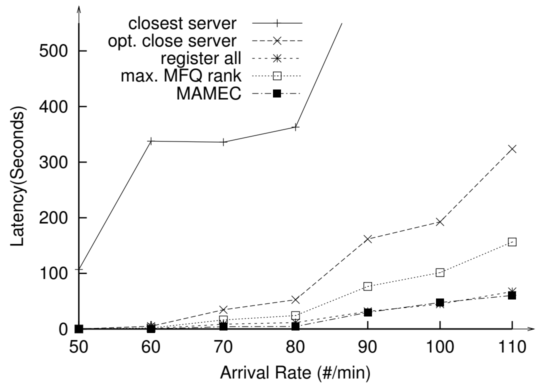
In this figure, the x axis represents the number of requests for a certain video every minute, the y axis represents the access latency. *Closest server* selection approach performs worst under all three schemes, since it does not consider any batch scheduling information. Optimized closest server selection significantly reduces user perceived latency, simply by considering the number of free channels at each video server first. The *Register all* approach has very short access latency, the reason is that each client always select the server which schedules the client request earliest. The MEC algorithm latency is very close to the Register all approach, which shows our expected cost computation formula is fairly reasonable.

Under video patching and HMSM schemes, the client access latency is much smaller than the persistent channel allocation scheme, since these two schemes make better use of multicast channels by intelligently merging the client batches during the video transmission process. We also note that MAMEC algorithm performs best among all server selection approaches.

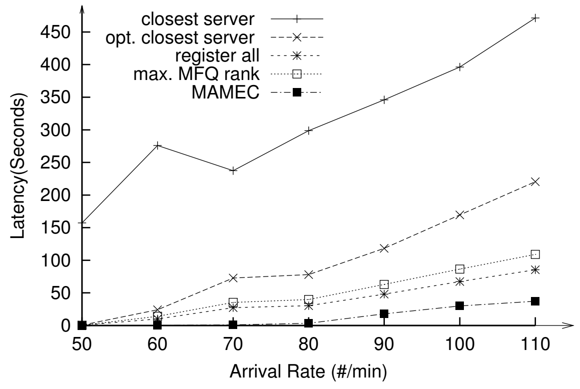
Given the same server capability, the user perceived latency is primarily determined by two factors: the server load and batching efficiency. Server load is evaluated the number of video requests served by a certain VoD server, while the batching efficiency is evaluated by the average size of a batch. The MEC and MAMEC algorithms not only distribute the



(a)Persistent Allocation



(b) Video patching



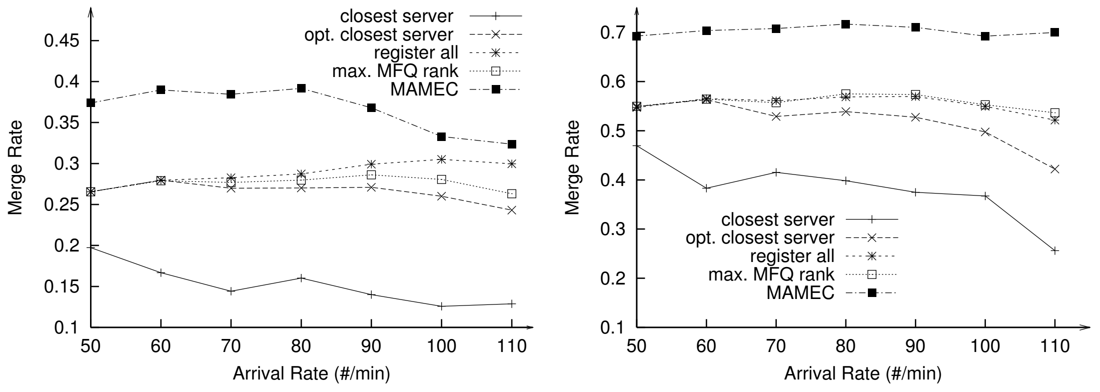
(c) HMSM

Figure 2: User Perceived Latency

client requests evenly to different servers, but also try to direct client to a server whose corresponding batch size is the largest.

3.4.2 Channel Merge Rate

As shown in the previous experiment, video patching scheme and HMSM scheme have much better user perceived latency compared to persistent channel allocation. The main reason is that these two schemes improve the channel efficiency by merging clients of different batches into one video channel which serves a larger batch of clients. In this experiment, we evaluate the channel merge rate of different server selection algorithms, and analyze the correlation between channel merge rate and user perceived latency.



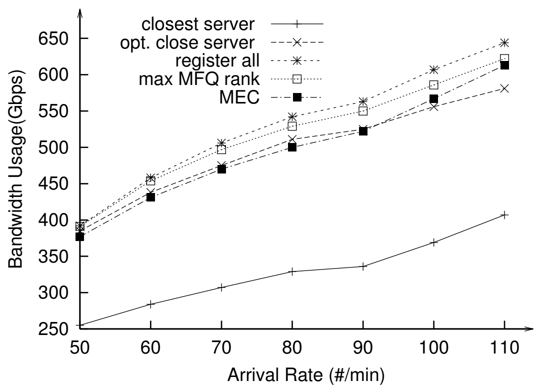
(a) Video patching scheme (b)HMSM scheme

Figure 3: Channel Merge Rate Comparison

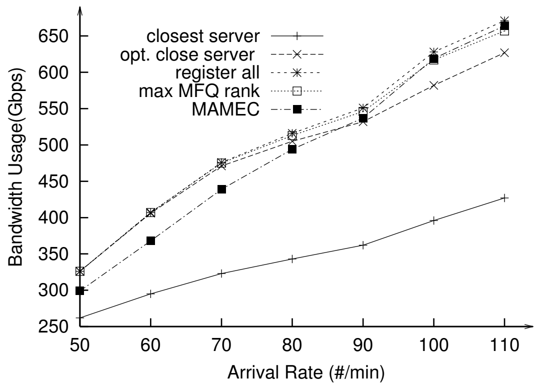
Figure 3 shows the channel merge rate under different server selection approaches. Under video patching and HMSM schemes, the user perceived latency is tightly correlated with channel merge rate. Higher channel merge rate will release more video channels at earlier stage, and thus start scheduling new request sooner. The MAMEC algorithm achieves the best user-perceived latency by efficiently merge the client requests.

3.4.3 Network Bandwidth Consumption

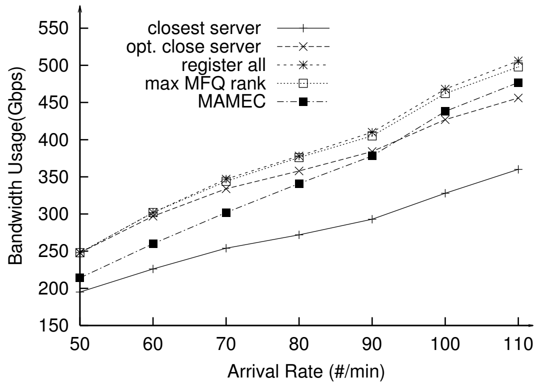
Figure 4 shows the network bandwidth consumption under different schemes. The Closest server selection algorithm consumes least bandwidth, since the client always selects the



(a)Persistent Allocation



(b) Video patching



(c) HMSM

Figure 4: Network bandwidth consumption

nearest server, and these clients tend to share more common links. This algorithm also sacrifices access latency, thus creating larger batches, causing more shared links among the same channel. Other algorithms tend to consume more network bandwidth resources. These algorithms put higher priority on server batching state information than the client-server distance. The clients selecting the same server does not show high geographically clustering property. Since the client requests are more evenly distributed among all the VoD servers, this reduces the chance to form larger video batches on a single server, this also makes the number of shared links in the multicast tree decrease. Our *expected cost* approach achieves fairly good results in both bandwidth consumption and user access latency.

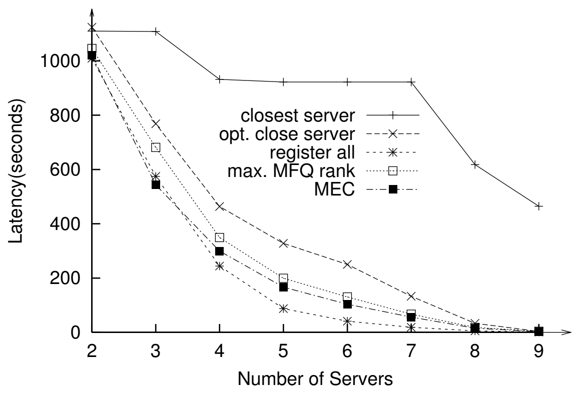
From figure 4, we observe that the network bandwidth consumption increases as the client request rate gets higher. But network bandwidth consumption does not increase in the same speed of client request rate. Two reasons contribute to such property. First, as the client request rate increases, the user access delay also increases, in consequence, the average batch size for each video channel gets larger. Thus, a multicast tree tend to have more shared links than the smaller multicast trees. Second, as the client request rate increases, clients close to each other have higher probability to be served by the same server, this further increase the degree of link sharing within the multicast tree.

We also notice that video patching and HMSM schemes consume less bandwidth resource than persistent channel allocation scheme. The reason is under video patching and HMSM, multicast groups can be merged during the video transmission process, creating larger multicast tree with more shared links.

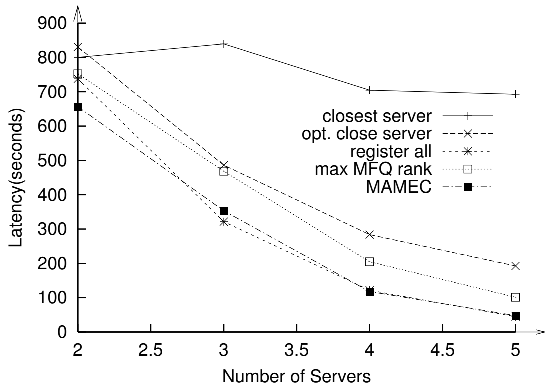
3.4.4 Influence of Number of Servers

We study the performance as the number of servers varies. We are interested in both the user perceived latency and network bandwidth consumption.

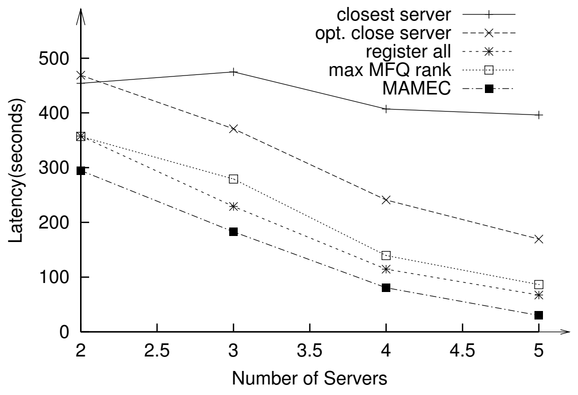
- **User perceive latency** Figure 5 shows the influence of the number of servers on the user perceived latency. In these experiments, the client request rate is 100 requests per minute. As the number of servers increases, the latency decreases rapidly. After the number of servers is above a certain value, adding more servers does not improve the performance



(a)Persistent Allocation



(b) Video patching



(c) HMSM

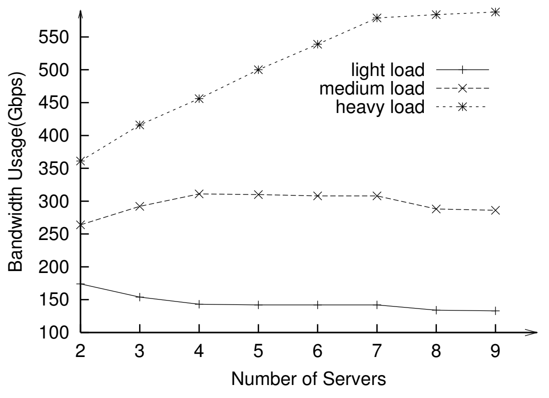
Figure 5: User Access Latency Under Different Number of Servers

significantly. The performance of the Closest server selection algorithm is not as sensitive to the number of servers as other server selection algorithms. The effect of increasing the number of servers on the access latency is not surprising, because adding servers increases service capacity. It is, nevertheless, important to confirm that the service replication and selection maintains the property that performance improves as more capacity is added. It is also worthwhile in determining when the performance reaches a level of diminishing return.

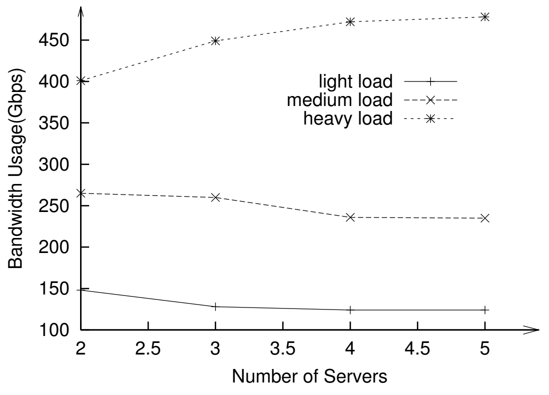
- **Network bandwidth consumption** To evaluate the influence of the number of servers on the bandwidth resource usage, we study the bandwidth usage behavior under three traffic loads: light (10 requests/min), medium (40 requests/min), and heavy (80 requests/min). Figure 6 shows the network bandwidth consumption when we use different number of servers. We show the result of MEC/MAMEC algorithms only. For the persistent allocation scheme, when the traffic load is light, as the number of servers increase, the total resource decreases. When the traffic load is medium, the network bandwidth consumption first increases, and then decreases. This is because as the number of servers increase, while the client can find a closer server, the load is distributed among the servers reducing the multicast efficiency; the influence of this efficiency becomes more apparent when the traffic load is heavy, the network bandwidth consumption keep increasing with the number of servers. For patching and HMSM schemes, efficient channel merging increases the advantage of multicast transmission, even under heavy traffic load. Hence the network resource consumption does not increase too much.

3.4.5 Overhead of the server selection algorithms

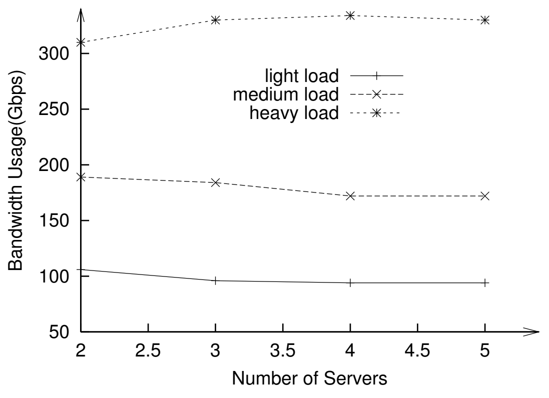
To support efficient server selection, each client has to first obtain the server state information before sending out a request to a certain VoD server. In this section, we evaluate the overhead of the server selection schemes. There are two possible approaches to obtain the server state information: a centralized approach, and a distributed approach. In a centralized approach, there is a front end server which is responsible to make server selection decisions. Upon receiving a client's request, it collects server state information and returns



(a)Persistent Allocation



(b) Video patching



(c) HMSM

Figure 6: Network Bandwidth Consumption Under Different Number of Servers

the IP address of a best server to this client. In a distributed approach, each client send request to all the VoD servers to collect state information, and make server selection decision on its own.

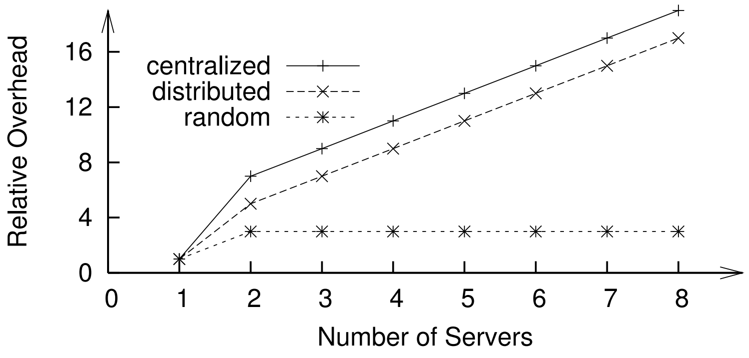


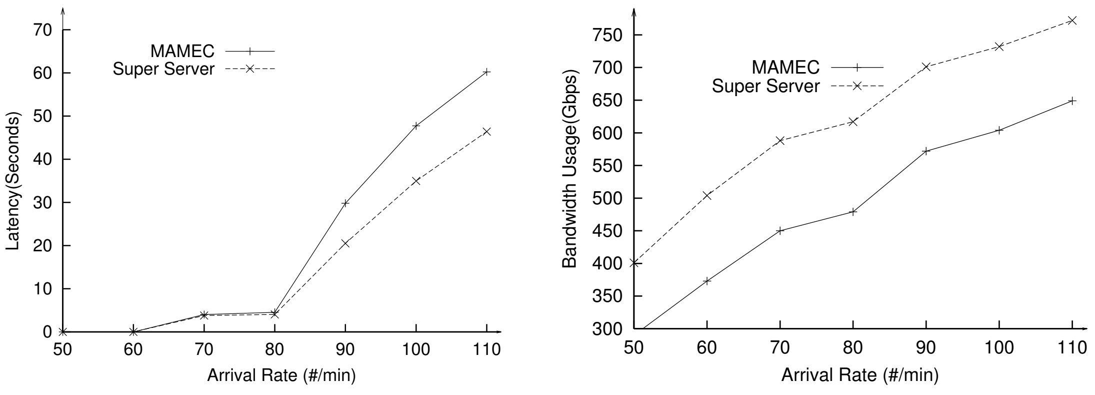
Figure 7: Server selection overhead

Figure 7 shows the simulation results when increasing number of servers are used in the replicated VoD system. In this Figure, "random" refers to the approach where a client send a request to a front end server, and this server returns a random VoD server IP address to this client. Also, we assume that the weight of each signal message (i.e. request message, server state message, reply message) is same. As the number of VoD server increases, the overhead of server selection also increases. We also find out that the overhead increases linearly as the number of server increases. We will discuss more about the overhead factor in the later section.

3.4.6 Comparison with a "Super" VoD Server

It should be clear that one can increase capacity without replication by simply adding power to a single server. In this section, we assume there is a "super" VoD server, which has the capacity of the sum of all the replicated servers. We run MEC/MAMEC algorithms on the replicated server environment, and compare the result with the super server environment. A super server provides a lower bound of on server selection algorithms in terms of access latency.

Figure 8 shows the performance of one super server versus replicated servers under the video patching scheme. Persistent allocation and HMSM have similar results.The super



(a) Access Latency (b)Bandwidth Consumption

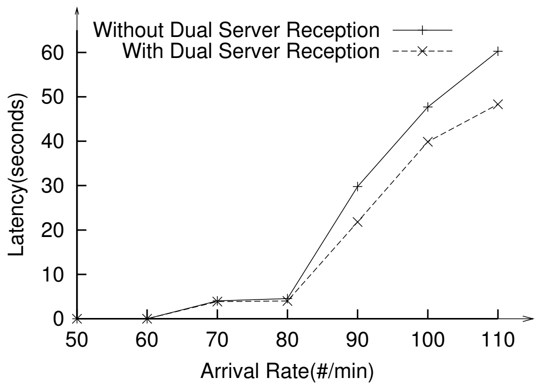
Figure 8: Super Server Vs. Replicated Server

server provides modestly better access latency, When this server schedules a new request, it has more on-going regular channels broadcasting the same video program, thus the newly scheduled channel has a better chance to merge into other channels. On the other hand, the super server consumes significantly more bandwidth than the replicated server environment. Since all the clients receive the video streams from the same server, the average distance from the client to the server is larger than the replicated server case.

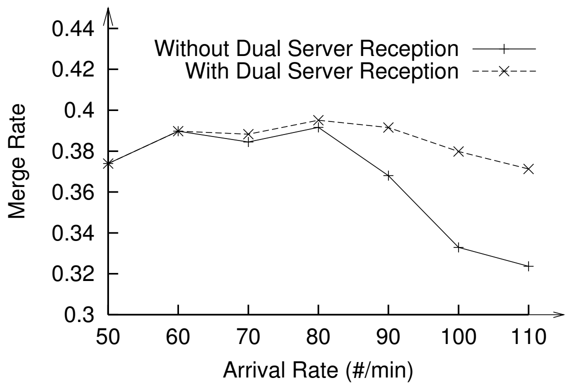
Based on the simulation result, we argue that increasing VoD service capacity through replication is preferable for the following reasons: First, replication allows shorter paths, thus improving overall bandwidth consumption. Second, the service capacity of a single server cannot be increased without bound as one ultimately would reach the limit of the server's access link bandwidth. Further, replicated VoD servers are more robust, flexible, and cost-effective.

3.4.7 Influence of Dual Server Reception

In the previous results, the client request is served by one VoD server. In some cases, however, a "patchable" channel maybe on another replicated server. In this case, if the client's request can be served by two servers, the channel merge rate can be improved, and the client access latency will be reduced. Dual server reception can only be used under video patching and HMSM schemes. In this section, we study the influence of dual server reception.



(a) Access Latency



(b) Channel Merge Rate

Figure 9: Influence of Dual Server Reception

Figure 9 compares the simulation result with/without dual server reception under the video patching scheme. The HMSM case has very similar results. Figure 9 (a) compares user access latency using the MAMEC algorithm. When the client request rate is low, there is not too much difference, since the video servers are not fully loaded yet. As the client request rate increases, allowing dual server reception can achieve a higher channel merge rate by merging into video channels of other video servers. This in turn reduces the user access latency. Dual server reception can also reduce the requirement for client buffer size by merging into the closest channel of all video servers. We will study the influence of different merging selection algorithm on overall performance in future research.

3.5 Implementation Issues

We envision a system architecture similar to the one proposed in [83], and shown in Figure 10. Replicated VoD servers and clients are distributed over the network. There is a central directory server which stores each video server's state information, including channel status and batching status. *Proxy servers* placed in appropriate locations in the network are used to direct clients to "best" servers.

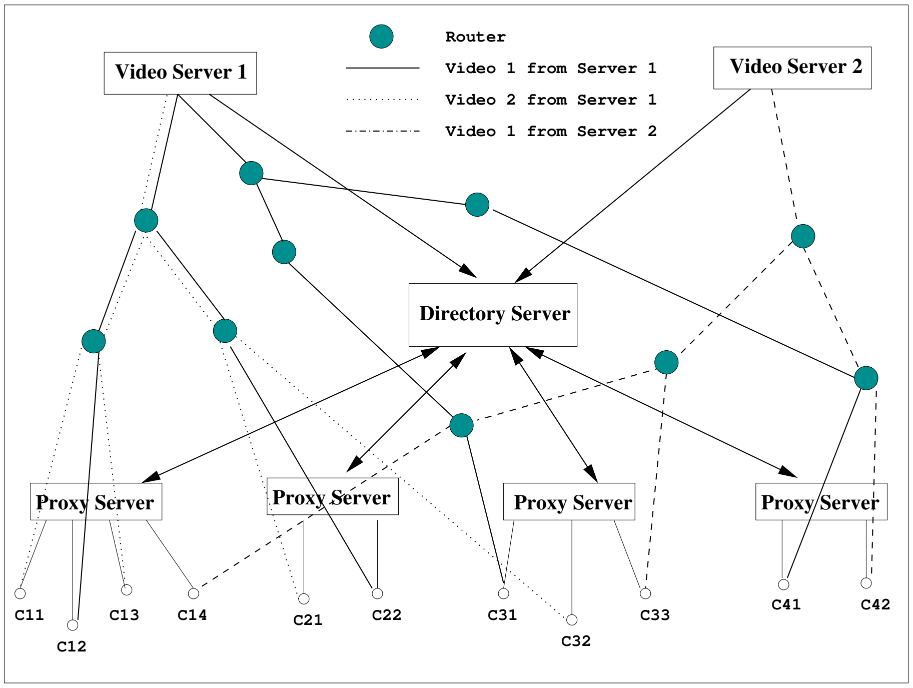


Figure 10: Replicated VoD Service Architecture

In such an architecture, clients first submit requests to the proxy servers (in a manner similar to DNS), proxy servers aggregate the client requests and forward the requests to the directory server. Directory server receives request from the proxy server, and returns server state information. The proxy server then makes a server selection decision and return the identity of the server(s) that this client should contact for service. The proxy servers are also responsible for video filtering. If two servers send the same video streams to the client, the proxy server should be able to drop one of the video stream and send a cancel signal back to that server.

The proxy servers need to maintain the information needed to make server allocation decisions. The type of information needed varies for the various selection techniques we described. In the most general algorithms the state of each server's video channel as well as the state of its client batches need to be known at the proxy servers. Generally, there are two approaches to achieve this purpose: VoD server push and directory server pull. For server push, whenever the server state information is changed, it will push the new value to the directory server. This approach can guarantee that the information maintained at the directory server is always up to date when it is requested from a client. This approach may create useless traffic if the state information is updated more frequently than needed. For the pull approach, if the server state is changed, an invalidation message will be sent to the directory server. When the client request comes, the directory server will pull the state information from video servers if the server state is invalidated. This approach may cause extra delay since the client has to wait for the directory server to fetch the updated state information from the video server. A combined push/pull approach is a reasonable solution for server state information update.

In practice, it may not be feasible to keep complete and up-to-date information for all the servers at all the proxy servers. One way to reduce the overhead of maintaining this information at the proxy servers is to not insist on full accuracy and consistency. In our future research, we plan to consider the effect of using slightly out-of-date and possibly inaccurate information on the performance of the selection techniques. An important observation from our simulations is that while selection algorithms that take into account all information about servers perform best, other algorithms which use only limited information provide relatively good performance as well. This seems to indicate that selecting according to slightly out-of-date information may also perform reasonably well.

3.6 Summary

In this paper, we consider the design of selection techniques for a set of replicated batching VoD servers. Our work explores the use of a range of selection techniques that use a varying amount of knowledge about the servers and their location. We show that, with the exception of the very naive Closest Server selection technique, server replication can indeed be used as a way to increase the capacity of the service leading to improved performance. As discussed in Section 4, using replication is a better approach to increase service capacity when compared to simply providing a more powerful single server. We have demonstrated here that one can indeed design server selection algorithms to make full use of the increased capacity provided by replication.

In our future research we plan to consider several issues relating to improve the efficiency and performance of replication of batching VoD servers, include:

- Evaluation of the effect of out of date information on the performance of the selection techniques. We will design a protocol which exchange server state information as described above, and observe the overall performance.
- The use of partially replicated VoD servers where not all the videos are available at all the servers. The video request pattern is likely to follow the Zipf distribution law, where most of the video requests are focused on a small subset of hot videos. For unpopular videos, replication may cause performance degradation, with longer user perceived latency due to low MFQ values, and more bandwidth usage due to very small multicast groups. It is more reasonable to replicate video programs according to how popular they are.
- The use of heterogeneous replicated servers in which different servers may use different batching techniques. In the real world, it is highly possible that different VoD servers adopt different scheduling approaches. How to fit our server selection algorithms to such an environment is important for implementation considerations.

CHAPTER IV

SCALABLE LIVE VIDEO STREAMING TO COOPERATIVE CLIENTS USING TIME SHIFTING AND VIDEO PATCHING

We consider the problem of how to enable live streaming of video content from a single server to a large number of clients using application layer multicast. In application layer multicast, video clients form an overlay network over which the video is propagated. In this approach, a client currently in the overlay network forwards the content it is receiving, and serves other client's request as a server. By distributing the transmission load to the clients all over the network, the video server is no longer the bottleneck. This approach is scalable in the sense that the forwarding capability of the overlay network is growing incrementally. New clients joining the network also bring in extra bandwidth capacity to the system. The major problem for application layer multicast is the video discontinuity caused by the dynamics of membership. Since the clients in the group can leave at any time, other clients which are receiving video content from them have to suffer service disconnection. Although the disconnected clients can resume the service by rejoining the application layer multicast tree, this process can take time and can result in loss of video content and interruption of video reception.

In this thesis, we develop a scheme using the transmission of a *single description coded* video over an application layer multicast tree formed by cooperative clients. Clients in the tree always receive full quality video stream. Video continuity is maintained in spite of tree disruption caused by departing clients using a combination of two techniques: (1) providing time-shifted streams at the server and allowing clients that suffer service disconnection to join a video channel of the time-shifted stream, and (2) using video patching to allow a client to catch up with the progress of the live video program.

4.1 Design Overview

In this section, we first give a general description of how our system operates. Then, we discuss the potential problem of the straightforward time shifting solution. We then apply video patching in live streaming service, and describe how the design goals are met. Finally, we give an example to illustrate the system operations.

4.1.1 Basic Operations

Our design of the system is composed of three components: 1) a time shifting video server, 2) a level-based tree management protocol, and 3) a video stream patching scheme.

A time shifting video server S broadcasts video program in C channels. Each channel can be used to transmit one video stream. There is an application layer multicast tree associated with each channel. The server serves the clients with the original stream, and m time-shifted streams. We label these streams $s_0, s_1, \cdots, s_m$. Stream s_0 is the original stream, while s_i starts after a $i \times d$ delay. Video server is the single source of the video content, and is the root of the application layer multicast tree. It processes client requests to join, leave, and rejoin the multicast group, and is responsible for maintaining the topological structure and resource availability of the multicast tree.

When a client first joins the multicast group, it always joins a multicast tree of the original stream. If the server has free video channel available, the client connects to the server directly. Otherwise, the client joins the tree by connecting to a client already in the tree who has enough available bandwidth resources, while at the same time, has the shortest overlay path to the video server. This node join protocol guarantees that the clients in the upper level of the tree are fully loaded, before the clients in the lower level of the tree start to adopt new clients as their children. In this way, we can get a "well-shaped" wide and short multicast tree. A wide and short tree can achieve lower bandwidth consumption, and can reduce the probability of service disconnection due to ancestor node failure.

A client in the multicast tree suffers service disconnection in two cases: 1) upstream link congestion, or 2) an ancestor node's failure. Similar to CoopNet, to detect service disconnection, the client sets a threshold value, if the packet loss rate is above the threshold, the client deems it as a service disconnection. For the case of ancestor node failure, the client detects 100% packet loss. The client manages to rejoin the group by connecting itself to another parent node. The *node rejoin delay* for a client is the time interval between the moment when the client is disconnected and the moment when the client is reconnected. We denote the *node rejoin delay* for client c as r_c. A straightforward approach for lossless video reception is: when the client rejoins the tree, it can select to join the video channel of an appropriate time-shifted video stream, so that it will not miss any video content. For example, if at time t_0, the client is disconnected, and it manages to rejoin the group at $t_0 + r_c$, the appropriate stream should be the stream with $\lceil \frac{r_c}{d} \rceil$ delay. Clients that join the same video channel form an application layer multicast tree.

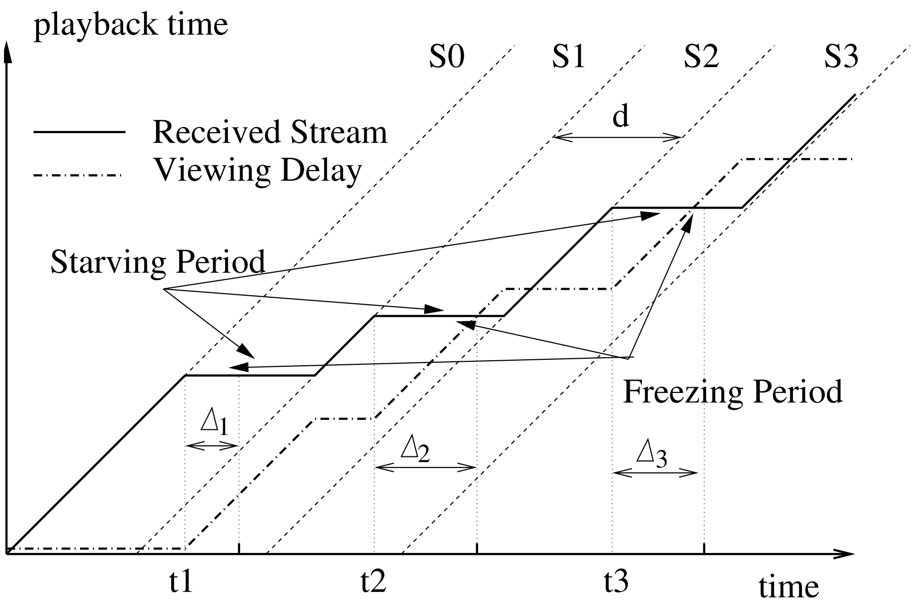


Figure 11: An example of indefinite shifting

A client might experience multiple service disconnections during the video reception process. Figure 11 shows an example when client c suffers multiple disconnections. In this figure, the horizontal axis is the actual time, while the vertical axis is the video playback time. The server sends out video streams with equal time shifting interval d. The client experiences three service disconnections at time t_1, t_2, and t_3. *Viewing delay* means the delay between the playback time of the video stream that the client is watching and the playback time of the original stream. *Starving period* is the time interval when the client is not receiving any video. *Freezing period* refers to the time period when the client side play out is temporarily stopped.

The client joins the original stream at time 0. During $[0, t_1]$, the viewing delay is 0. At time t_1, it is disconnected from the application layer multicast tree. It manages to reconnect to the tree at time $t_1 + \Delta_1$, the missed video is $[t_1, t_1 + \Delta_1]$. The designated time-shifted stream is s_i, where $i = \lceil \frac{\Delta 1}{d} \rceil = 1$ (assume $\Delta_1 < d$). The client begins to receive from stream s_1 at time $t_1 + d$. The viewing delay during $[t_1, t_1 + d]$ increases from 0 to d. The client is disconnected again at time t_2, and rejoins the tree at $t_2 + \Delta_2$. Note that at this time, the missed video portion is $[t_2 - \lceil \frac{\Delta 1}{d} \rceil \times d, t_2 + \Delta_2]$. The designated time-shifted stream should be s_j, where $j = \lceil \frac{\Delta 1}{d} \rceil + \lceil \frac{\Delta 2}{d} \rceil = 2$. (again, we assume $\Delta_2 < d$). The client begins to receive from stream s_2 at $t_2 + d$, and the viewing delay is increased to $2 \times d$. Similar event occurs after the third disconnection. In a general case, if a client suffers n service disconnections, it has to switch to stream s_k, where $k = \sum_{i=1}^{n} \lceil \frac{\Delta_i}{d} \rceil$, and the client viewing delay will be $k \times d$. Here, Δ_i denotes the rejoin delay after the i_{th} disconnection.

As shown in the Figure 11, each time the client rejoins the multicast tree, it has to join a stream with a larger time shifting value. This can result in *indefinite time shifting*, which is undesirable since the client's reception time could be much longer than the actual video program time. Also, if the time shifting value exceeds the boundary that the server can provide $(m \times d)$, the client will suffer video content loss. Another problem demonstrated in this example is *video freezing* caused by *video starvation*. In this example, the *starving period* and *freezing period* is same. Whenever the client is disconnected from the application layer multicast tree, the client side's playback is paused. In our design, we introduce *video patching* to tackle the *indefinite time shifting*; and use *initial buffering* to provide continuous video streaming.

4.1.2 Video Patching in Live Streaming

Video patching is a popular channel allocation technique in Video on Demand (VoD) service. It is designed to provide better server channel utilization and thus lower server load. In video patching, video channels of the VoD server are classified in two categories: regular channels and patching channels. Regular channels transmit the entire video stream, while patching channels only transmit the initial portion of a video as needed. When the server allocates a free channel to client requests, it first scans the on-going channels. If there is no regular channel distributing the requested video, or the starting time for this video is too early for the client to patch, this channel is allocated as a regular channel. Otherwise, the channel is allocated as a patching channel. Under video patching, the clients receive data from both the regular channel and the patching channel. The data from the patching channel is used to make up for the data the clients are missing from the regular channel. While receiving from the patching channel, the data being received from the regular channel is buffered by the clients for playout when needed. When the missing portion is completely received, the patching channel is released, and the clients only receive packets from the regular channel. More detailed description of video patching can be seen in [45].

In our scheme, the server sends out the original stream, as well as multiple time-shifted streams spaced by a fixed time interval d. The time-shifted stream serves as patching stream in our system. Here is how it works: when the connection to the tree is re-established after a service disconnection, a client would have missed the video from the point it is disconnected to the point it is reconnected. The reconnected client utilizes a time-shifted stream as patching stream. The patching stream is used to retrieve the missed video portion. At the same time, this client also receives the original stream, this stream is used to catch up the progress of the video program. During the patching period[1], the client is actually receiving at twice the speed as the normal stream rate. After the progress of the video is caught up, the patching stream is released, and the client receives only from the original stream.

There are several major differences between video patching in VoD service, and video patching in our scheme. 1) Different purposes: video patching in VoD service is used to reduce the access latency, while video patching in our scheme is used to provide lossless video reception. 2) Different starting time: video patching in VoD service starts at the

[1]patching period denotes the time when a client is receiving both the original stream and the time-shifted stream

beginning of the service, in our scheme, video patching could happen at anytime during the video reception process. 3) Different releasing time: in VoD service, the patching channel is released when the video playback difference with another regular channel is fixed, while in our service, the patching channel is released when the missed video is retrieved.

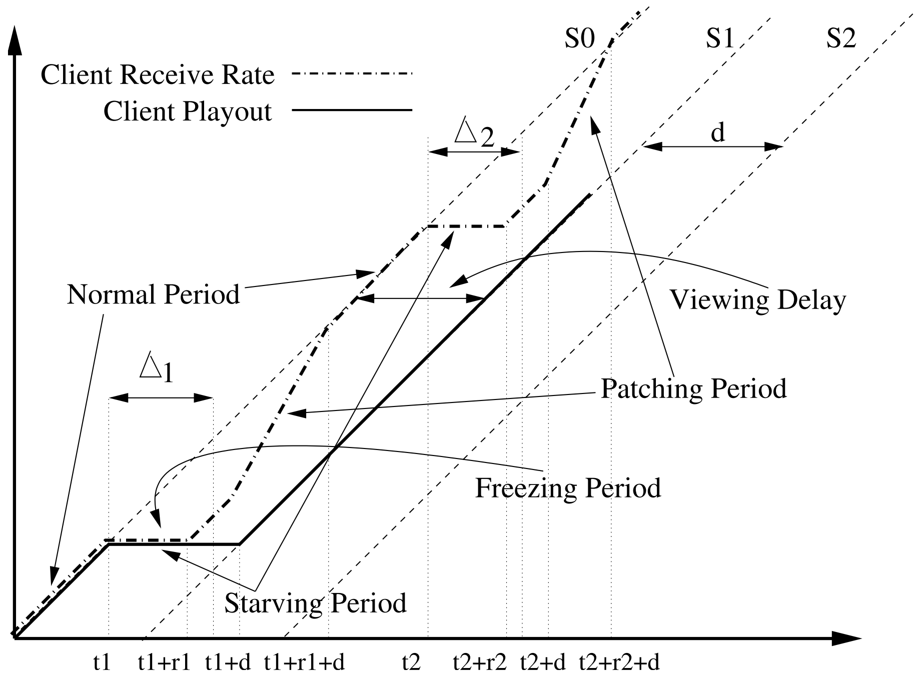


Figure 12: Video patching in live streaming

We illustrate how the video patching scheme works, and how it eliminates *indefinite time shifting* through an example shown in Figure 12. Assume at time t_1, the client is disconnected. It rejoins the group by sending out two "rejoin" signals to the server. One of them is for the original stream, the other is for the patching stream. The client rejoins the original stream at time $t_1 + r_1$, the missed video portion is $[t_1, t_1 + r_1]$. It rejoins the patching stream at time $t_1 + \Delta_1$ (here, we assume $\Delta_1 < d$), and begins to receive the patching stream s_1 at time $t_1 + d$. At time $t_1 + r_1 + d$, the missed video portion is made up, and the patching channel is released. From this time, the client receives only from the original stream. Assume at time t_2, the client is disconnected again. At this moment, the client's playing out time is $t_1 - d$, and its buffer stores the video portion of $[t_1 - d, t_1]$. The node rejoins the original stream at time $t_2 + r_2$, and the missed video is $[t_2, t_2 + r_2]$. It rejoins the patching stream at time $t_2 + \Delta_2$. We analyze two cases based on the value of Δ_2. In the first case where $\Delta_2 \leq d$, the client starts to receive patching stream at $t_2 + d$. During $[t_2, t_2+d]$, the client is playing out from the buffer, the client's viewing delay remains unchanged. When the client rejoins the tree, it still receives the patching stream from s_1, instead of the other streams with longer time shifting values. In the second case, where $\Delta_2 > d$, say $(i-1) \times d < \Delta_2 < i \times d$, $i > 1$. The client starts to receive patching stream at $t_2 + i \times d$. During $[t_2, t_2+d]$, the client is playing out from the buffer, the buffer is exhausted at $t_2 + d$. When the client rejoins the tree at $t_2 + i \times d$, it receives the patching stream from s_i, and the client viewing delay is $i \times d$. In a general case, if a client suffers the n_{th} disconnection, the client viewing delay should be determined by the formula below:

$$d_n = \begin{cases} \lceil \frac{\Delta_n}{d} \rceil \times d & n = 1 \\ d_{n-1} + \lceil \frac{max(\Delta_n - d_{n-1}, 0)}{d} \rceil \times d & n > 1 \end{cases}$$

Solving this formula, we get:

$$d_n = max(\lceil \frac{\Delta_i}{d} \rceil) \times d \quad i \in [1..n]$$

The client side viewing delay is determined by the maximum value of the *node rejoin delay*. Thus, the *indefinite time shifting* can be eliminated.

4.1.3 Continuous Video Streaming

As we stated in previous sections, the combination of a time shifting server and a video patching scheme can achieve lossless streaming, and prevent *indefinite time shifting*. But there are still some periods that the client's playback is halted, when the client is temporarily disconnected from the multicast tree. This typically happens when the client is first disconnected, when the client buffer is empty; or when the client's rejoin time is longer than the playback time of the buffered video.

To avoid interruption of play out during the *starving period*, buffered video accumulated during an initial playout delay can be used. In our design, when the client first joins the multicast tree, instead of immediately playing out the video stream, it can buffer the initial part of the video for a certain time. This time interval is called *initial access delay*. By waiting for an appropriate *initial access delay*, when there is a service disconnection in the future, the client can still playout the video from its buffer instead of stopping the playout.

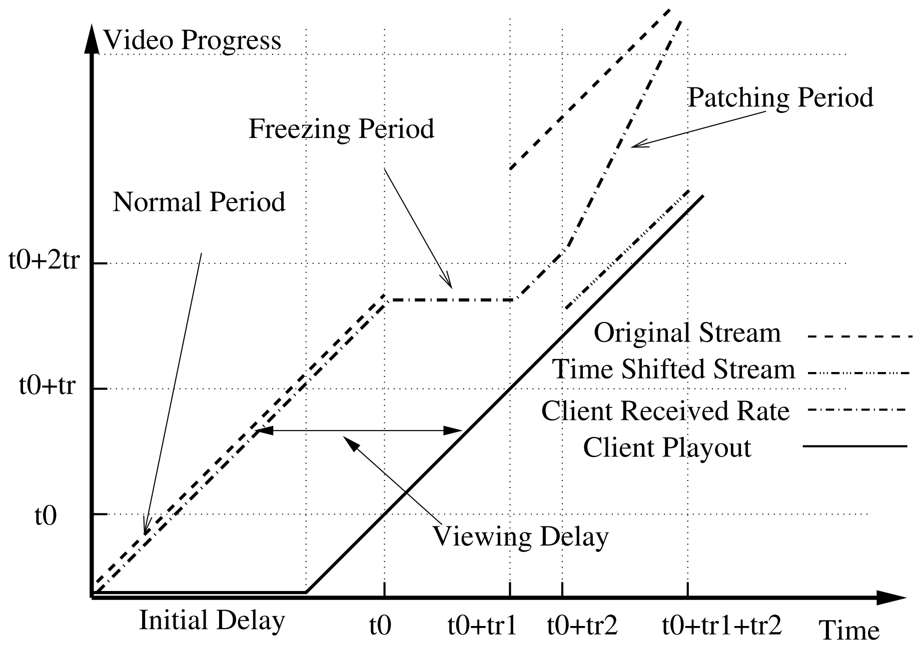
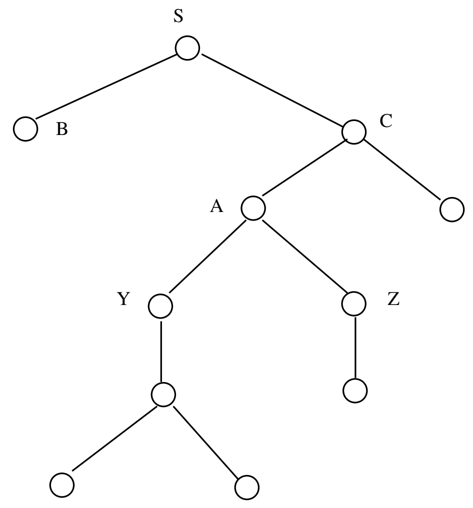


Figure 13: Continuous video streaming with initial delay

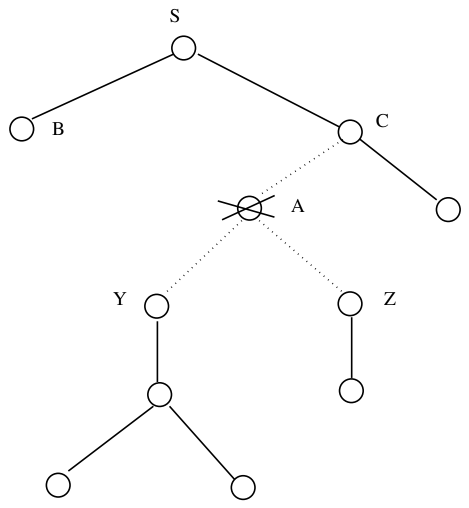
We illustrate our solution in Figure 13. The client connects to the server at time 0, instead of playing out immediately, it buffers the initial D time unit video. At time D, it begins to play out the video from its buffer, while it keeps receiving from the original stream. As to how large this D should be, there is a tradeoff between access latency and video continuity. Obviously, longer access latency can result in smoother video reception. In the case of time shifting without video patching, the initial delay should be $D \geq \sum_{i=1}^{n} \lceil \frac{\Delta_i}{d} \rceil \times d$. If video patching is introduced, the initial delay can be reduced to $D \geq max(\lceil \frac{\Delta_i}{d} \rceil) \times d$ under non overlapping failures.

In Figure 13, at time t_0, the client is disconnected. Since the client buffer stores video $[t_0 - D, t_0]$, it can still play from the buffer while reconnecting to the server. If the initial delay D is larger than the *node rejoin delay*, the client buffer will not be drained out before it is reconnected to the server. Once the client is reconnected, it receives the time-shifted (patching) stream as well as the original stream, until the missed video is made up. During this process, the client does not experience any pause of playout, neither does it suffer any extra delay. If D is smaller than the *node rejoin delay*, then the client side video playback has to be paused.

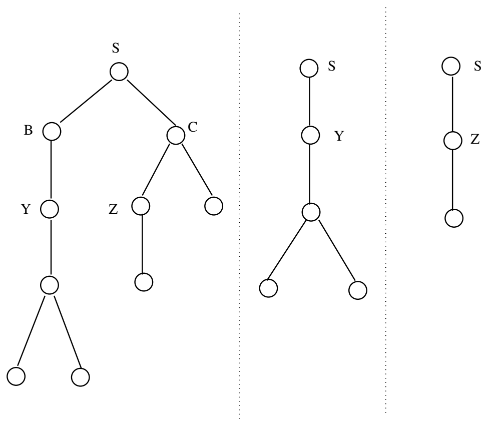
4.1.4 An Example of System Operations



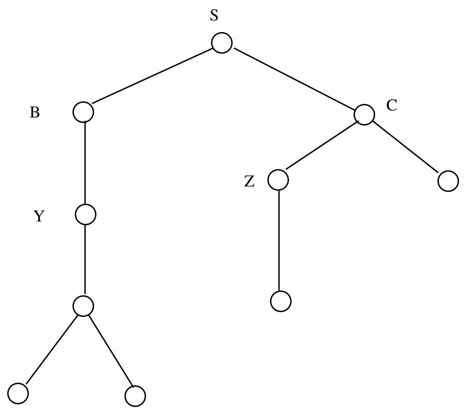
(a) the original tree



(b) node A disconnected



(c) patching structure



(d) new tree

Figure 14: The example of system operations

Figure 14 shows several snapshots of the application layer multicast tree during the video streaming process. Figure 14(a) shows the multicast tree structure where all the clients in the tree receive the original stream. At time t_0, node A leaves the group either due to congestion or node failure. This causes the subtrees rooted at nodes Y, and Z to suffer service disconnection. This is shown in Figure 14(b).

Nodes Y and Z send "rejoin" message to server S respectively. We describe the rejoin process for node Y. Node Z has similar rejoin process. The "rejoin" message includes the rejoin requests for both the original stream and the time-shifted stream. To reconnect the client to the original stream, server S selects a client which is currently in the multicast tree as the parent node of Y. In this case, client B is selected. Node Y connects with client B at time t_{Y1}, and begins to receive the original stream from it. At the same time, the server allocates a free video channel to send out the patching stream to Y. Assume the server starts to send out the patching stream at t_{Y2}, and stream s_i is the designated patching stream, where $i = \lceil \frac{t_{Y2}-t_0}{d} \rceil$. Figure 14(c) shows the multicast tree structure at time t_1, when both nodes Y and Z are in patching period. Figure 14(d) shows the tree structure when both patching channels for nodes Y and Z are released.

We analyze the influence of this service disconnection to node Y. The children of Y suffer the same experience. Node Y begins to receive the original stream at time t_{Y1}, it misses video $[t_0, t_{Y1}]$ due to service disconnection. The patching stream starts at t_{Y2}. Assume at time t_0, the client buffer has b_0 time unit of video content. Y playout the video content from its buffer, when the client is disconnected. If $b_0 \geq t_{Y2} - t_0$, then the client still plays video from the buffer when the patching stream begins to feed the client. Thus, the client's viewing process is not interrupted, and the client viewing delay remain unchanged. Otherwise, the client may suffer a pause of $t_{Y2} - t_0 - b_0$. At time $t_0 + t_{Y2} + t_{Y1}$, the patching period finishes and the patching stream is released. At this time, the client buffer size is $b_0 + max(0, t_{Y2} - t_0 - b_0)$.

4.2 Design Details

In this section, we describe our system in detail, including the time shifting video server design, and the tree management schemes.

4.2.1 Time Shifting Video Server

A media server is the single source for the live video content, and is the root of the application layer multicast tree. In this section, we describe the time shifting video server design in our system.

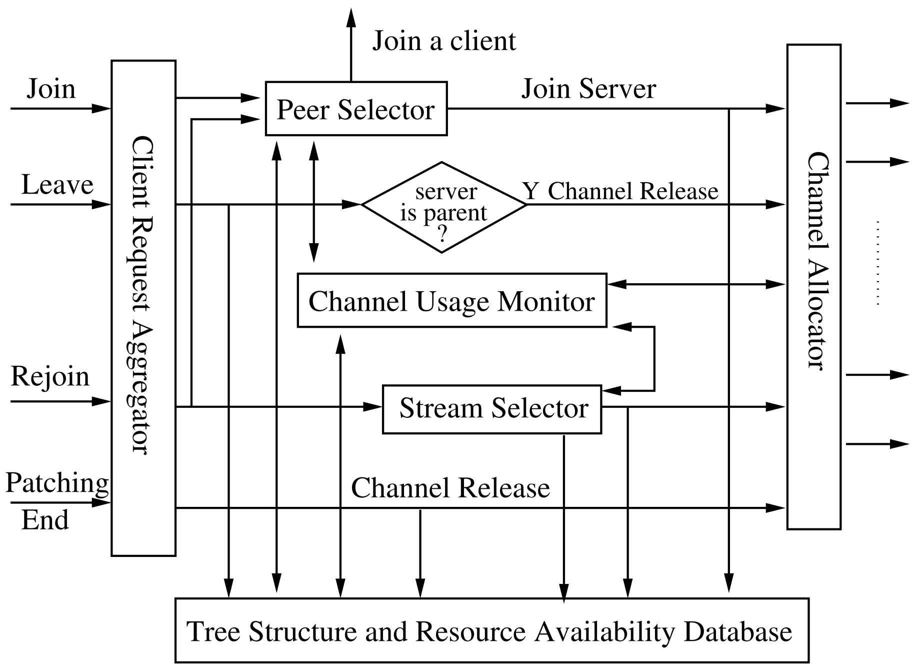


Figure 15: Video server Design

Figure 15 shows the structure of our server design. The *Client Request Aggregator* receives four kinds of client requests: *join, leave, rejoin*, and *patching end*. When the server receives a *join* request, it contacts the *Peer Selector* to find the appropriate parent node[2] to connect with. The *Peer Selector* obtains global topology and resource availability information from the centralized database. In the case of a graceful *leave*, the server signals the children of the leaving client to rejoin other parent nodes. The server also updates the *tree structure and resource availability* information upon a node leave. If a client experiences severe packet loss due to network congestion or ancestor node failure, it sends a *rejoin*

[2]Parent node can be either the video server or a client which is currently in the application layer multicast tree.

message to the server. The server then assigns another parent node to forward the original stream to it. At the same time, the server also contacts the *Channel Usage Monitor*, and assigns a free video channel to transmit the time-shifted video stream to this client. The *Stream Selector* is responsible for assigning the client a stream with appropriate time shifting value. When the missed video portion has been fully received, the client sends a *patching end* message to the server, the server then releases the corresponding patching channel.

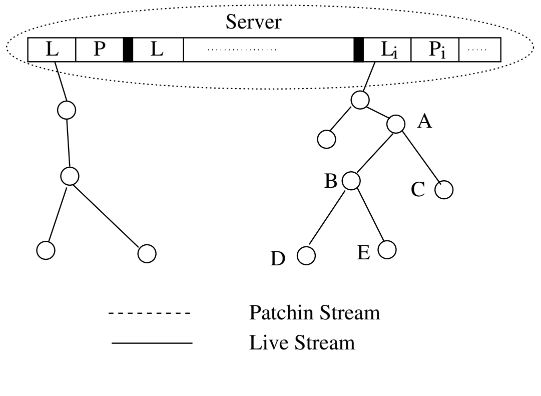
4.2.2 Channel Allocation:

Our video server streams video content in two kinds of channels. The channel for the original stream is called the *live channel*, and the channel for the time-shifted stream is called the *patching channel*. The data rates for the live and the patching channels are the same. Video content that is transmitted on a particular channel is multicast to the clients in the application layer multicast tree associated with this channel. Channel allocation deals with whether the channel is used to transmit the original video stream or the time-shifted video stream. Allocating more channels to the original stream at the server leads to a shorter and wider tree. Reserving more channels for the time-shifted streams means that when the client's rejoin message reaches the server, there is a higher probability that the server has a free patching channel available. In this way, the client can start to receive the patching stream earlier. In this section, we propose three channel allocation schemes, as described below.

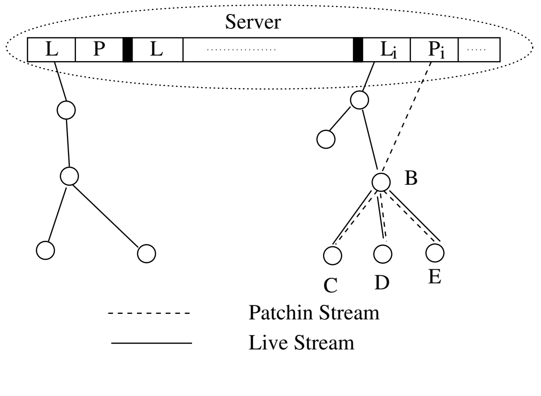
4.2.2.1 1:1 Allocation

One way to allocate video channels is that whenever a server allocates one live channel to the client, it also reserves one patching channel for this client. We call this allocation scheme, *1:1 allocation.* This approach has the benefit that whenever there is a node failure in the application layer multicast tree of a live channel, there is a free patching channel available for the clients below the failed node to patch the missed video content.

Figure 16 shows an example of the *1:1* allocation. As shown in Figure 16(a), node A in the multicast tree of channel L_i leaves the group. The subtree of node A manages



(a)Tree Structure



(b) Patching Structure

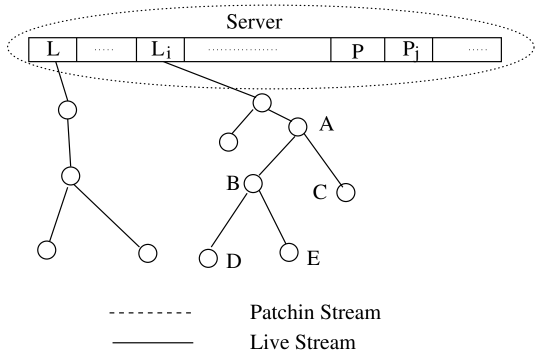
Figure 16: 1:1 channel allocation

to rejoin the patching channel P_i for the time-shifted video stream, as well as rejoin the multicast tree for the original stream. The problem with this channel allocation scheme is that each patching channel is bound with a live channel. If at time t, there are more than one patching channel requests from live channel L_i, only one of them can be served, even though the patching channels for other live channels are left unused.

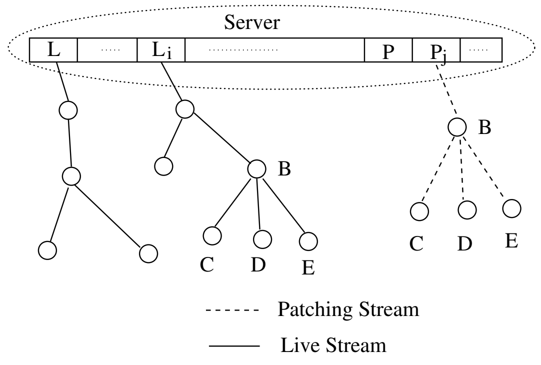
4.2.2.2 Static Multiplexing

To overcome the inefficiency of *1:1* allocation, we propose the *static multiplexing* allocation scheme. In this scheme, m of the C video channels are allocated as live channels, while the other $n = C - m$ channels are allocated as patching channels. There is no fixed binding in this scheme, the patching channel can be used to patch the disconnected clients from any live channel.

Figure 17 shows how this channel allocation scheme works. Node A of channel L_i leaves the group, causing service disconnection for all its children. These nodes receives the patching stream by rejoining one of the free patching channel P_j. Obviously, this scheme is more efficient than the *1:1* scheme, since the video patching request will be served as long as there is a free patching channel available.



(a)Tree Structure



(b) Patching Structure

Figure 17: Static multiplex channel allocation

4.2.2.3 Dynamic Multiplexing with Channel Merging

In the *static multiplexing* scheme, the value of m and n is pre-determined and fixed throughout the video transmission process. A more flexible scheme is to assign the number of live and patching channels dynamically. In this section, we propose a *lifetime-based allocation* scheme. The lifetime of an original stream is the remaining playback time of the video program. The lifetime of the patching stream is the duration of the patching period. For a video program of length T, at time t_0, the lifetime of the original stream is $T - t_0$. As to the patching stream, the life time is the time shifting value of this stream; the lifetime for stream s_k is $d \times k$. Our allocation algorithm works as follows: For the patching stream, at time t_0, the number of patching channels reserved is:

$$|P| = \lceil \frac{\sum_{i=1}^{m}(i \times d)}{T - t_0 + \sum_{i=1}^{m}(i \times d)} \times C \rceil$$

For the original stream, at time t_0, the number of video channels allocated is:

$$|L| = C - |P|$$

The number of patching channels $|P|$ is monotonically increasing as the video program proceeds. It is possible that when the server needs to allocate more patching channels, there is no free channel available. We introduce a channel merging scheme to deal with this problem. In this solution, when the server's request for more patching channels can not be satisfied, two live channels that have the fewest number of clients in their multicast trees are merged, and the freed channel is used as a patching channel. To merge two live channels, just connect the root node of the channel with fewest members to the highest possible level of the other channel.

4.2.3 Patching Stream Selection

The *Stream selector* in the video server determines which time-shifted stream should be used to patch the missed video portion for a certain client. When a client is disconnected from the group, it records the video playback time t_d, and sends a node rejoin request to the server. Assume when the server receives this node rejoin request, the live video playback time is t_1.

If the server has an available channel at this time, it estimates the connection set up time as $t_1 - t_d$. This is the time for the rejoin signal to travel from the client to the server. Thus, stream s_i is selected, while i is calculated by the formula below:

$$i = \lceil \frac{2 \times (t_1 - t_d)}{d} \rceil$$

If there is no available channel when the rejoin message reaches the server, this rejoin message will be held until a free patching channel is available. For example, at time t_2. Then the stream s_j should be selected, while j is determined by the formula below:

$$j = \lceil \frac{(t_2 - t_d) + (t_1 - t_d)}{d} \rceil$$

4.2.4 Node join algorithm

A "well shaped" application layer multicast tree should be wide and short. A shorter multicast tree means a smaller number of overlay hops from a client to the server, thus a smaller average stretch. Stretch is the ratio of the latency along the overlay to the latency along the direct unicast path [6, 13] . The stretch of the application layer multicast tree is the average stretch value over all the clients in the tree. Furthermore , by reducing the number of intermediate clients, the chance that a client suffers service disconnection due to ancestor nodes leave is also reduced. In this section, we design a level-based scheme to handle client join. In this scheme, a newly arriving client joins the node whose overlay

Table 2: Node Join Algorithm

```
Join (C, S) {
1:   joined = false;
2:   Push the server IP address $S$ in current list $L_c$;
3:   while (joined==false and $L_c \neq \emptyset$ ) {
4:     repeat
5:       Probe the next IP address $p_n$ in $L_c$;
6:       if ($p_n$ has enough resource for new client) {
7:         $C$ join the tree and become $p_n$'s child;
8:         joined ==true;
9:         }
10:      Push the children of $p_n$ into $L_n$;
11:    until($L_c$ is visited or joined == true)
12:    $L_c = L_n$;
13:    }
14:  return joined;
15: }
```

path distance to the server is shortest, and has enough available bandwidth resource to accommodate a new client. The join algorithm is shown in Figure 2.

The patching scheme in our design requires extra bandwidth for each client, we now analyze how this scheme influences the overall structure of the tree. Assume the bandwidth of a video stream is b, and the average bandwidth of each host is $k \times b$. The height of a tree with N hosts should be $h_1 \approx log_k N$. Because of the use of the video patching scheme, each host has to allocate twice the bandwidth of a video stream: one for the original stream, the other is for the patching stream. Thus, the height of the tree is $h_2 \approx log_{\frac{k}{2}} N$. We now have:

$$\frac{h_2}{h_1} \approx \frac{log_{\frac{k}{2}} N}{log_k N} = \frac{logk}{logk - 1}$$

Table 3: Bandwidth First Node Join Algorithm

```
BWJoin($c, S, bw_c$) {
01:    joined = false;
02:    Push the server IP address $S$ in current list $L_c$;
03:    while ($L_c \neq \emptyset$ ) {
04:      Push the children of members in $L_c$ into $L_n$.
05:      repeat
06:        probe the bandwidth of next host $p_n$ in $L_n$;
07:        if ($p_n$'s bandwidth $< bw_c$)
08:           if (Join_Level($L_c$) == true) return(true);
09:        else
10:           $c$ becomes $p_m$'s parent's child;
11:           $p_m$ becomes the child of $c$;
12:           return(true);
13:        if ($p_n$'s bandwidth == $bw_c$)
14:           if (Join_Level($L_c$)==true) return(true);
16:           else
17:              if (Join_Level($L_n$)== true) return(true);
18:      until($L_c$ is visited)
19:      if ($L_n = \emptyset$)
20:           if (Join_Level($L_c$)==true) return(true);
21:      $L_c = L_n$;
22:      }
23:   return joined;
24: }
```

To further reduce the height of the tree, we design a *high-bandwidth-first* tree join algorithm. The key idea is to push the client with larger bandwidth up to the higher level of

Table 4: Level Join Algorithm

```
Join_Level($L_c$) {
1:    repeat
2:      Probe the next host $p_a$ in $L_c$;
3:      if($p_a$ has enough bandwidth)
4:         c join the tree and become $p_a$'s child;
5:         return true;
6:    until ($L_c$ is visited)
7:    return(false);
8: }
```

the tree. The multicast tree built by this scheme has the feature that clients in the lower level of the tree do not have more bandwidth capacity than the clients in the upper level of the tree. Figure 3 and Figure 4 shows the *high-bandwidth-first* join algorithm.

4.2.5 Node Leave

Clients in the application layer multicast tree may leave the group at any time. A leaf node leaving the group does not have much impact on the application layer multicast tree. An internal node leaving the group, on the other hand, will cause service disconnection for its children. There are two kinds of leave: graceful leave and node failure. With a graceful leave, the client first informs the server and its children. Its children then manage to rejoin the tree by connecting to other parent nodes. It leaves the tree after the adaptation of the tree is finished. In the case of a node failure, the client leaves the group without informing any other hosts. The tree recovery operation due to a node failure is a two step process: First, the failed node and affected region [3] detection. Second, disconnected nodes rejoin the

[3] Affected region of a client c denotes the set of all nodes that are disconnected due to node failure of c.

tree, this step follows the same operation as the rejoin process under a graceful node leave.

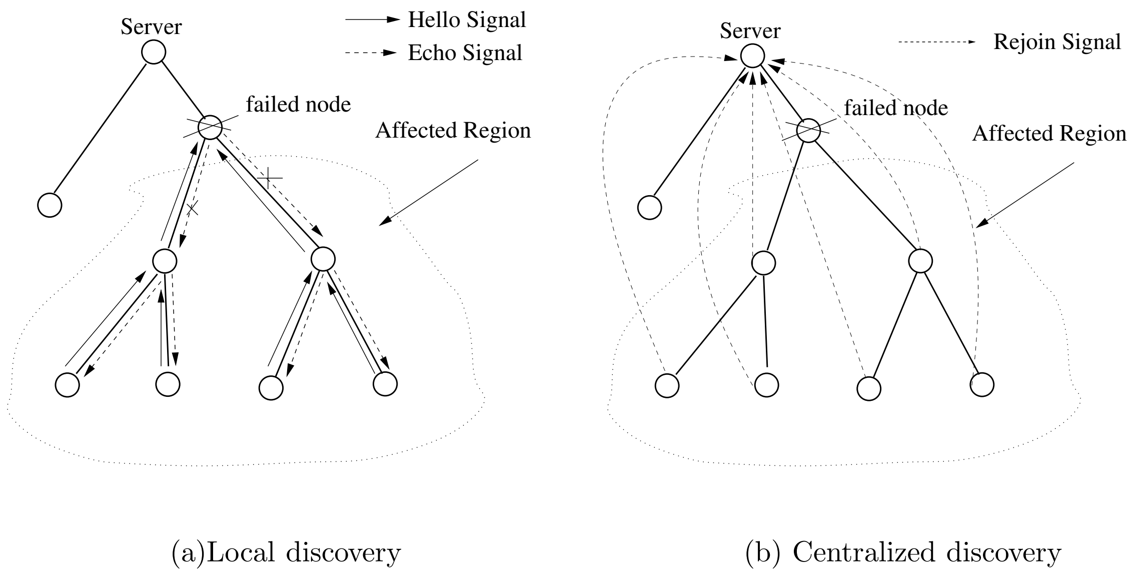


Figure 18: Failed node discovery process

There are two approaches to discover the failed node and the affected region, as shown in Figure 18. First approach is through localized detection. A client that detects a heavy packet loss sends a *hello* message to its parent node. If the parent node is experiencing the same problem, it sends an *echo* message back to the sender, and sends another *hello* message to its parent. If the parent node does not suffer packet loss, then it is the link congestion between the child and the parent causing the problem. This process repeats until a client does not receive *echo* message from its parent, then either this parent node is failed or the link between this client and its parent is disconnected. This approach does not require global topology knowledge, and detection does not exert extra load on the media server. The problem of this approach is that it might be slow in detecting the affected region. The other approach is to use the central video server. The video server maintains the topology of the application layer multicast tree. Each client that suffers service disconnection reports to the server, the server then figures out which node or link is failed, and the corresponding affected region.

The node rejoin process works like this: the affected clients first try to elect a central node. A central node is a child of the failed node with enough bandwidth resource to accommodate all its siblings. If there exists such a central node, then the central node rejoins the parent of the leaving node, and all the children of the leaving node rejoin this central node as its children. If no central node can be elected, each child of the leaving node as well as the sub-tree rooted at them rejoins the application layer multicast tree independently.

4.2.6 Viewing time and buffer requirement

In this section, we analyze the client viewing behavior and resource requirement under the case of non-consecutive node failures. Non-consecutive failure means that, for a client, the next service disconnection does not happen until the client is recovered from previous node failure, and the missing video is fixed up.

During a client's viewing process, we assume its service will be interrupted n times. Each time, the node repair time (or rejoin time) is $r_i (i = 1, \ldots, n)$. We are interested in the following aspects during the client viewing process: i) What is the maximum buffer requirement for the client? ii) What is the accumulated delay the client will experience? We draw the following conclusion:

Conclusion *The overall client side viewing delay, and the maximum client side buffer requirement under non-consecutive loss is* $max(\lceil \frac{r_j}{d} \rceil \times d)$, $j = 1, 2, \cdots, n$

Proof:

We prove by induction.

i) For $n = 1$, which means there is only one service disconnection during the viewing process. Assume the client disconnects at time t, and rejoins at time $t + r_1$. From time $t + r_1$, the client begin to receive the original stream, and at time $t + \lceil \frac{r_1}{d} \rceil \times d$, the patching stream begins to serve the client. Thus, the client experience a delay of lceil $\lceil \frac{r_1}{d} \rceil \times d$, and the buffer usage is also $\lceil \frac{r_1}{d} \rceil \times d$. Our conclusion is true for $n = 1$.

ii) Assume the conclusion is true for the case of n,

iii) Let's consider the case of $n + 1$. Assume the overall delay for first n disconnections, as well as the maximum buffer usage is $\lceil \frac{r_x}{d} \rceil \times d$. When the client is disconnected from the tree at time t for the $n+1_{th}$ time. The repair time for this service disconnection is r_y, and the selected stream has a delay of $\lceil \frac{r_y}{d} \rceil \times d$. Three cases should be considered according to the relationship between r_x, and r_y, they are shown in Figure 19. We denote $k_x = \lceil \frac{r_x}{d} \rceil$, and $k_y = \lceil \frac{r_y}{d} \rceil$.

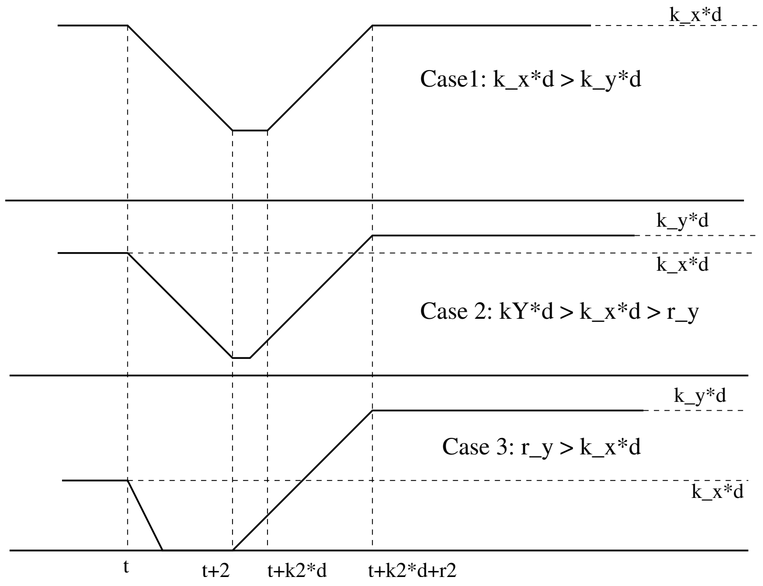


Figure 19: Client Side Buffer Size Under General Case

- *case 1:* $k_y \times d \leq k_x \times d$. At $t_y + r_y$, original stream begin to feed the client's buffer. Stream k_y is selected as the patching stream. At time $t_y + k_y \times d$, the client begins to receive the patching stream from server. Note that at time $t_y + k_y \times d$, the client is still playing from the buffer, so there is no extra delay introduced, and there is no freezing period during this time interval. At time $t_y + k_y \times d + r_y$, the missed portion is fixed. The buffer usage is: $k_x \times d - r_y + 2 \times r_y - r_y = k_x \times d$.

- *case 2:* $k_y \times d \geq k_x \times d \geq r_y$. At $t_y + r_y$, client is playing from the buffer. Original stream begin to feed the client buffer. At time $t_y + k_y \times d$, the client begin to receive the patching stream from server. The client buffer becomes empty before $t_y + k_y \times d$. Thus, the client suffer an extra delay of $k_y \times d - k_x \times d$. And the freezing period is $k_y \times d - k_x \times d$. The patching stream is played out while the original stream is buffered at the client. At time $t_y + k_y \times d + r_y$, the missed portion is fixed. The overall delay at the client is: $k_x \times d + k_y \times d - k_x \times d = k_y \times d$ latency. The buffer size used is $k_y \times d$.

- *case 3:* $r_y \geq k_x \times d$. At $t_y + r_y$, client buffer is empty. Original stream begin to feed the client buffer. At time $t_y + k_y \times d$, the client begin to receive the patching stream from server. The patching stream is played out while the original stream is buffered at the client. At time $t_y + k_y \times d + r_y$, the missed portion is fixed. The client suffer and extra delay of $k_y \times d - k_x \times d$, as well as the freezing period. The overall latency is $k_x \times d + k_y \times d - k_x \times d = k_y \times d$ The buffer size used is $k_y \times d$.

Thus, our conclusion is true for the case of $n + 1$.

4.3 Performance Evaluation

In this section, we set up the simulation environment, and evaluate our design through simulation experiments.

4.3.1 Simulation Environment Setup

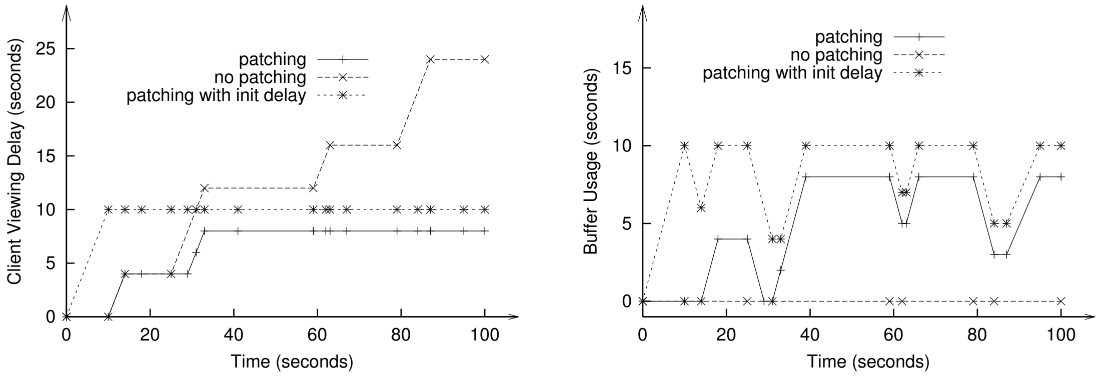
We use the GT-ITM transit-stub model to generate a network topology [8] of about 1400 routers. The average degree of the graph is 2.5, and core routers have a much higher degree than edge routers. The media server and the end hosts are connected with the edge routers. We categorize the bandwidth capacity of the end hosts into three levels. 1) Narrow bandwidth: with $1.5Mbps$ bandwidth, 70% of the end hosts are in this category. This bandwidth capacity is only enough for receiving video streams. The hosts with such bandwidth capacity can only be leaf nodes in the tree. 2) Medium bandwidth: with $10Mbps$ bandwidth, 20% of end hosts belong to this category. 3) High bandwidth: with $100Mbps$ bandwidth, only 10% of hosts have such high-speed connection.

There is one fixed media server in the topology, it has $100Mbps$ high speed connection. The time shifting value between stream s_i and s_{i+1} is 4 seconds. The server processing capability and I/O capacity is not a bottleneck compared to the server side bandwidth. We further assume links between the routers are never a bottleneck.

4.3.2 Evaluation of client video reception performance

In this section, we study the client video reception performance in our system. We are interested in the following performance metrics: *client viewing delay, maximum buffer usage, video continuity,* and *video completeness*. *Client viewing delay* means the delay between the client side playback time and the playback time of the original stream. *Maximum buffer usage* records the maximum buffer usage throughout a client's video reception process. *Video continuity* for a client is evaluated by the duration and frequency of freezing period. *Video completeness* refers to the rate between the received video and the transmitted video.

4.3.2.1 Effect of video patching



(a)Viewing Delay (b) Buffer Usage

Figure 20: Video patching on client video reception

We first compare the video reception performance of a client with or without video patching scheme. In this experiment, we record a client's viewing behavior from the time it first joins the application layer multicast tree until it leaves the group. We assume infinite client buffer size in this simulation.

Figure 20 shows the simulation result. Figure 20(a) shows the client viewing delay. Without video patching, the client suffers the longest viewing delay. Video patching scheme can greatly reduce the viewing delay. As to the buffer usage, video patching scheme requires some buffer space to store the video packets that is not being played out. The no patching scheme does not use any buffer space, as shown in Figure 20(b).

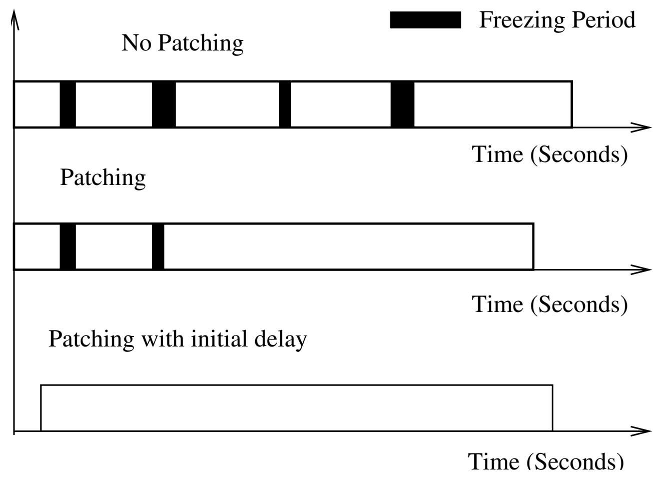


Figure 21: Client video playback performance

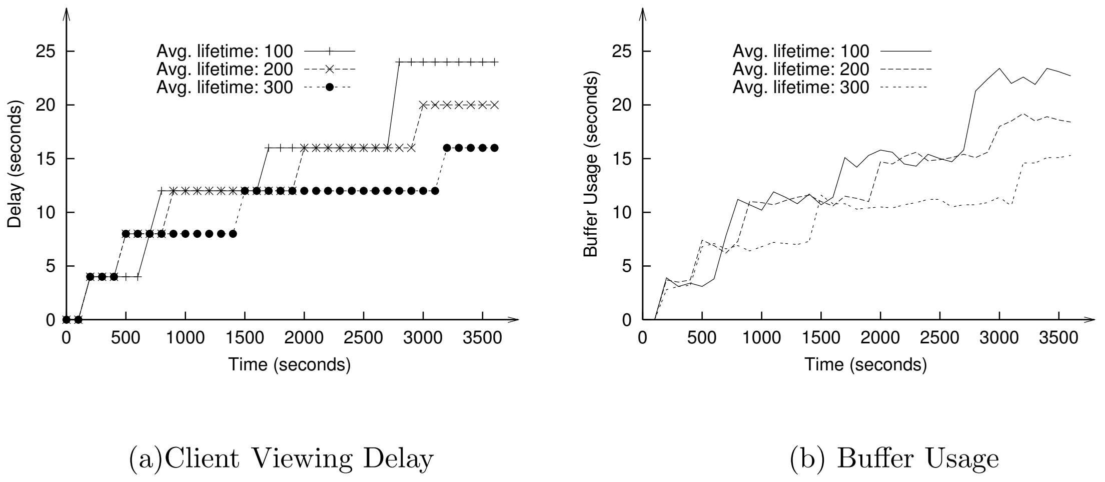
Figure 21 shows the video continuity during the viewing process. Without video patching scheme, the client suffers freezing period every time it is disconnected. With video patching, the freezing period is significantly reduced, since the client can still playout the previously buffered video content when the subsequent service disconnection happens. We also find that, if the client starts playout the video after an appropriate initial delay, the client can receive continuous video stream.

4.3.2.2 Adaptation to different traffic patterns

We now study the video reception performance of a client under different traffic patterns. In this experiment, the clients join the multicast tree according to a *Poisson* process, their average time in the group varies from $100s$ to $300s$. We manually insert a client into the group, and it does not leave the group throughout the simulation process. The simulation time is one hour, we sample the viewing delay for this client every 100 seconds, and calculate the average buffer usage during this 100 second interval.

Figure 22 shows the simulation result. The shorter the average client lifetime, the longer the client viewing delay. Since as the clients leave the group more frequently, it is more possible that the client is disconnected again before the patching period finishes. In this way, the client has to join later and later time-shifted streams each time it rejoins the multicast tree.

The client buffer usage is also increased as the clients join/leave the group in a more frequent manner. Since under this case, the clients have to join the patching stream with larger time shifting value. This causes longer patching period, and accumulates more data in the client buffer.



(a)Client Viewing Delay (b) Buffer Usage

Figure 22: Video Patching on viewing performance

4.3.2.3 Client viewing delay distribution

In this section, we focus on the viewing delay for all the clients in the system. In the simulation experiment, we record the maximum viewing delay of each client, and study their overall distribution. The average lifetime of a client is 100 seconds.

Figure 23 shows the distribution of client viewing delay. The client buffer usage demonstrates very similar distribution. The maximum client viewing delay value is 24 seconds. Most of the clients' viewing delay is between 4 and 16 seconds. The factors that affect the viewing delay are: the client rejoin latency, and the server video channel usage condition. The client rejoin latency is mostly determined by its location in the application layer multicast tree. There are some clients directly connected with the video server, these clients suffer no viewing delay at all. Those clients further away from other clients tend to need longer time to rejoin the application layer multicast tree. Another factor that influences

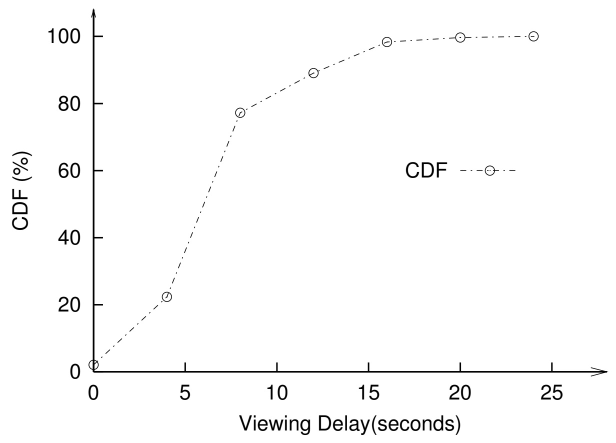


Figure 23: Viewing delay distribution

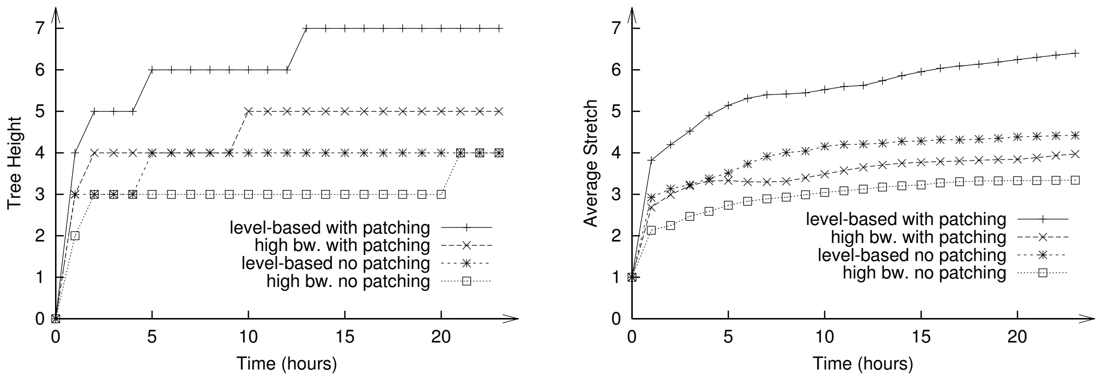
the viewing delay is the server channel usage condition. If the server does not have a free patching channel available when the client rejoins the tree, the patching stream has to be delayed, as well as the client viewing delay. Clients that suffer longer viewing delay also need more buffer space, since the patching period tends to be longer.

4.3.3 The effect of video patching on tree structure

Video patching requires extra bandwidth on the client side, which reduces the forwarding capability, in terms of fan out, of the clients. Thus, the "width" of the tree will be reduced.

In this section, we study the influence of the patching scheme on the structure of the application layer multicast tree, and the average stretch of the end hosts. In this experiment, the clients join the multicast group in a *Poisson* process, and stay in the group throughout the experiment. Thus, the number of clients in the tree is monotonically increasing. The maximum number of clients in the tree is about 100000 in our simulation. The simulation time is 1 day in this experiment. We compare four tree join algorithms in this experiment: *level-based* algorithm, with and without video patching; *high-bandwidth-first* algorithm, with and without video patching.

Figure 24(a) shows the height of the tree. The level-based tree join algorithm with video patching generates the highest multicast tree. The high-bandwidth-first algorithm promotes



(a)Level-Based Scheme (b) High-BW First Scheme

Figure 24: Influence of video patching on tree structure

the clients with high bandwidth to the upper level of the multicast tree, thus increases the fan out of the tree in the higher level. The height of the tree for the high-bandwidth-first algorithm is significantly smaller than the level-based scheme. Furthermore, for the high-bandwidth-first algorithm, the height of the tree with video patching does not increase much compared to the scheme without video patching.

Figure 24(b) shows that the level-based tree join algorithm with video patching has the worst stretch performance, and is much worse than the same tree join algorithm without video patching. For the high-bandwidth-first scheme, the shape of the tree is optimized, and the height of the tree is comparatively shorter, the stretch performance with or without video patching is close to each other.

4.3.4 Server Channel Allocation Scheme

The server side bandwidth resource (video channel) availability determines the shape of the multicast tree, and the starting time of the patching stream. How to effectively assign server channels is important to the overall system performance, and to the satisfaction of client video reception. We have proposed three video channel allocation scheme: 1:1, static multiplexing, and dynamic multiplexing. In this section, we evaluate two aspects of video channel allocation scheme: 1) video channel utilization 2) client queuing delay for the patching channel.

4.3.4.1 Server channel utilization

Server channel utilization means the percentage of the number of channels in use to the number of all the video channels. In this experiment, we assume the server can support 100 video channels simultaneously. We run the simulation for 1 hour, and record the average channel utilization value every 100 seconds. Figure 25 shows the simulation results.

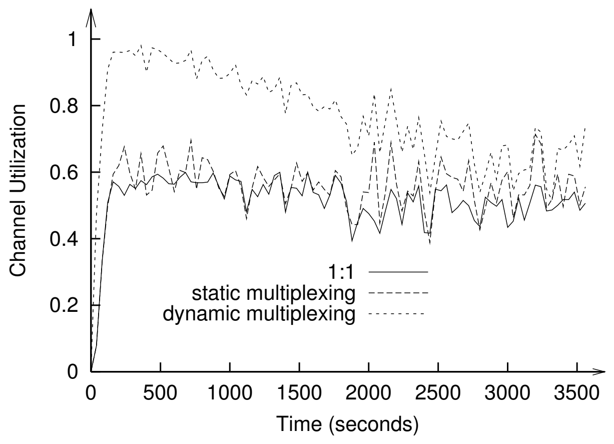


Figure 25: Video Server Channel Utilization

For *1:1* channel allocation scheme, the channel utilization value is worst. In this scheme, half of the video channels are allocated as live channel, and each live channel is bound with a patching channel. A patching channel is used only after a service disconnection in the application layer tree of the associated live channel, otherwise, it is free. And the life time of a patching channel is short compared to the live video program time. Furthermore, in *1:1* scheme, even if there are multiple rejoin requests in the application layer multicast tree of a live channel, the associated patching channel can only serve one request at a time, other requests have to be put into a queue until this patching channel is free.

In *static multiplexing* scheme, we use half of all the video channels as live channel, and reserve the other half as patching channel. The channel utilization value is better than the *1:1* scheme. Since multiple rejoin requests from the same live channel can be satisfied as long as there are free patching channels. And the channel utilization has larger variations, since when there are multiple rejoin requests, they are served simultaneously, instead of one at a time in *1:1* scheme.

Dynamic multiplexing with channel merging performs best in terms of video channel utilization. In the beginning stage, most of the video channels are used as live channel. The channel utilization value is close to 100%. As time proceeds, the server needs to reserve more channels for video patching. It does this by merging the live channels with fewest clients, and reserve these channels as patching channel. Thus, there is a degradation in channel utilization.

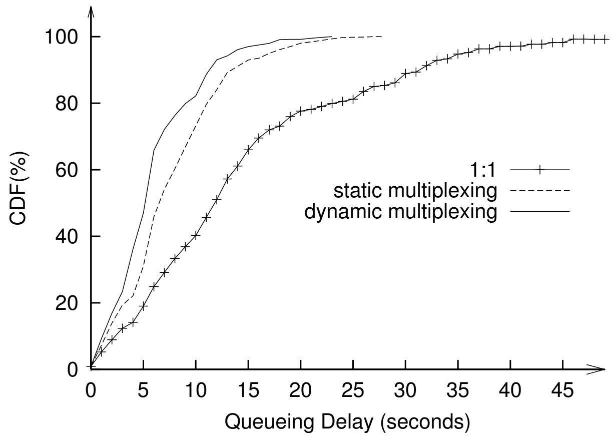


Figure 26: Client Requests Queuing Delay

4.3.4.2 Client queuing delay

Client queuing delay refers to the time when the server begins to receive the video patching request until the time this requests is being served. We compare the client queuing delay under the three video channel allocation schemes. We use the same configuration as last section, Figure 26 shows the simulation result.

We find out that the queuing delay for *1:1* scheme is significantly longer than the other two schemes. Since the patching channels are not shared in this scheme. If a node with many children leaves the group, there could be many rejoin requests at the same time. Under *1:1* scheme, only one request can be served at a time, while the other requests have to wait until this patching channel is free. For the other two schemes, they use channel multiplexing, so that the rejoin requests can be served as long as there are patching channels available. The queuing delay in these schemes is significantly reduced. For the dynamic multiplexing case, when the patching channel is not enough to handle the rejoin requests, some live channels are merged and used as patching channel. Thus, dynamic multiplexing can further reduce the queuing delay.

4.3.5 Protocol Overhead Analysis

In this section, we consider several aspects of complexity in our system design. We only give qualitative analysis in this paper, for quantitative analysis, please refer to our technical report [37].

• Message processing overhead: in our system design, the server has to process four kinds of messages: *join, leave, rejoin* and *patching end*. We assume the weight for processing these messages is the same. Our node rejoin message is composed of *join live* and *join patching* messages. The *patching end* message is coupled with a *join patching* message. Assuming there are N node join message, and M node rejoin messages throughout the video streaming process, the number of messages should be: $2 \times N + 3 \times M$. For those no-shifting, no patching solutions, the number of messages is: $2 \times N + M$. For a CoopNet solution with m descriptions, each *join, leave, rejoin* message involves operations over m trees, the number of messages should be: $(2 \times N + M) \times m$.

• Tree management overhead: the video server maintains one application layer multicast tree for the original stream, and multiple application layer multicast trees for the patching stream. Note that the patching tree has considerably smaller size and shorter lifetime than the original tree.

• Bandwidth overhead: although the MDC codec used in CoopNet introduces some bandwidth overhead [4, 35], our solution is more bandwidth intensive. Since we need to reserve same amount of bandwidth for each original stream allocated. But the adoption of the MDC approach brings extra overhead such as coding/decoding complexity, synchronization of multiple streams, etc.

4.4 Summary

In this paper, we deal with the problem of continuous live video streaming to a group of cooperative clients. Our solution is centered around a time-shifting video server, and a video patching scheme. The time-shifting video server sends multiple video streams with different time shifting values. Clients in the application layer multicast tree can retrieve the missed video content by joining the time-shifted video stream. To avoid *indefinite time-shifting* due to multiple service disconnection during the video reception process, we introduce the video patching scheme. During the patching period, a client is receiving the time-shifted video stream as well as the original stream. The video content that is not being played out immediately is stored in a client buffer. When a subsequent service disconnection occurs, a client can play the video content from the buffer while rejoining the group. In this way, the client can receive the complete video program even though the forwarding infrastructure is unreliable. Continuous video streaming is achieved if the client starts video playout after some *initial delay*.

Our design has the following features: 1) lossless video reception: by allowing clients rejoin the time-shifted video stream, the client can receive the whole video content from the point it first joined the group. 2) stable video quality: the client receives full quality video throughout the video reception process. 3) continuous video streaming: continuous video streaming can be achieved by sacrificing initial video access delay. 4) Compared to CoopNet's MDC-based system, our system has the advantage that it can use standard-based single description encoded streams. 5) Moderate complexity: the overhead of message processing and tree management is at the same level with a no-shifting, no-patching solution.

CHAPTER V

COOPERATIVE PATCHING: A CLIENT BASED P2P ARCHITECTURE FOR SUPPORTING CONTINUOUS LIVE VIDEO STREAMING

We introduce a client-based data recovery scheme that achieves continuous live video streaming among cooperative but unreliable clients. In last chapter, we describe a server-based design which uses a combined approach of time-shifting video server and P2P video patching technique. There are two major limitations in that approach: i) potential performance degradation. Video server is the only point to provide the patching stream for a client, patching stream uses significant amount of server side bandwidth. In the case where multiple patching requests are received, the server might have to delay some of the requests if there is no free channel available; ii) server complexity. The scheme requires a specially designed video server, it not only has to broadcast the live video, but also has to carry out the functionality of an on-demand video server. It has to process video patching requests, serve time-shifted video stream on demand, set up and tear down the transient trees of patching stream.

In this chapter, we describe our design of a *cooperative patching* architecture, which recovers lost video packets and achieves continuous video playback at client side. We first explain how this cooperative patching architecture works, we then argue that the cooperative patching architecture can achieve the same objective as the server-based approach through simulation experiments.

5.1 Cooperative patching architecture

In this section, we first give an overview of our design, then we describe the structure of client that's used in cooperative patching. We propose and analyze several patching parent selection algorithms at last.

5.1.1 An overview

In *cooperative patching* architecture, video server multicasts live video to receivers through an application layer multicast tree. When a client joins the multicast group, it caches the initial portion of the video that has been received. Instead of immediate playback, it delays the start of playback for a certain period. It also keeps the video that has been played out for another period before discarding it. In this design, each end host maintains a list of patching parents. A patching parent of a client refers to an end host which provides patching stream to this client, any end host currently in the multicast group can be a patching parent. When a client is disconnected from the tree due to ancestor node failure, it rejoins the tree by sending two signals. It sends a "patching" request to one of its patching parent. The selected patching parent then streams the video to recover the lost content. It also sends a "rejoin" request to join the tree and receives original video from a normal parent to keep up the progress of video reception. A normal parent of a client denotes the node which forwards original video to it.

Video patching is a channel allocation technique in Video on Demand (VoD) service. In video patching, video channels of the VoD server are classified as regular channels and patching channels. Regular channels transmit the entire video stream, while patching channels only transmit the initial part of a video as needed. When the server allocates a free channel to client request, it first scans the on-going channels. If there is no regular channel distributing the requested video, or the starting time for this video is too early for the client to patch, this channel is allocated as a regular channel. Otherwise, the channel is allocated as a patching channel, and the client receives data from both the regular and patching channel. The data from the patching channel is used to make up for the missed portion from the regular channel. When the missed portion is received, the patching channel is released, and the clients only receive packets from the regular channel. More detailed description of video patching can be seen in [45].

In this paper, we refer video patching in VoD service as classic video patching. *Cooperative patching* differs classic video patching in several major aspects: i) different objective: classic video patching is used to improve channel usage efficiency and to reduce the access latency, while *cooperative patching* is used to provide lossless and uninterrupted video reception; ii) different source: the server is the single source for the patching stream in classic video patching. In *cooperative patching*, any end host can be a patching stream source; iii) different start time and release time: classic video patching always starts at the beginning of the transmission, while cooperative patching can happen at any time during the streaming process. In classic video patching, the patching channel is released after the progress of the video catches up with the regular channel, while in *cooperative patching*, the patching channel is released after the missed video is recovered.

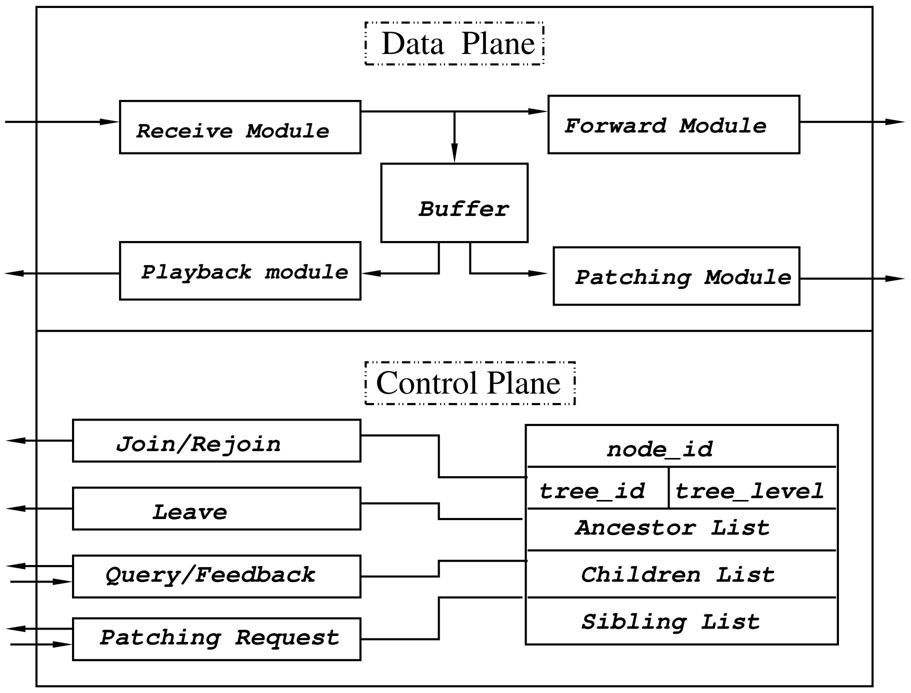


Figure 27: Peer Structure

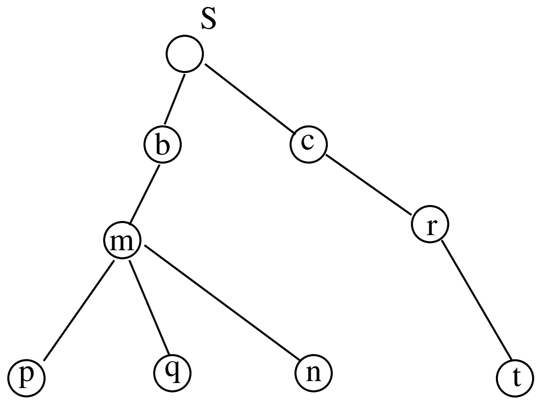
5.1.2 Peer Design

The function of a peer is composed of two parts: data plane and control plane. Data plane handles data receiving, data buffering, and data forwarding. Control plane is responsible for the signaling mechanism that maintains membership information and application layer tree structure. A typical peer structure is shown in Figure 27.

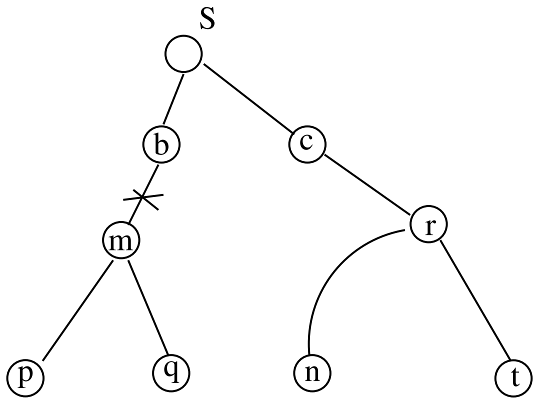
• *Data plane*: a client c begins to receive video when it joins the application layer multicast tree. It buffers d_0 time unit of video before playback. After the video content is played out, it is kept in the client buffer for another d_1 time before being discarded. For example, if c joins the application layer multicast tree at time t_0, it starts the playback at $t_0 + d_0$, the buffered video is $[t_0, t_0 + d_0]$. At time t_1, the client buffer stores video content of $[t_1 - d_0 - min(d_1, t_1 - t_0), t_1]$. When other host joins the application layer multicast tree as $c's$ child, c then starts to forward data to it.

• *Control plane:* a client c is represented by a seven tuple, i.e. $c = \{id, t_{id}, l, A, C, S, P\}$. Here, id is the IP address of c. t_{id} refers to c's ancestor node which is directly connected with the server. l is the overlay distance to the server. A is the set of ancestors, which are end hosts on the up stream of the client-server path. C is the set of children nodes of c. S denotes the sibling nodes of c. P is the list of patching parents. When a client joins the tree, it obtains the information about t_{id}, l, A, C, S from its parent node. It obtains P according to the patching parent selection algorithm. For example, as shown in Fig 28 (a), if a client $m = \{m, b, 2, \{b\}, \{p, q\}, \emptyset, \{x\}\}$. When node n joins the tree as m's children, it should be $n = \{n, b, 3, \{b, m\}, \emptyset, \{p, q\}, \{x\}\}$. When n rejoins the tree by connecting to a different parent node, it needs to update l, A, S, t_{id}, this information can be obtained from its new parent. Its previous parent and its new parent will only need to update their children list C. As shown in Fig 28(b), $n = \{n, c, 3, \{c, r\}, \emptyset, \{p, q\}, \{x\}\}$

• *Client buffer usage analysis:* we analyze the buffer usage of a client c during a disconnect-recover service cycle. We assume $d_0 = d_1 = d$, if the client doesn't suffer any service disconnection, its buffer usage should be $2d$. At time t_1, client c detects service disconnection, the client's playback point is $t_1 - d$. The buffered video content is $[t_1 - 2d, t_1]$. Assume c rejoins the original stream and patching stream respectively at time $t_1 + r_1$ and $t_1 + r_2$. The missed video is $[t_1, t_1 + r_1]$. We consider the case where $r_1 = r_2 = r$ in Fig 29,



(a)New client join



(b) Client rejoin

Figure 28: An example for client control plane

this happens when the patching parent and the normal parent is the same node.

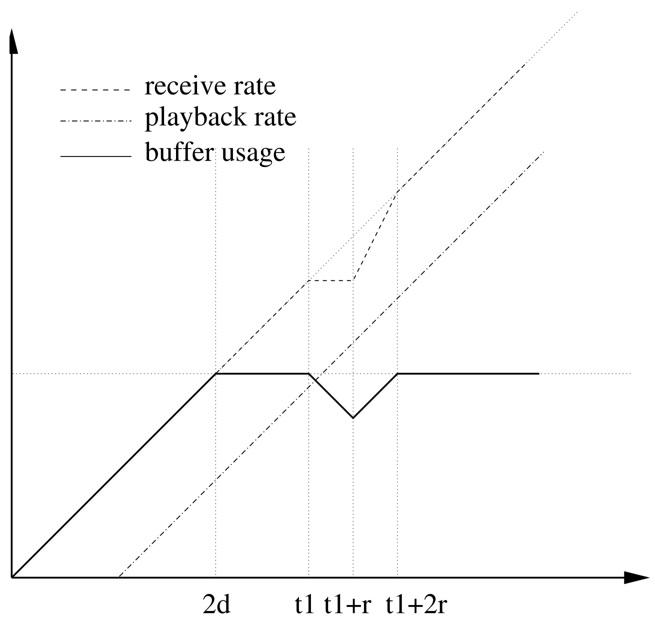


Figure 29: Case 1: $r < d$

1) $r_2 < d$: client c rejoins the group before its buffer is exhausted. Video playback is not be paused, and the buffer usage is still $2d$ after the patching is finished.

2) $d < r < 2d$: under this case, the client playback freezes before reconnecting to the tree. After it re-enter the normal play stage, the buffer usage will be $d + r$

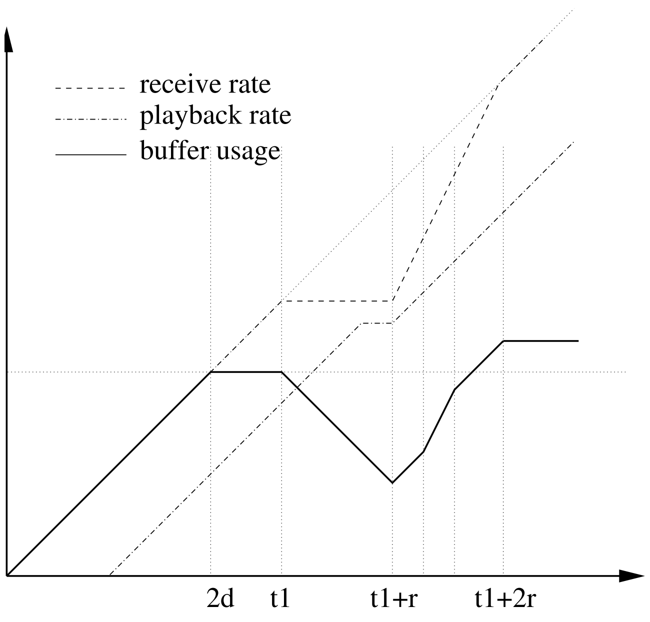


Figure 30: Case 2: $d < r < 2d$

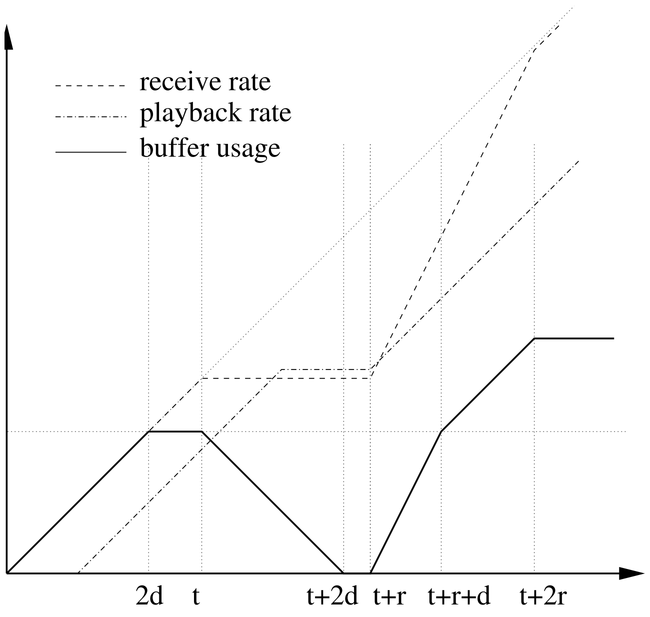


Figure 31: Case 3: $r > 2d$

3) $r_1 > 2d$: under this case, the client playback freezes and the buffer will be completely empty. After it returns to normal state, the buffer size will be $d + r$.

In conclusion, the client buffer usage should be $d + max(r, d)$. By carefully selecting the initial delay and a combined fast data recovery scheme, we can make sure that $r < d$, thus the client buffer usage will not exceed $2d$.

- *Peer State Transition Diagram:* A peer inside application layer tree can be in one of the five states: *buffering, normal receiving, patching, consuming, freezing.* As shown in Figure 32. When a client first joins the group, it is in the *buffering* state. At this state, the client buffers the received packet without playing out. After the initial buffering delay, the client enters *normal* state, where client receives the packet from its parent and plays back the video from its own buffer. In case of service disconnection, the client transforms to *consuming* state. In this state, the client is not receiving any packet but keeps playing back from its buffer. At the same time, the client starts to send out rejoin request. If the client's buffer is exhausted before reconnecting to the tree, it enters *freezing* state, the client is not receiving any packet, and client playback is paused. The disconnected client rejoins the tree by sending two signals: *rejoin original* and *rejoin patching.* When the client rejoins the application layer multicast tree, it enters the *patching* state. In this state, the client is receiving both patching stream and original stream. When the video patching is finished, the clients returns to the *normal* state.

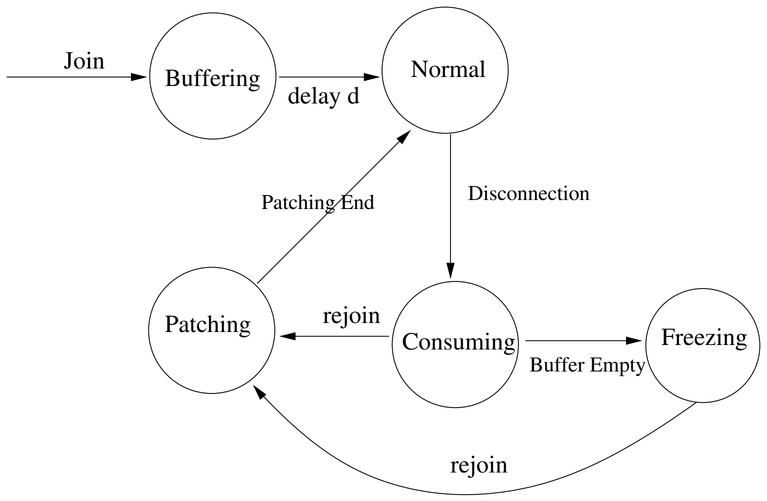


Figure 32: Peer State Transition Diagram

5.1.3 Selecting patching parents

Each end host in the multicast tree maintains a list of patching parents, it sends a patching request to these nodes when it detects service disconnection. The length of this list should be short in order to reduce the signaling overhead. At the same time, this list should guarantee a high hit ratio in order to provide timely data recovery. How to select and maintain this list of patching parents is crucial to the overall system performance. In this section, we first describe some heuristics in designing patching parent selection algorithms, we then discuss how to suppress the signaling overhead. At last, we propose a set of parent selection algorithms.

1. Heuristics in selecting patching parents

• Peer heterogeneity: in the application layer multicast tree, nodes which are closer (in terms of overlay hops) to the server generally suffer fewer service disconnections. The nodes which are further away from the server are more vulnerable, since each upstream node failure can cause them to be disconnected from the tree. In correspondence, each client in our scheme maintains different number of patching parents. Specifically, for a node l hops away from the server, it has $l-1$ patching parents.

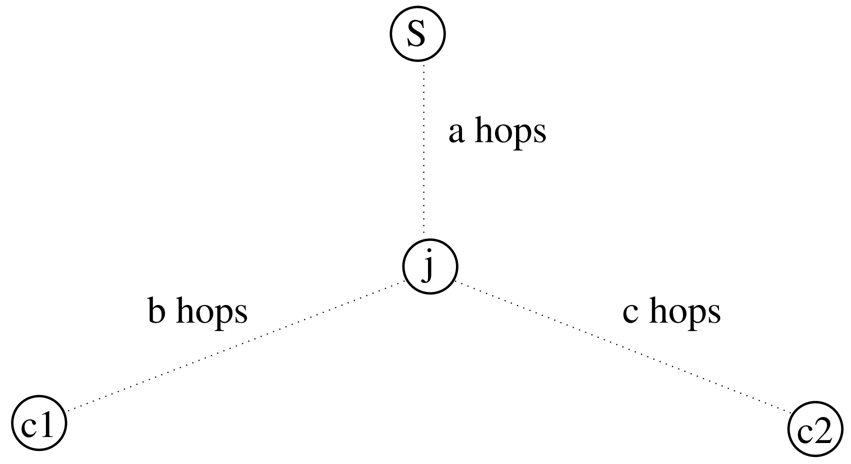


Figure 33: Node correlation

• Peer correlation: we evaluate the patching parent selection algorithm by two metrics. First, child-parent correlation, which is the correlation between an end host and its patching parent. Second, inter-parent correlation, which is the correlation between two candidate patching parents. Correlation between two nodes is evaluated by the number of shared overlay hops against the overall overlay hops. For example, in Fig 33, correlation between two nodes c_1 and c_2 is calculated by the formula below, p is the node failure probability (Note that we assume same node failure rate). A good patching selection algorithm should select the end hosts such that the client-parent and inter-parent correlation are both minimized.

$$Corr(c_1, c_2) = p^c(p^a + p^b - p^{a+b})$$

2. Suppress signaling overhead of maintaining the list

• Patching parent list maintenance: patching parents are normal end hosts in the overlay network, they can join/leave the group at any time. To keep the parent list up to date, there are generally two approaches to adopt: first, a proactive approach, once a node leaves the group, it informs the nodes which list it as patching parent to update their parent list; second, a reactive approach, a client updates its parent list once its patching request is unsatisfied due to a "node no longer in the group" message. In practice, it may not be feasible to keep complete and up-to-date information for all the hosts. One way to reduce the overhead of maintaining this information is to not insist on full accuracy and consistency. In our future research, we plan to consider the effect of using slightly out-of-date information on the performance of the selection algorithms.

• Tree based recovery structure: In application layer multicast tree, a node failure will have the whole subtree disconnected. Each disconnected client sending out rejoin request is obviously inefficient. We employ a tree-based recovery structure [58] to suppress the rejoin requests. Each disconnected client first sends a query request to its parent, if its parent suffers the same problem, it sends a feedback to its children, and sends another query to its parent. This process repeats until a client does not receive a feedback message from its parent, thus a node failure can be detected. Only the child of the failed node sends out the rejoin requests.

3. Patching parent selection algorithms

We propose following patching parent selection algorithms. Assume client c is l hops away from the video server in the multicast tree.

• Random parent selection: client c randomly select $l-1$ end hosts in the application layer multicast tree as its patching parents. This provides a baseline illustrating the performance in the absence of any information about the other end hosts.

• Direct sub-tree based random selection: assume there are m end hosts which connect directly to the video server. These m nodes are the root for the m direct subtrees, we note them as f_i, $i = 1, \cdots m$. Client c is in one of the subtree f_j. If $l < m$, this algorithm randomly picks a node in l of m subtrees. If $l \geq m$, then it first picks a node from each subtree of f_k as patching parent, and then repeat this process until l patching parents are selected.

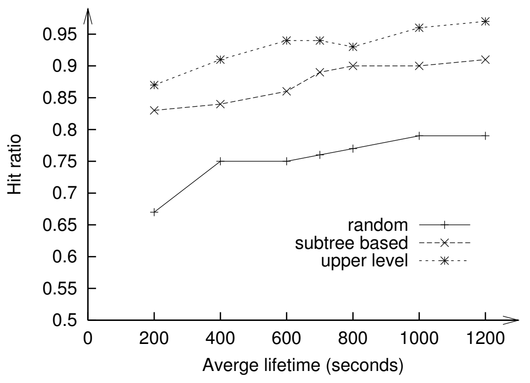
• Upper level node selection algorithm: in this algorithm, client c only selects the hosts that are closer to the video server. Since a client suffers service disconnection needs time to return to the *normal receiving* state, if it rejoins a node which is further away from the server, the probability of getting another service disconnection before fully recovered is higher. This will make the recovery process more complex and more expensive.

5.2 Performance Evaluation

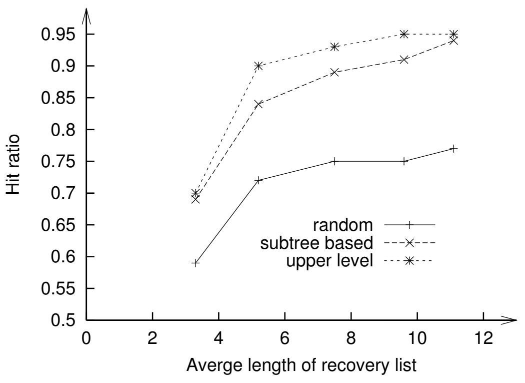
We use the GT-ITM transit-stub model to generate a network topology [8] of about 1400 routers. The average degree of the graph is 2.5, and core routers have a much higher degree than edge routers. The media server and the end hosts are connected with the edge routers. We categorize the bandwidth capacity of the end hosts into three levels. 1) Narrow bandwidth: with $1.5Mbps$ bandwidth, 70% of the end hosts are in this category. This bandwidth capacity is only enough for receiving video streams. The hosts with such bandwidth capacity can only be leaf nodes in the tree. 2) Medium bandwidth: with $10Mbps$ bandwidth, 20% of end hosts belong to this category. 3) High bandwidth: with $100Mbps$ bandwidth, only 10% of hosts have such high-speed connection. There is one fixed media server in the topology, it has $100Mbps$ high speed connection.The server processing capability and I/O capacity is not a bottleneck compared to the server side bandwidth. We further assume links between routers are never a bottleneck.

5.2.1 Parent selection algorithms

In this section, we compare the performance and overhead of different patching parent selection algorithms. In this experiment, we pre-construct an application layer multicast tree with 10000 nodes. The clients join the multicast tree according to a *Poisson* process. The average lifetime of each node in the group conforms to an exponential distribution. The total simulation time is 1 hour.



(a)Different avg. lifetime

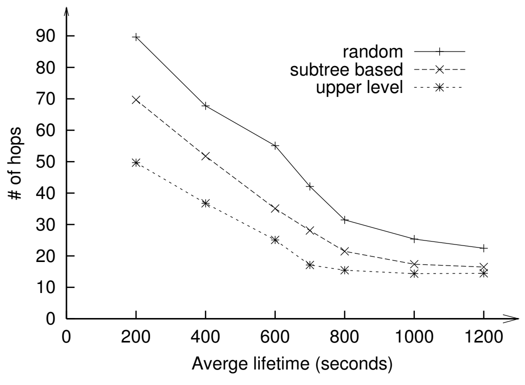


(b) Different list length

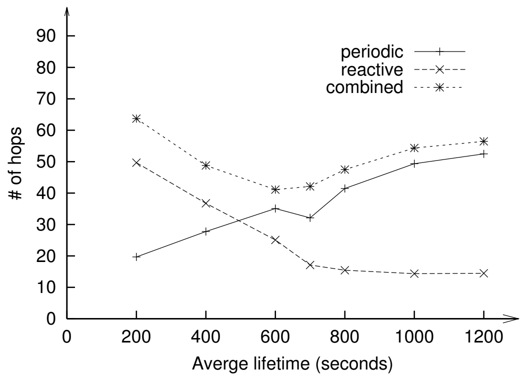
Figure 34: Hit ratio comparison

• Hit ratio: Figure 34(a) shows the hit ratio of different parent selection algorithms. The average lifetime value varies from 200s to 1200s. We can see that random selection algorithm is worst, and the subtree based algorithm performs better. As the average lifetime increases, the hit ratio increase, since the probability of multiple nodes failure is decreased. We further compared the hit ratio as the length of patching parent list varies. The average lifetime of a client is 400s. As shown in Figure 34(b), as the length of list increases, the hit ratio also increases. The increase becomes slowly as the average list length is more than 6.

• Overhead: our next experiment evaluates the overhead to maintain the patching parent list. Assume the set of clients in the tree is C, l_c denotes the tree level of the client, and h_i is the overlay hop that a client needs to locate its i_{th} patching parent. Assume a client needs about u times of patching parents update. This overhead O is calculated by the formula:



(a)Different algorithm



(b) Different scheme

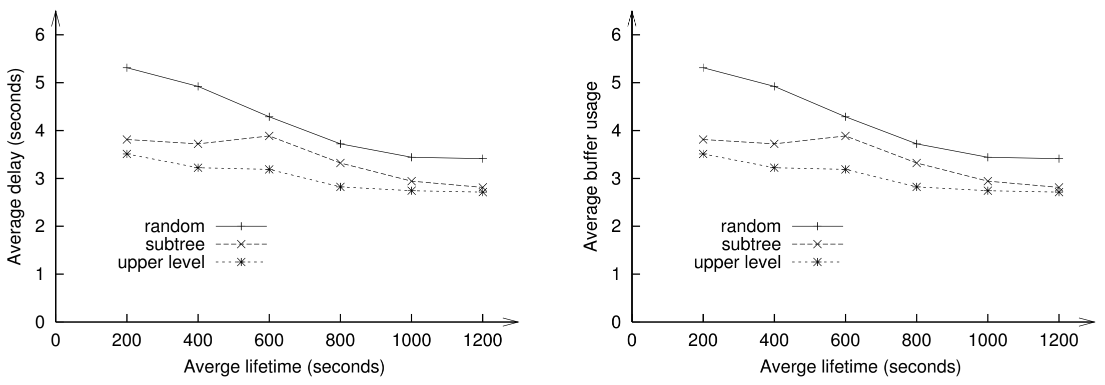
Figure 35: List maintenance overhead comparison

$$O = \frac{\sum_{\forall c \in C} \sum_{j=1}^{u} \sum_{i=1}^{l_c} h_{ij}}{\sum_{\forall c \in C} 1}$$

Figure 35(a) shows the overhead of maintaining the patching parent list under different lifetime. The number of hops indicates the number of end hosts involved in forwarding the probe message. In this scheme, the parent list update is not triggered until no valid parent is found. As we can see that as the node average lifetime increases, the list maintenance overhead is decreased. We also find out that different selection algorithms have comparable overhead. Figure 35(b) shows the overhead under different list update schemes. Proactive update here means that the client update its parent list every T seconds. Reactive means that the client update the list if no parent in the list can recover the data. Combined approach means that a client update the list once its patching request is failed, and at the same time, it refreshes the list when the timer expires. As the average lifetime increases, the reactive scheme has less overhead, since it doesn't update that often. While the proactive approach overhead increases, since the scale of the tree gets larger and the client updates its list every T time units.

5.2.2 Client video reception performance

We now study the video reception performance of a client under different traffic patterns. In this experiment, the clients join the multicast tree according to a *Poisson* process, their average time in the group varies from $200s$ to $1200s$. We manually insert a client into the group, it does not leave the group throughout the simulation process. The simulation time is one hour, we sample the viewing delay for this client every 100 seconds, and calculate the average buffer usage during this 100 second interval.



(a)Client Viewing Delay (b) Buffer Usage

Figure 36: Video Patching on viewing performance

Figure 36 shows the simulation result. We find out that the shorter the average client lifetime, the longer the client viewing delay. Since as the clients leave the group more frequently, it is more possible that the client is disconnected again before the patching period finishes. In this way, the client has to patch more and more data each time it rejoins the multicast tree. The client buffer usage is also increased as the clients join/leave the group in a more frequent manner. Since as the patching period get longer, more data will be accumulated at the client buffer.

5.2.3 Comparison with time shifting server approach

We compare the performance of cooperative patching architecture with the time shifting server based approach in this section. Figure 37 shows the performance results.

Cooperative patching can greatly reduce the viewing delay, since it removes the queuing delay at the client side. We also find out that as the average lifetime increases, the difference between two approaches is reduced. Since as the clients become more stable, the probability of multiple requests at the server is also reduced, thus reduces queuing delay. As to the

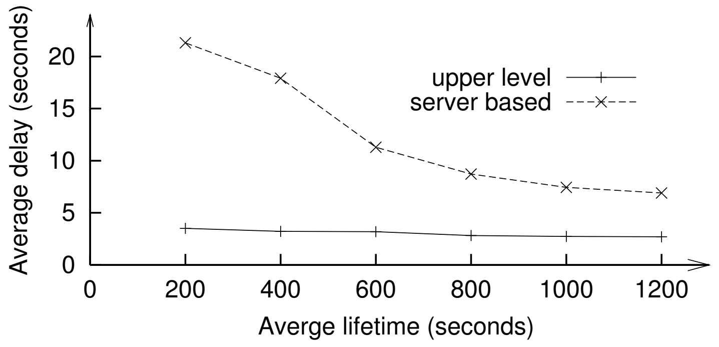


Figure 37: Comparison with the server based approach

overhead, the server-based approach is more efficient. Since in cooperative patching, an extra overhead is introduced as each client has to maintain a list of patching parents.

5.3 Summary

In this paper, we consider the problem of continuous live video streaming to a set of co-operative but unreliable peers. We propose a client based *cooperative patching* architecture to recover missed video and to maintain continuous video playback. In this design, when a client joins the application layer multicast tree, it first caches the initial portion of the video to its buffer. After playback begins, it keeps the video that has been played out for a certain time before discarding it. Each client also maintains a list of patching parents. Once a client detects service disconnection, it sends a query message to its parent, and the parent will send a feedback message back if this parent node is alive. This query process is stopped once a host does not receive any feedback. After the root of the disconnected subtree is located, it then sends a “patching” request to its patching parent, and a “rejoin” message to the server. The patching stream is stopped once the missed video content is fully recovered. One of the key technique in *cooperative patching* architecture is patching parent selection. Several parent selection algorithms are proposed, and they are verified to be reasonable through simulation experiments.

Our design achieves lossless video reception by recovering the missed video from another client in the tree, continuous streaming is achieved by sacrificing initial video access delay. With appropriately selected initial access delay, no extra delay is introduced even though there is service disconnection. As to the complexity, this design has the same level of message processing and tree management overhead compared to our previous work. Our design also has the advantage of no need of modification on existing video system. Video data recovery is shifted to the client side. By replicating video data at multiple locations, the delay for starting the patching stream is also significantly reduced.

CHAPTER VI

V3: A VEHICLE TO VEHICLE VIDEO STREAMING ARCHITECTURE

Streaming live video content to driving vehicles can provide a very attractive service. Many applications can benefit from such a service: driver assistance and safety applications, business or entertainment applications, and military or scientific applications. As to how to support video streaming service to vehicles on the road, there is an *infrastructure-based* approach, and there is a *vehicle-to-vehicle (V2V)* approach. In the infrastructure-based approach, video cameras as well as video servers are deployed across the road network. These video servers are connected through a network of base stations or simply through the Internet. Video data is propagated along the infrastructure before hand off to the mobile end host-the vehicle. The infrastructure-based approach generally suffers from high base station deployment and maintenance cost. Furthermore, it lacks the flexibility and adaptability to cover every corner of the entire road network.

In this chapter, we investigate the problem of supporting a video streaming service using a vehicle-to-vehicle (V2V) approach. Video streaming through a V2V network is practical with the advance of wireless communication and video compression techniques. On the other hand, video streaming in V2V networks is challenging since V2V network is highly dynamic and highly partitioned. It is extremely difficult to provide continuous video streaming service with acceptable user access latency.

We propose *V3*, an architecture to support video streaming applications in V2V networks. This architecture is composed of two parts: a *video source trigger sub-system* and a *video data transfer sub-system.* The video source trigger sub-system uses a novel signaling mechanism to continuously trigger video sources to send video data back to receivers. The video data transfer sub-system adopts a *store-carry-and-forward* approach to transmit video data in a partitioned network environment. In this chapter, we first describe our design of

V3 by focusing on a special case where a single receiver is interested in a single destination region. A more general case involves both multiple receivers and multiple destination regions, as shown in Table 1. We then study the impact of multiple receivers on the system design, and propose a multicasting framework to support a streaming video service from multiple destination regions to multiple receivers.

6.1 Problem Description

In video streaming applications over V2V networks, the receivers are vehicles on the road, while the video source can be either other vehicles or just a stationary video server with wireless capability. Based on mobility and the number of video sources, we categorize these applications into four classes:

- Single stationary source (SSS): In such applications, a single stationary video source streams live video to the receiver vehicle. Typical applications include a video camera at the intersection of some major street broadcasting traffic information; or a roadside hotel's video server pushing commercial information to drivers on the road.

- Multiple stationary sources (MSS): In case of big events, such as a disaster report, multiple cameras are needed to cover the potentially large affected area. In such applications, the receiver can switch between the views of different cameras, or integrate the video from multiple cameras to get a complete view.

- Single mobile source (SMS): In such applications, a vehicle that drives on the road is the video source. The receiver in such applications is interested in the video data that this vehicle provides, no matter where it is.

- Multiple mobile sources (MMS): In such applications, multiple vehicles that drive past a certain area are the video sources. Each vehicle captures a portion of the video data about that area and sends it back to the receiver.

In this paper, we focus our design on MMS applications, as they are the most challenging ones. A sample scenario for an MMS application is shown in Figure 38. In this figure, the

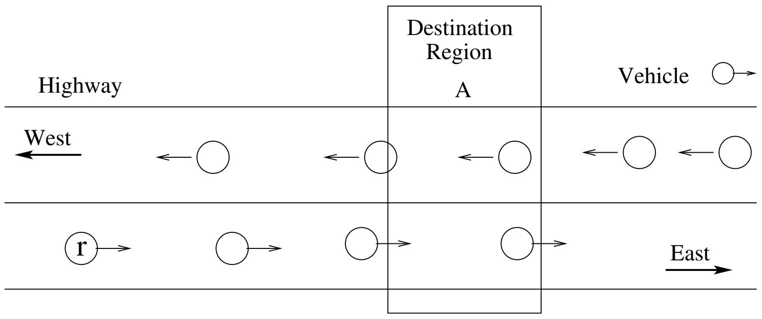


Figure 38: A sample scenario

vehicles are driving on a two-way highway. A receiver vehicle r is interested in video captured within destination region A. Vehicles that are currently in the destination region can capture valid video data and send it back to the receiver. A vehicle that captures valid video data is called a *source vehicle*. Each source vehicle only stays in the destination region for a short time period, the receiver, therefore, needs to rely on video data from multiple source vehicles to achieve continuous video playback.

We formalize the problem description as below: V represents all the vehicles on the road. A set of receivers $R \subset V(R = R_1 \cup R_2 \cdots \cup R_k)$ wish to obtain video information about destination regions A ($A = A_1 \cup A \cup \cdots \cup A_k$) respectively. A_i ($i \in [1..k]$) is a certain geographic region along the road network. A receiver r_{ij}, the j_{th} receiver interested in A_i ($1 \leq i, j \leq k$), then sends a request $Q = (r_{ij}, t_0, l_0, A_i, D_{ij})$ towards the destination region. In this request, r_{ij} is the unique ID of the receiver, t_0 is the time when the request is generated, l_0 is the initial location of r_{ij}, A_i is the destination region, D_{ij} sets the deadline of the data request. In general, deadline requirement $D_{ij} = \{G_{ij}; t_{ij}\}$ includes a geographic region requirement G_{ij} and a time requirement t_{ij}. It means that the video data that is received after receiver r_{ij} arrives within area G_{ij}, or after time t_{ij} should be discarded. Our objective is to design an architecture which supports any receiver r_{ij} to receive video data about destination region A_i and satisfies the deadline requirement.

6.2 *V3: A Single Receiver Scenario*

6.2.1 Design overview

Our architecture is composed of a *video triggering sub-system* and a *video transmission sub-system*. The video triggering sub-system is responsible to forward *video trigger messages* to the destination region, while the video transmission sub-system sends video data back to the receiver. The video trigger messages are signals that instruct a particular vehicle to turn on its video camera and to start capturing video. In this section, we first show an example of a typical service process, and describe how the architecture operates based on the example.

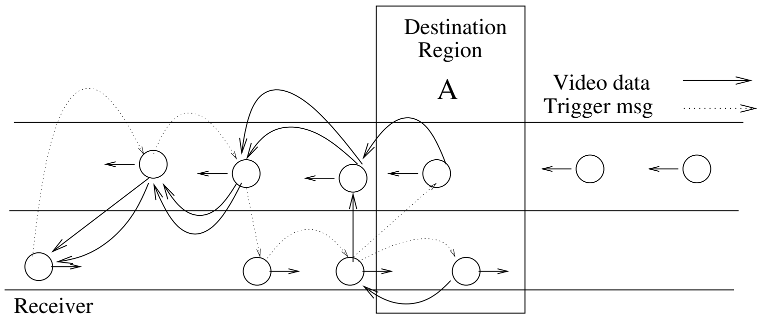


Figure 39: A typical service process

As shown in Figure 39, vehicles are driving on a road which supports two way traffic. The destination region A is a rectangular area which covers a certain portion of the road. A vehicle r that wishes to receive streaming video service about A is called a *receiver*. Vehicles that are in the destination region can capture video information desired by that receiver. The service process in our architecture is a two-step process. First, the receiver sends a request towards A. This request is forwarded by other vehicles on the road until it reaches the vehicles in A; Second, upon receiving the request, the vehicle currently in the destination region begins to capture video data and sends it back to this receiver. We describe these two steps below:

1. **Triggering the video source:** We use results from Geocasting [54, 55, 56] to develop a scheme that forwards trigger messages to the destination region. We use a directed

flooding approach to send the trigger message to the destination region. In our scheme, upon receiving a trigger message, a vehicle first calculates if it is in the *trigger message forwarding zone (TMFZone).* If so, it forwards the trigger message to other vehicles within its radio range. Otherwise, it buffers the trigger message, and re-calculates its location every α time units. We will describe the details of TMFZone calculation in the next section. If a vehicle in the destination region receives the trigger message, it begins to capture video data and sends the video back to the receiver. To provide continuous video information to the receiver, vehicles that are not in the destination region prepare themselves upon receiving the trigger message, and begin to capture video immediately when they enter the destination region.

2. **Transmitting the video data:** We use a store-carry-and-forward scheme with two extensions to transmit video data to the receiver. First, in our scheme, a vehicle has to be in the *data forwarding zone (DFZone)* to be able to forward data to other vehicles. Second, there are multiple *forwarders* which forward data to the neighbor vehicles. A source vehicle always tries to forward data to other vehicles within its radio range. A vehicle that receives and buffers the video data becomes a *data vehicle.* Not all the data vehicles will forward video data to its neighbors. A data vehicle that is assigned the responsibility to forward video to its neighboring vehicles is called a *forwarder.* A forwarder v_d in the DFZone constantly detects the existence of other vehicles within its radio range. If there exists a vehicle v_0 in v_d's radio range, and v_0 does not have the data in $v_d's$ buffer, v_d then forwards its data to v_0. Now v_0 becomes a data vehicle, if it is also assigned as a forwarder, it will forward data to its next hop vehicles. This process repeats until the data reaches the receiver.

6.2.2 Triggering the video source

Not all vehicles driving in the destination region are necessarily source vehicles. A trigger message can turn any vehicle with required capability into a source vehicle. A typical trigger message format is shown in Table 5. In this message, the receiver has a unique *ID* to identify itself. *Time* and *location* denote the initial state of the receiver vehicle.

Table 5: Trigger message format

Data Field	**Description**
ID	ID of the receiver vehicle
Time	the time when the query is sent
Location	location of the query vehicle
Velocity	current speed and driving direction of the query vehicle
Destination	geographic description of the destination region
Deadline	the expected delay constraint

$Velocity$ describes the current speed and direction of the receiver. *Destination* describe the geographic location of the destination region. Video data that arrives after the *Deadline* is invalid and will be discarded.

In this section, we address the following issues concerning the video triggering scheme in our architecture: i) how to send a trigger message to the destination region, ii) which vehicle should be triggered, iii) how to achieve continuous streaming service with transient video sources.

• *Trigger message routing scheme.* Two classes of routing protocols can be used to route the trigger message to the destination region: flooding based protocols and non-flooding based protocols [5, 61]. We use a directed flooding approach for the following reasons: i) The overhead is tolerable, since a request message is very lightweight compared to video traffic; ii) A flooding-based scheme provides shorter delay and higher reliability; iii) A flooding based approach enables continuous triggering, while a unicast based approach does not. Continuous triggering is a technique which triggers a set of vehicles to become video sources and provide continuous video streaming service about the destination region.

Directed flooding tries to limit the message overhead by defining a forwarding zone, which comprises a subset of all network nodes. Our routing scheme differs significantly from previous directed flooding protocols in the forwarding zone calculation specifics. The TMFZone in our scheme is not just a geographic region, it is defined by location, vehicle

Table 6: TMFZone Calculation

Current Location	**Driving Directions**	**Current Time**	**In TMFZone ?**
$*$	$*$	after boundary	No
$*$	towards A	before boundary	Yes
$d(A, v) > d(A, r)$	away from A	$*$	No
$d(A, v) < d(A, r)$	$*$	before boundary	Yes

mobility, and time boundary. For a vehicle v, its TMFZone calculation is shown in Table 6. In this table, $*$ means this item can be any value. Note that $d(A, v)$ and $d(A, r)$ are the distance from vehicle v and r to the destination region.

As to the time boundary estimation, a straightforward solution is to use the time deadline as the time boundary. If the current time is after the deadline, a vehicle obviously does not need to forward the trigger messages. A better solution is to use the trigger message travel time as an estimator, if the deadline minus the current time is smaller than the trigger message's travel time, it is highly possible that, when the video data from the destination region travels back to the receiver, the deadline has already passed. Specifically, if $t_c - t_s > t_d - t_c + \delta$, this vehicle is out of the TMFZone and then stops forwarding the trigger message. Here, t_c, t_d, t_s refer to current time, deadline, and trigger message start time, respectively. The δ is used to make up the error caused by a receiver's location change while it is driving towards the destination region.

• *Video source selection.* When the request message reaches the destination region, any vehicle in this area can be triggered into a source vehicle. If there are multiple vehicles in the destination region simultaneously, only one of them needs to be triggered. We design a video source selection scheme to trigger the vehicle which will possibly stay in the destination region the longest. This value can be calculated by the estimated distance to the exit point over the speed of the vehicle, using the formula below. In this formula, $cLocation_v$ is the current location of vehicle v, while $ePoint_v$ is the exit point of v.

$$t_{in} = \frac{dist(cLocation_v, ePoint_v)}{speed(v)}$$

• *Continuous trigger.* Each driving vehicle only stays in the destination region for a short period of time. To provide continuous video streaming service to the receiver, continuous trigger messages have to be sent to the vehicles in the destination region. In this architecture, we use a combined approach of pre-trigger and post-trigger to achieve continuous triggering. In pre-trigger, before the request message reaches the destination region, vehicles that receive the request become "potential video sources". As they arrive at the destination, they can turn themselves into video sources immediately. In post-trigger, a vehicle keeps forwarding the trigger message even after it has already left the destination region. Thus, vehicles driving towards the destination region from the opposite side of the receiver can be triggered into video source.

Table 7: DFZone Calculation

Current Location	**Driving Direction**	**Current Time**	**In the DFZone ?**
*	towards r	before boundary	Yes
*	*	after boundary	No
$d(r, v) < d(r, A)$	away from r	before boundary	Yes
$d(r, v) > d(r, A)$	away from r	*	No

6.2.3 Transmitting data to the receiver

Our video data transmission scheme differs from traditional store-carry-and-forward approach in two aspects: First, we propose a novel DFZone calculation algorithm. Second, we employ multiple forwarders to forward video data to the vehicles in next partition. In this section, we address several issues concerning these two aspects.

• *Calculating the DFZone.* Our DFZone calculation algorithm also considers the location, the mobility of the vehicle, and the current time. A source vehicle that has not forwarded any data to other vehicles, will seek to forward video data as long as the time boundary is not passed. Once a source vehicle has forwarded its data to other vehicles, it downgrades itself to a normal vehicle. For a normal vehicle v, its DFZone calculation algorithm is shown in Table 7:

Note that time boundary estimation in DFZone calculation is different with that in the TMFZone calculation. In TMFZone calculation, time boundary estimation needs to consider both the time it takes for a trigger message to reach the destination region, and the time needed for the data to be transmitted back to the receiver. While in data forwarding phase, only the time needed for video data to reach the receiver is considered. We use the formula below to estimate the time for the data to reach the receiver. In this formula, t_{est} denotes the estimated data arrival time to the receiver, t is the current time, $gTime$ is the time when the video data is generated, $d(l_1, l_2)$ is the distance between location l_1, and l_2. We use the average speed of data travel from the destination region to the current location as the current speed, and calculate the estimated data arrival time. If $t_{est} \leq deadline$, and the location and vehicle mobility limit are both satisfied, then v_d is in the forwarding zone.

$$t_{est} = t + \frac{d(cLocation, rLocation) \times (t - gTime)}{d(destination, cLocation)}$$

• *Forwarder selection algorithm.* Clearly, it is undesirable to have all the data vehicles forward data to other vehicles in their radio range. On the other hand, assigning only one forwarder will reduce data delivery reliability and cause longer delay. In V3, a subset of data vehicles are selected as forwarders. Based on the amount of knowledge that a vehicle can assume, we design several different forwarder selection algorithms: i) No knowledge: each vehicle only knows its own mobility information. In this case, a *random selection* algorithm can be used. In this algorithm, when a vehicle v_d forwards data to a group of vehicles V_c, it randomly select a set of vehicles $V_f \subset V_c$ as forwarders; ii) Partial knowledge: each vehicle knows the destination of other vehicles in its radio range, but does not know the detailed mobility information. We use a *direction based selection* algorithm in which the vehicles that drive towards the receiver can be potential forwarders. A *destination based selection* algorithm selects from the vehicles whose destination are closest to the receiver's location.

iii) Full knowledge: each vehicle knows the mobility function of other vehicles within its radio range. A *fastest forwarder* algorithm which picks the vehicles that could reach the receiver the fastest is used.

6.3 V3: Streaming video to multiple receivers

In this section, we discuss the impact of multiple receivers on system design, and propose our design for a multicasting framework in V3. We first describe our solution for streaming video to multiple receivers from a single destination region; We then describe how to schedule the transmission of video data in case of multiple destination regions.

6.3.1 The impact of multiple receivers

The preliminary design of V3 can support streaming video service to multiple receivers from a single destination region. A straightforward solution ignores the existence of other receivers, and serves each receiver's request independently. If multiple trigger messages or video segments are buffered in the same vehicle, they are forwarded separately on a first come first serve (FCFS) basis. This solution is obviously inefficient, since each vehicle will have to send out duplicated data packets to the next hop vehicle. The system performance will also be degraded: due to the limitation on bandwidth resources and vehicle contact time, sending redundant video information will certainly increase the end to end delay.

In the scenario where different receivers are interested in the same destination region, there are two kinds of redundancy in the system: i) Destination redundancy: the trigger messages share the same destination region. The same set of vehicles will be triggered once they enter the destination region. Thus, when multiple trigger messages reside on the same vehicle, only one *integrated* trigger message needs to be sent out; ii) Data redundancy: any video data captured within the destination region is valid for all the receivers as long as it satisfies the deadline requirement. Sending duplicated video packets through multiple unicast streams is inefficient and unnecessary. A better solution is using multicast to stream video data to these receivers.

In this design, providing live video streaming service to multiple receivers follows a similar architecture as the single-receiver version of V3 architecture. The system is also composed of a video source trigger sub-system and a video data transfer sub-system. The video source trigger sub-system uses a flooding-based approach, and employs a message integration scheme to suppress the signaling overhead. An explicit multicasting scheme for a partitioned network environment is used to forward video packets to a group of receivers. We propose a set of multicast routing algorithms based on different knowledge oracles. Currently, we focus on relatively small group of users. Very large scale multicasting is beyond the scope of this paper.

6.3.2 Integrated trigger message forwarding

We use a flooding-based approach to send the trigger messages to the destination region. Since these trigger messages share the same destination region, a vehicle that buffers trigger messages from more than one receiver can then integrate the receiver information, and forward an integrated trigger message to the next hop.

Table 8: Integrated trigger message

Destination Region			
ID_1	$Location_1$	$Velocity_1$	$Deadline_1$
ID_2	$Location_2$	$Velocity_2$	$Deadline_2$
$\cdots$	$\cdots$	$\cdots$	$\cdots$
ID_k	$Location_k$	$Velocity_k$	$Deadline_k$

• **Integrated trigger message:** A typical integrated trigger message is shown in Table 8. In this message, *Destination* describes the geographic location of the destination region. Each receiver r_i $(1 \leq i \leq k)$ has a unique ID_i to identify itself. $Time_i$ and $location_i$ denote the initial state of r_i. $Velocity_i$ describes the initial speed and direction of r_i. Video data that arrives after $Deadline_i$ is invalid and will be discarded. In the integrated trigger message, the extended deadline should be: $Deadline_{ext.} = max\{Deadline_1, Deadline_2, \cdots, Deadline_k\}$. In this way, as long as the trigger message arrives at the destination region earlier than the extended deadline, the vehicles inside this region will be triggered into a video source. It is worth pointing out that, once a receiver r_j's time requirement has expired, the corresponding entry should be deleted from the integrated trigger message.

• **Trigger message routing scheme:** The key to the trigger message routing scheme is TMFZone (trigger message forwarding zone) calculation. For an integrated trigger message, TMFZone$_{ext.}$ = TMFZone$_1 \cup$ TMFZone$_2 \cup \cdots \cup$ TMFZone$_k$. As long as a vehicle is inside any receiver's TMFZone, it will forward the trigger message to its next hop vehicles. An intermediate vehicle, upon receiving the trigger message, first calculates TMFZone$_{ext.}$. If it is in TMFZone$_{ext.}$, it then forwards the trigger message towards the destination region. Specifically, assume that an intermediate vehicle v, at time t, buffers extended trigger message *TRIG*. The algorithm to decide whether a vehicle is inside the extended TMFZone is shown in Figure 9. In this algorithm, t_{is} denotes the trigger message generation time of vehicle r_i:

Table 9: TMFZone calculation algorithm

```
bool TMFZone(v, TRIG, t) {
1:   k = number of receivers in TRIG;
2:   for i= 1 to k {
3:     Extract the $i_{th}$ receiver info
4:     if ($t - t_{is} < Deadline_i - t$) {
5:       if ($r_i$ drives towards A) {
6:         return $true$;
7:         }
8:         else {
9:           if ($d(A, v) < d(A, r)$) return $true$;
10:          }
11:      }
12:    }
13:  return $false$;
14: }
```

6.3.3 Multicast routing framework

A vehicle receiving an integrated trigger message starts capturing video data upon entering the destination region. A naive solution which sends duplicated data using multiple unicast streams to different receivers is inefficient and can cause excessive viewing offset. We design a multicast routing framework to deliver video to a group of receivers in highly partitioned and highly dynamic V2V networks.

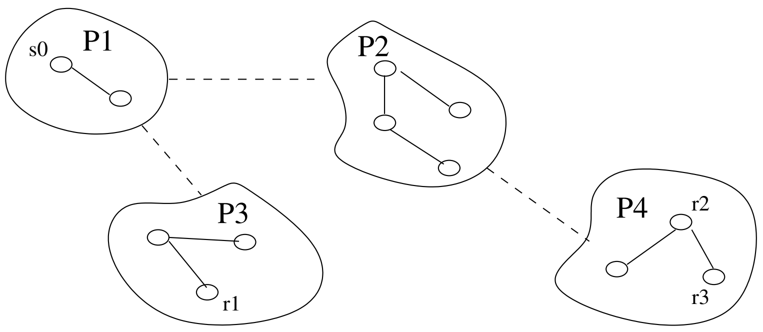


Figure 40: Multicast Routing Approach

• **Multicast routing approach:** The multicast routing approach in this paper is demonstrated in Figure 40. In this figure, s_0 is a source vehicle, and r_1, r_2, r_3 are receivers that wish to obtain data from s_0. The V2V network has four partitions [1]: P_1, P_2, P_3 and P_4. We use a two-level tree-based routing approach for data delivery purposes. The bottom level is intra-partition communication. Since the network is connected inside each partition, traditional multicasting schemes can be used [16]. A multicast tree is formed inside each partition, video data is then propagated along this tree. The top level is responsible for inter-partition communication. In this level, each unit is a partition of vehicles. In Fig 40, the dashed line denotes a potential link between two partitions. There is a multicast tree which connects different partitions of the V2V network through these potential links. Note that in a highly partitioned V2V network environment, data communication between two partitions is carried by a store-carry-and-forward mechanism. A vehicle responsible for inter-partition data forwarding might have to buffer the data for a significant amount of

[1]A partition refers to a group of vehicles in V2V network that forms a connected graph.

time before sending it to the next hop vehicles.

Specifically, a source vehicle constantly seeks to forward data to vehicles that are within its radio range. Any vehicle that receives valid video data becomes a data vehicle. Inside each partition, video data is propagated along a multicast tree. These partitions can be further categorized in two classes: *receiver partition*, and *normal partition.* A receiver partition has at least one receiver, while a normal partition does not have any receivers. For a receiver partition, all the receivers in this partition have to be included in the tree. Each data vehicle calculates if it is in the DFZone (data forwarding zone). A subset of data vehicles within the DFZone are elected as forwarders. Only the elected forwarders will seek to forward data to vehicles of next hop partitions. This process repeats until the data reaches the receiver. Since each integrated trigger message may contain multiple receivers, the data forwarding process continues until all the receivers are reached or the extended deadline has expired.

• **Data forwarding zone (DFZone) Calculation:** A data vehicle might buffer video data that are destined for multiple receivers. Similar to TMFZone calculation algorithm, in this design, the extended DFZone is the union of all the DFZones from all the receivers. $\text{DFZone}_{ext.} = \text{DFZone}_1 \cup \text{DFZone}_2 \cup \cdots \cup \text{DFZone}_k$. Assume that a data vehicle v, at time t, buffers video segment SEG_v. The algorithm to decide whether a vehicle is inside the extended DFZone is shown in Figure 10.

• **Multicast routing algorithms:** We propose several routing algorithms that select the next hop forwarders based on the knowledge oracles.

- Null knowledge: each vehicle has no knowledge about the driving behavior of its neighboring vehicles. We use a *random selection* algorithm, in which a random set of forwarders are elected in each partition to forward data to the next partition.
- Partial Knowledge: a vehicle can obtain the current location, velocity, and destination information about its neighboring vehicles. *i) Direction-based selection:* a vehicle knows the driving direction of its neighboring vehicles. Thus, a data vehicle can then

Table 10: DFZone Calculation Algorithm

```
bool DFZone(v, $SEG_v$, t) {
1:  k = number of receivers in $SEG_v$;
2:  for (i= 1 to k) {
3:    Extract the $i_{th}$ receiver information;
4:    $t_{est} = \frac{distance(cLocation, location_{r_i})}{distance(location_A, cLocation)} \times (t - t_g)$;
5:    if $(t + t_{est} \leq Deadline_i)$ {
6:      if ($r_i$ drives towards A) {
7:        return $true$;
8:        }
9:        else {
10:         if $(d(r_i, v) < d(A, r))$ return $true$;
11:         }
12:     }
13:   }
14:  return $false$;
15: }
```

forward the data to those that are driving towards the set of receivers. *ii) Destination-based selection:* a vehicle can obtain the destination of its neighboring vehicles. Thus, it is meaningful to bring up the concept of branch point. As we assume that each receiver will report their initial location and estimated trajectory. Thus, the data will be split to different groups, and each forwarder is designated for a different set of receivers.

- Complete knowledge: a data vehicle not only knows the driving behavior of it neighboring vehicles, but also has global knowledge of road traffic conditions, and each vehicle's exact trajectory. We propose an *optimal selection* algorithm: in this case, a vehicle that can reach the next partition the fastest is selected as a forwarder to that partition.

6.3.4 Streaming video from multiple destination regions

In a more general situation, different receivers are interested in different destination regions on the road. In this case, it is highly possible that, at a certain time t, an intermediate vehicle buffers video segments that are captured from multiple video sources. How to schedule the transmission of these data is vital to the overall video reception performance for all the receivers.

The problem of scheduling determines which data will be forwarded when there is a chance to forward the data. We propose a set of scheduling algorithms based on the knowledge that this vehicle possesses.

- *Null knowledge:* The intermediate vehicle does not have any information about the video data. In this case, a *first come first serve (FCFS)* scheme is used: the vehicle forwards the data in the order that it receives them.

- *Video data Knowledge:* The intermediate vehicle can obtain application layer information of video data. We propose a *earliest deadline first* algorithm: the vehicle forwards the data with the earliest deadline.

- *Complete knowledge:* A vehicle not only knows the deadline of each data segment, but also knows the the time it takes to forward this data to the receiver. We define an *emergency index*, which is used to determine the order of data forwarding. At time t, for a video data from destination region i to receiver v_j, its emergency index value is $E_{ij} = (Deadline_j - t) - (t_{arrive} - t)$. Here, $Deadline_j$ indicates the deadline requirement for receiver v_j, and t_{arrive} denotes the time when the data arrives at the designated receiver. We use an *emergency based scheduling* algorithm which picks the data with the largest emergency value.

6.4 Simulation environment setup

We use TSIS (Traffic Software Integrated System) 5.1 [17] to generate a traffic trace for our simulation. TSIS is an integrated development environment that enables users to conduct vehicle traffic operation analysis. We use the built-in tools TRAFED, a GUI-based network and simulation input editor to draw the road network topology. In this experiment, the road network is a segment of freeway I-75 in northwest Atlanta. The road network is shown in Figure 41. This highway network supports two way traffic, each direction has 4 to 5 lanes. The total length of the highway is around 9 miles, with 6 entrance/exits. We then use CORSIM, a microscopic transportation simulator to generate vehicle traffic on the road network created by TRAFED [17].

As shown in Figure 41, the destination region is the segment of the highway $\frac{1}{4}$ mile south of exit 250. This region is generally a significant congestion point during Atlanta's rush hour. The size of the destination area is $\frac{1}{4} \times \frac{1}{4}$ mile. The deadline is $\frac{3}{4}$ mile north of exit 250, this point is usually where the congestion starts.

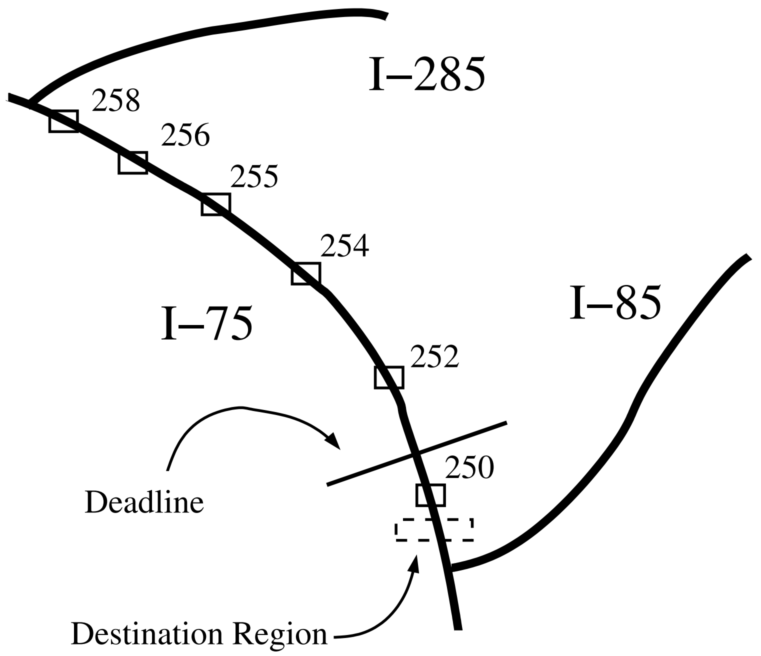


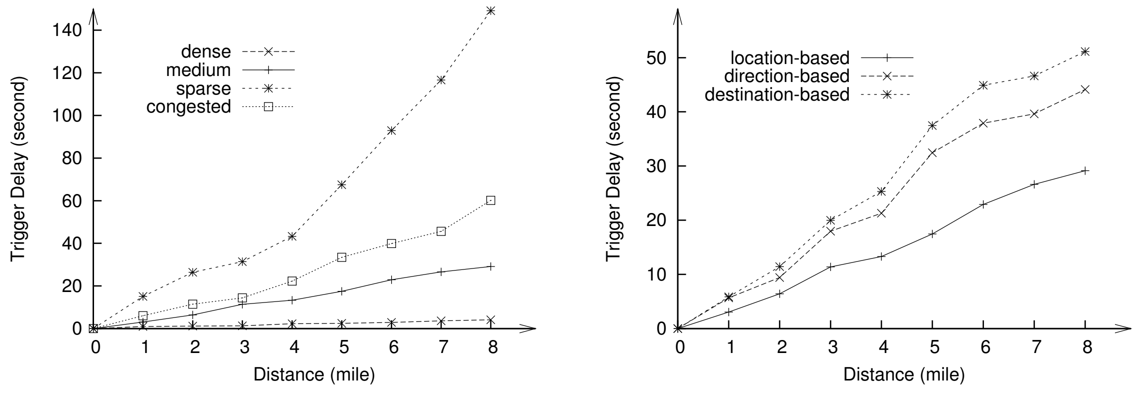
Figure 41: Highway structure for simulation experiments

Four kinds of vehicle traffic conditions are used in our simulation: *dense, medium, sparse*, and *congested*. Dense traffic means that average vehicle-vehicle distance is much smaller than a vehicle's radio range, and the vehicles can still maintain normal speed. Medium traffic means that average vehicle-vehicle distance is similar to the vehicle's radio range, while sparse traffic refers to the condition where vehicle-vehicle distance is larger than the vehicle's radio range. Different vehicles, such as trucks, cars, have different average speed, and different speed distributions. Vehicles drive normally ($40mph$ to $60mph$) under dense, medium and sparse traffic conditions. Under congested traffic, such as rush hour traffic in metro areas, vehicles drive significantly slower ($10mph$ to $15mph$) than normal speed. In these simulation experiments, the default penetration ratio is set at 30%.

6.5 Performance evaluation for the single receiver scenario

6.5.1 Evaluation of trigger message forwarding

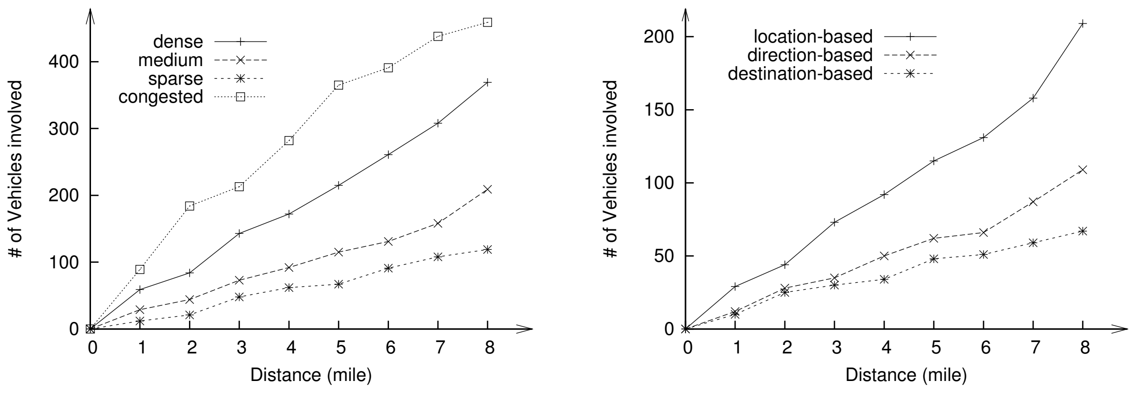
We first study the performance of the video source trigger sub-system. We are interested in the *video trigger delay*, and the *video trigger overhead*. Video trigger delay is the delay between the time when the receiver first sends out the trigger message and the time when the first trigger message reaches the destination region. Trigger overhead is evaluated by the number of vehicles that are involved in the trigger message forwarding process.



(a) Delay vs. traffic (b) Delay vs. scheme

Figure 42: Video trigger delay

• **Video trigger delay:** This experiment evaluates the performance of the video trigger system. The results are shown in Figure 42. In this figure, the distance refers to the distance between the receiver and destination region when the trigger message is generated. Figure 42(a) shows the video trigger delay under different traffic conditions. The trigger delay is very small under dense traffic, since the network remains connected for most of the time. It increases as the density of traffic decreases, since the network is more likely to be partitioned. Note that the trigger delay under congested traffic is much higher than that under dense traffic. One major reason is low market penetration ratio, even under congested traffic, there still exist partitions on the road. What's more, under congested traffic condition, the relative location of vehicles remains unchanged. Trigger messages that are stuck in one partition needs long time to travel to another partition. Figure 42(b) shows the video trigger delay under different forwarding schemes. This experiment is carried under medium traffic density. Location-based flooding performs the best, since every possible path is explored. In the direction-based and destination-based schemes, trigger messages can be delayed when the upstream wireless link is disconnected. Under this condition, even though there may exist a path in the opposite traffic direction, the trigger message can not be forwarded.



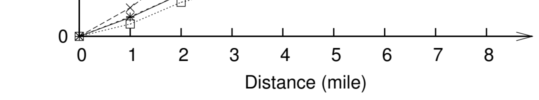
(a) Overhead vs. traffic (b) Overhead vs. scheme

Figure 43: Video trigger overhead

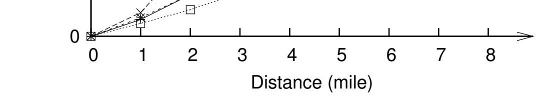
• **Video trigger overhead** The video trigger message overhead is shown in Figure 43. Figure 43(a) shows the trigger overhead under different traffic conditions. As the traffic density increases, more and more vehicles are involved in the trigger message forwarding process. Figure 43(b) shows the video trigger overhead under different forwarding schemes. Location-based flooding has the largest overhead, while destination-based scheme has the smallest overhead. This is because the location-based flooding scheme involves every vehicle in the TMFZone, while direction-based and destination-based schemes only select a subset of vehicles.

6.5.2 Evaluation of video transmission

We then study the performance of video data transmission in the V3 architecture. We are interested in the following performance metrics: *viewing offset* and *service delay.* and *data delivery ratio.* Viewing offset is the time between the receiver vehicle's video playback time and the video generation time. Service delay is the delay between the time when a receiver vehicle sends out the trigger message and the time when the receiver vehicle first receives the video data. Data delivery ratio is evaluated by the number of video packets received over the number of packets generated. Note that the delivery ratio here not only includes those packets that never reach the receiver, but also includes those discarded since the deadline has expired.



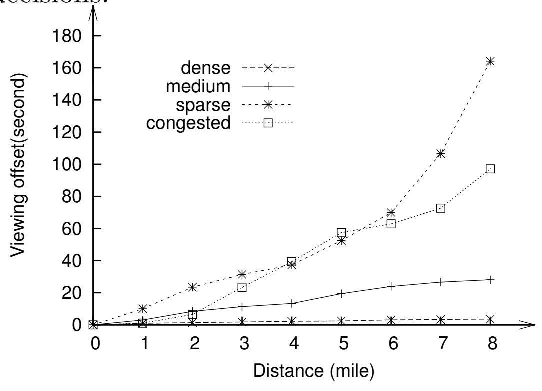
(a) Viewing offset.



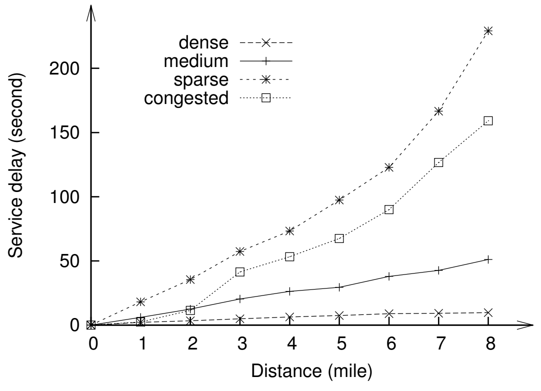
(b)Service delay.

Figure 44: Effect of data forwarding schemes

• **Effect of data forwarding schemes** Figure 44 shows the viewing offset and service delay under different forwarder selection algorithms. In this experiment, we assume medium traffic condition. The optimal selection algorithm, which assumes that each vehicle knows the trajectory of all the other vehicles, selects the vehicle which could deliver data to the receiver fastest as a forwarder. The random algorithm performs the worst, as a vehicle obtains more knowledge of each vehicle's driving behavior, it makes better forwarder selection decisions.



(a) Viewing offset



(b) Service Delay

Figure 45: Effect of traffic conditions

• **Effect of traffic conditions** Figure 45 shows viewing offset and service delay under different traffic conditions. Similar to video trigger message forwarding, both viewing offset and service delay perform the best under dense traffic. While sparser traffic results in much longer delay. Note that distance in Figure 45(a) is the distance between the receiver and the destination region when the video data is generated, while the distance in Figure 45(b) is the distance between the receiver and the destination region when the video request is generated. The service delay is a few seconds under dense traffic conditions, and is in the 10's of seconds under medium traffic conditions. Even for congested traffic, the delay is around 2 minutes. Thus, video data will reach the receiver long before the deadline. Also, we have assumed a comparatively low penetration ratio (30%) in these experiments. If the penetration ratio increases, both the viewing offset and the service delay will decrease significantly. We show the effect of penetration ratio in the next experiment.

• **Effect of market penetration ratio** Low market penetration ratio causes network partitions even under dense traffic conditions. We study the effect of penetration ratio on client viewing offset in this section. In this experiment, the average vehicle speed is 60mph, the starting distance between the video receiver and the destination region is 8 miles. An

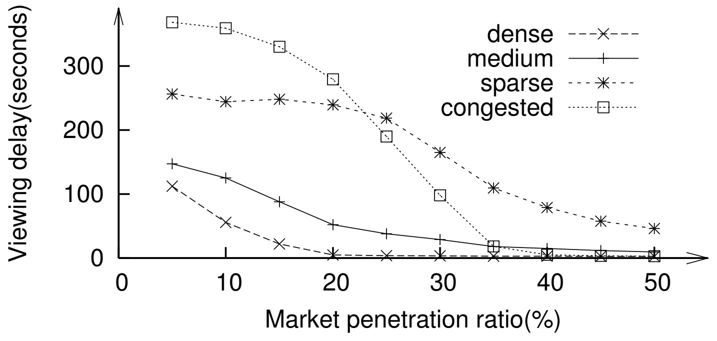
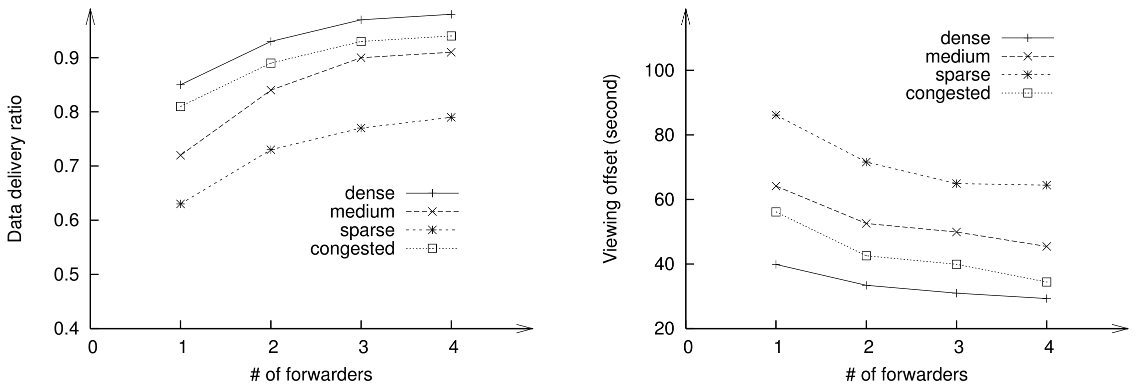


Figure 46: Effect of market penetration ratio

optimal forwarding scheme is used to transmit the video data. Figure 46 shows the simulation results. As the penetration ratio increases, the viewing offset decreases significantly. Since under this condition, the number of partitions decreases as more and more vehicles are equipped with required capability. It is worth noticing that under congested traffic, the viewing offset drops from 360 seconds to below 10 seconds as the penetration ratio increases from 5% to 40%.



(a)Data delivery ratio

(b) Viewing offset

Figure 47: Effect of number of forwarders

• **Effect of number of forwarders** In this experiment, we evaluate the effect of number of forwarders on the data delivery ratio, and client viewing offset. The destination based forwarder selection algorithm is used in this experiment. The simulation results are shown in Figure 47. Figure 47(a) shows the effects of the number of forwarders on data delivery ratio. As the number of forwarders increases, more video data is received before the deadline. One reason is that as more forwarders are selected, the damage of a forwarder leaving the highway is reduced. We also find out that, when the number of forwarders is more than 3, adding more forwarders does not improve the data delivery ratio much. Figure 47(b) shows the viewing offset under different traffic conditions. Selecting more forwarders increases the possibility to forward data to next hop vehicles early. We also find out that as the traffic density increases, the effect of adding more forwarder also increases.

6.6 Performance evaluation for multiple receivers scenario

In this section, we evaluate the performance of V3 architecture under the multiple receiver scenario.

6.6.1 Evaluation of trigger message forwarding

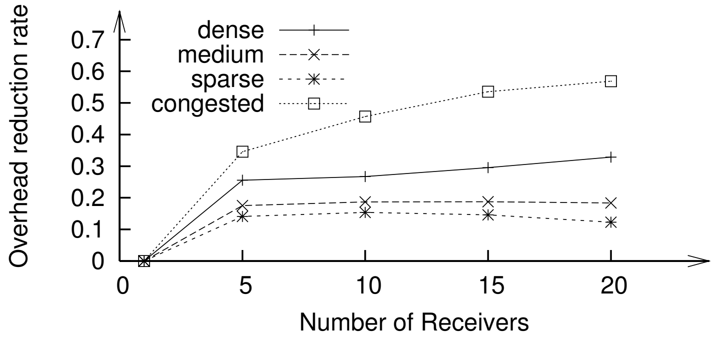
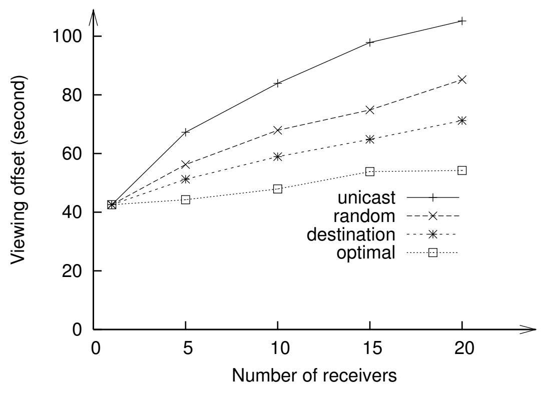


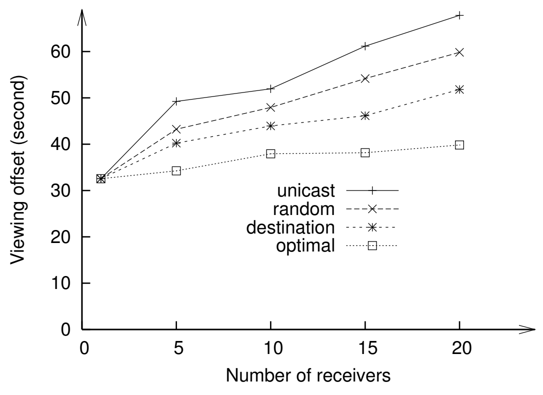
Figure 48: Effect of trigger message integration scheme

We first evaluate the impact of multiple receivers on the video trigger sub-system. In this experiment, we are interested in video trigger overhead. Note that a vehicle can be counted multiple times as it forwards trigger messages from different receivers. This experiment evaluates the efficiency of the proposed trigger message integration scheme under different traffic conditions. The receivers at random locations of the road generate video requests in a short period of time, these requests conform to a uniform distribution. The experimental results are shown in Figure 48.

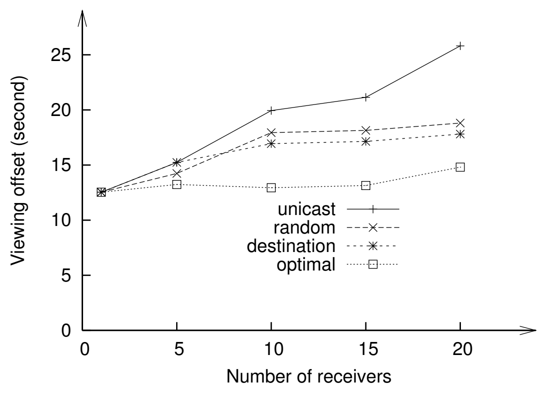
Under sparse traffic conditions, the reduction of video trigger overhead is not very significant. This is because that the chance of a vehicle buffering multiple trigger messages is not very high. As the traffic gets heavier, trigger messages tend to be buffered in the same vehicle more often. It is worth noticing that under congested traffic conditions, the trigger integration scheme can significantly reduce the video trigger overhead. Since in this situation, the partitions are less dynamic, and trigger messages inside any partition can thus be integrated more efficiently.



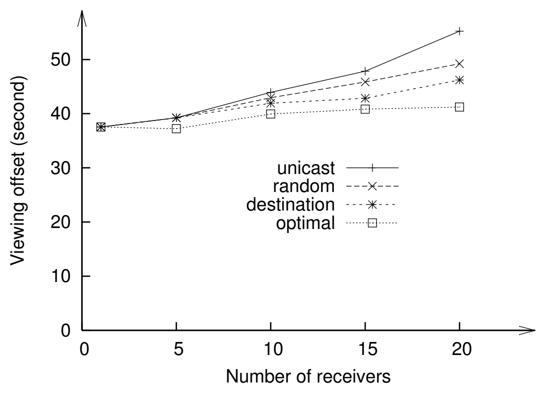
(a) Sparse traffic



(b) Medium traffic



(c) Dense traffic



(d) Congested traffic

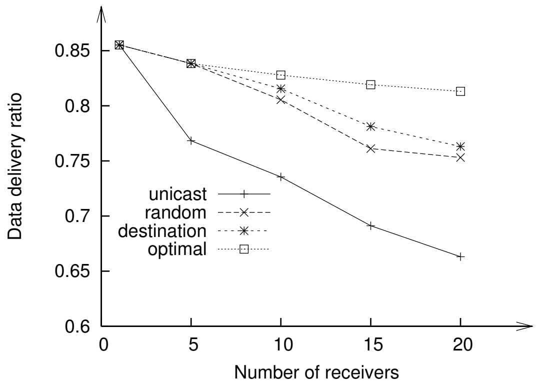
Figure 49: Viewing offset comparison

6.6.2 Evaluation of multicast routing framework

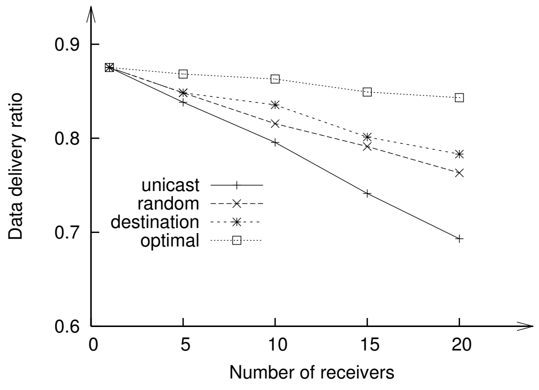
In this section, we study the performance of the multicast routing framework in the V3 architecture. We are interested in the following performance metrics: viewing offset, and data delivery ratio. We evaluate the following schemes: unicast, random selection, destination-based selection, and optimal selection.

• **Viewing offset:** In this experiment, each receiver records its viewing offset during the video streaming process. The average viewing offset among all the receivers is compared under different traffic conditions. The simulation results for different forwarding schemes are shown in Figure 49. The unicast scheme needs to send out duplicated packets in sequence, thus it has the longest viewing offset. This effect is more significant under sparse traffic, since under sparse traffic, the data forwarding capacity of the V2V network is small, making it more difficult to deliver duplicated video data. As the traffic on the road gets heavier, the capacity of the network is also increased. The average receiver viewing offset is decreased correspondingly. Also, as each vehicle obtains more information about other vehicles and road traffic conditions, it can make better decisions that lead to smaller viewing offsets.

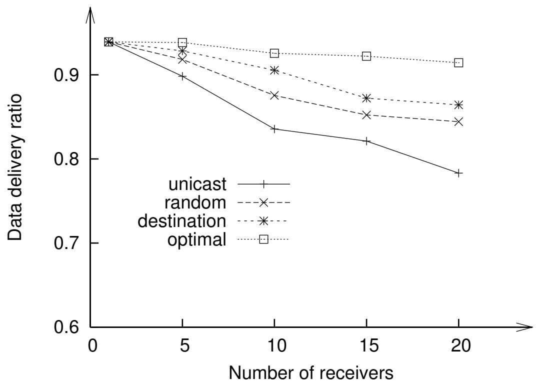
• **Data delivery ratio** The next experiment evaluates the data delivery ratio under different traffic conditions. The results are shown in Figure 50. Note that since video data arrives after the deadline is discarded, video data generated right before the deadline has a small chance of being delivered on time. As shown in Figure 50, the unicast scheme has the lowest data delivery ratio, since this scheme causes the longest delays. As a result, a high percentage of video data will be discarded due to missing the deadline. The destination-based selection scheme has a higher data delivery ratio than the random selection scheme. Since the destination-based selection algorithm selects a better set of forwarders which has a higher probability of delivering data to the corresponding partitions. The optimal selection algorithm performs the best. Since it not only can deliver the packets to the receiver partitions, but also can deliver the data the fastest, reducing the chance of data loss caused by deadline expiration. Also, the data delivery ratio increases as the road traffic gets heavier.



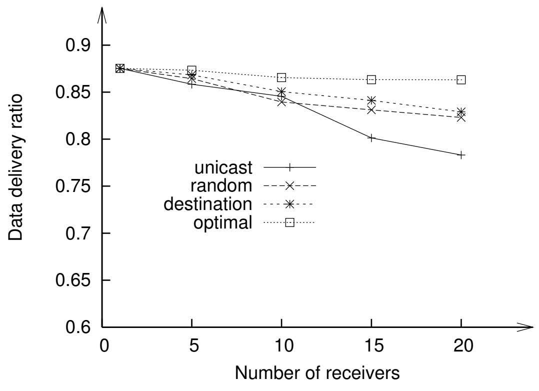
(a) Sparse traffic



(b) Medium traffic



(c) Dense traffic



(d) Congested traffic

Figure 50: Data delivery ratio comparison

6.6.3 Impact of multiple destination regions

In this section, we evaluate the performance of the proposed video data scheduling algorithms. In these experiments, we have multiple destination regions that are very close to the first destination region. In this way, video data can be forwarded along similar routes to the receivers. Thus, an intermediate vehicle has a higher probability to buffer video from multiple destination regions at the same time, increasing the impact of scheduling algorithms.

In this experiment, the optimal routing algorithm is used, and the traffic condition

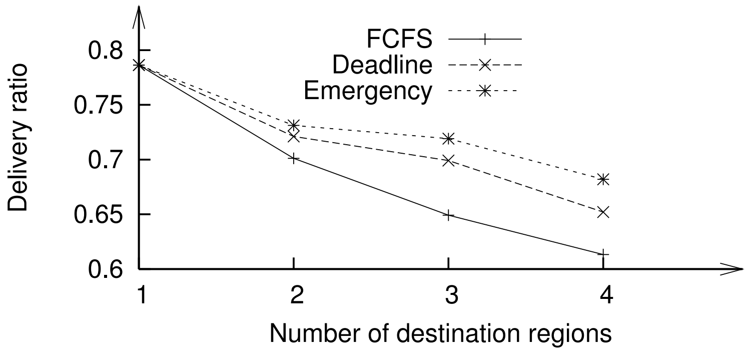


Figure 51: Data delivery ratio

is medium. We also assume that there are 20 receivers requesting for video data at the same time. The requests are uniformly distributed to different destination regions. The simulation result is shown in Figure 51. As the number of destination regions increases, the data delivery ratio decreases. This is because, even under the same video request intensity, video data from different destination regions have to be treated separately, causing longer viewing offset. Also, as the intermediate vehicle obtains more information (such as deadline, road and traffic conditions), it can make better scheduling decisions, thus providing better data delivery ratio. We have observed the similar patterns for the data delivery ratio under other traffic conditions.

6.7 Summary

In this paper, we propose our design of V3 – an architecture to enable a live video streaming service to driving vehicles. This architecture is composed of two parts: a video source triggering sub-system, and a video data transfer sub-system. In the video triggering scheme, each vehicle in the TMFZone, upon receiving the trigger message, stores and forwards this message to other vehicles within its radio range. To provide continuous video streaming, we also propose a combined approach of pre-trigger and post-trigger to trigger a set of vehicles to become video sources. V2V networks may be partitioned at all times. We use a store-carry-and-forward scheme to facilitate data transmission in such a network environment. Several forwarder selection algorithms are proposed to balance the video transmission delay and bandwidth overhead. This design handles video streaming to a single receiver, as well as multiple receivers. To support video streaming from multiple destination regions, we design a set of scheduling schemes that use increasing amount of knowledge. The simulation experiments verify that the flooding overhead is tolerable for lightweight triggering messages. They also demonstrate the feasibility of using the proposed architecture to support V2V video streaming with acceptable performance.

We can draw the following conclusions based on our experience with V3: 1) A V2V approach can be effective in disseminating real time video streams. This architecture is flexible and easy to set up, it is especially useful where there is no infrastructure support; 2) The performance of video transmission gets better as more vehicles participate in V3. As we can see, the market penetration ratio is critical to the video trigger delay, and service delay performance. This is especially true under congested traffic conditions;3) A combined architecture of infrastructure-based approach and V2V approach can be more efficient under certain scenarios. For example, it is possible to deploy video cameras, and access points along major highways that lead to metro areas. Since these highways usually have very high traffic volumes, and require constant attention. Setting up such infrastructure can provide better performance.

CHAPTER VII

CONCLUSIONS AND FUTURE WORK

Deploy commercial quality video streaming service to a large scale of receivers has alway been a challenging task. In this thesis, we focus on providing scalable and resilient video streaming applications to end hosts through various network contexts. This research seeks to investigate the major challenges of streaming video in evolving and challenging network environments. Our objective is to design appropriate video streaming techniques according to different network structures, and to develop schemes to compare and evaluate these video streaming mechanisms. Specifically, this research focuses on the following issues: i) our research address the scalability issue of VoD service over the Internet; ii) our research also tries to design a resilient scheme that supports continuous video streaming in unreliable overlay networks; iii) our research work also explores the feasibility of streaming live video in partitioned mobile ad hoc network environments. In this chapter, we summarize the main contents of the thesis with highlights on major contributions, and point our several future research directions.

7.1 Summary of Contributions

- **Server selection in replicated video on demand systems:** First we study the problem of scalable VoD services using server replication and multicasting. In particular, our research focuses on the server selection schemes in replicated batching VoD systems. Our design objective is to select servers for clients in a way that minimizes a client's user access latency. We design a set of server selection algorithms based on various amount of knowledge used. We find out that in order to direct a client's request to a server which can serve its request earliest, both the server state information and channel allocation schemes information have to be considered. We also developed an experimental set up to run the simulations. Experimental results show that the server selection algorithms that are proposed can be used to efficiently distribute client requests to the "best" server. This techniques can be coupled with replicated VoD system to improve the system capacity and to improve the client perceived performance.

- **Continuous live video streaming in overlay networks using time shifting and video patching:** We then study the problem of supporting live video streaming applications to a large group of receivers using application layer multicast. The major challenge for such application is to maintain continuous video streaming service under unreliable forwarding infrastructure. Previous solutions either can't achieve continuous video streaming, or can not maintain consistent video quality. In this research, our objective is to design a scheme that can maintain continuous video streaming service while maintain consistent video quality. We present a solution which combines a time-shifting video server and video patching technique. With this solution, a client in the overlay network can recover the lost video data by rejoining the time-shifted video stream, and at the same time, maintain the video playback progress by connecting to the original video stream. The simulation experiments concludes that, this solution can provide continuous video streaming in unreliable overlay networks with a standard single description encoded video stream.

- **A Cooperative patching architecture in overlay networks:** The server-based scheme that we proposed has two major limitations: i) potential performance degradation due to limited bandwidth resource at the video server; ii) complex server design, since it requires a specially designed video server, the server not only has to broadcast the live video, but also has to serve time-shifted stream on-demand. We propose a design of a *cooperative patching* architecture, which recovers lost video packets and achieves continuous video playback at client side. This architecture relieves server load by completely shifting video patching responsibility to the client side. This design will especially be beneficial for legacy video systems, since it does not require modification of the video server.

- **V3: a vehicle to vehicle video streaming architecture:** Streaming live video content to driving vehicles can provide a very attractive service. Many applications can benefit from such kind of service, for example, driver assistance and safety applications, business and entertainment applications, etc. In this research, we focus on providing video streaming service to driving vehicles in a vehicle-to-vehicle approach. The major challenge for this work is that the V2V network, a special mobile ad hoc network in which mobile nodes are installed inside a vehicle, may be partitioned all the time. We propose V3, a vehicle to vehicle video streaming architecture, to enable live video streaming in V2V networks. Our design is composed of two phases: video source triggering, and video data forwarding. We use a directed-flooding scheme to send trigger message toward the video source, and use a combined approach of pre-trigger and post-trigger to continuously trigger the vehicles into video source. We also use a store-carry-and-forward approach to send data back at the receiver. We first study a preliminary case where a single receiver is interested in the video information about a single destination region. We then extend our design to handle more general case where multiple receivers are interested in different destination regions. We propose a trigger message integration scheme to reduce the video trigger overhead. We also design a multicasting framework which sends data to a group of receivers. To evaluate our design, we use the TSIS software which is widely used in transportation research community, to generate real transportation traffic on highway network. We then run simulation experiments based on the generated traffic trace. Experimental results show the feasibility of streaming live video in V2V networks.

7.2 Future Work

This thesis addresses research issues in video streaming applications in evolving network environments. Some of the future efforts that can improve the system performance, or make our design fits better with the real world application include:

- **Replicated VoD Systems:** server replication is an efficient way to support scalable video on demand service. Techniques for server selection in a replicated VoD system play an important role in fully utilizing the benefits of replicated resources. Our work explores the use of a range of selection techniques that use a varying amount of knowledge about the servers and their location. It is desirable to consider the following aspects of future works: i) evaluation of the effect of out of date information on the performance of the selection techniques; ii) The use of partially replicated VoD servers where not all the videos are available at all the servers; iii) the use of heterogeneous replicated servers in which different servers may use different batching techniques. Since in the real world, it is highly possible that different VoD servers adopt different scheduling approaches.

- **Continuous video streaming in overlay networks:** supporting live video streaming using application layer multicast has a widely adopted solution in recent years. Due to the unreliability nature of the data forwarding structure, how to achieve resilient video streaming is vital to the overall video streaming service quality. Our work uses a reactive solution, in which end hosts rejoin the application layer multicast group when they detect a service disconnection. In future work, we try to investigate the proactive solution for resilient video streaming. For example, adding some random links in the application layer multicast tree to form a more robust mesh topology to transmit the video data. We also think of using FEC encoded video and packet level retransmission scheme to improve the video playback quality. Other work includes reducing the user access latency by joining two streams at the beginning, and starts the video playback immediately.

- **Video streaming in V2V networks:** video streaming in V2V network is a very challenging task. Our design of V3 is just the first step of a practical system. Many issues are still remain unsolved: i) how to coordinate the video capture procedure: sometimes there are more than one vehicle within the destination region, how to coordinate these vehicles so that there is no overlap in captured video; ii) incorporate application layer schemes to improve the video quality. For example, dynamic DFZone management, which starts the data forwarding process before the trigger message arrives at the destination region, can significantly reduce the viewing delay; iii) mixed video forwarding structure: in many cases, infrastructure-based can provide much better performance. It is worth studying a mixed architecture in which an infrastructure-based approach and a vehicle-to-vehicle approach work together to provide live video streaming service.

REFERENCES

[1] http://www.akamai.com, Oct., 2001.

[2] C.C. Aggarwal, J.L.Wolf, and P.S. Yu. On optimal batching policies for video-on-demand storage servers. In *Proceedings of the IEEE International Conference on Multimedia Systems*, June 1996.

[3] K. Almeroth and M.Ammar. On the Use of Multicast Delivery to Provide a Scalable and Interactive Video-on-Demand Service. In *Journal on Selected Areas of Communication*, August 1996.

[4] J. Apostolopoulos. Reliable Video Communication over Lossy Packet Networks using Multiple State Encoding and Path Diversity . In *Proceedings of VCIP 2001.*

[5] A. Bachir, A. Benslimane. Towards supporting GPS-unequipped vehicles in inter-vehicle geocast In *proceedings of VTC 2003-Spring.*

[6] S. Banerjee, B. Bhattacharjee, C. Kommareddy Scalable Application Layer Multicast In *Proceedings of ACM SIGCOMM*, August 2001.

[7] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, S.Khuller Construction of an Efficient Overlay Multicast Infrastructure for Real-time Applications In *Proceedings of Infocom*, 2003

[8] K. Calvert, M. Doar, and E. Zegura. Modeling Internet topology. *IEEE Communications Magazine*, June 1997.

[9] R. L. Carter and M. E. Crovella. Server selection using dynamic path characterization in wide-area networks. In *Proceedings of IEEE INFOCOM*, 1997.

[10] S.W. Carter and D.D.E. Long. Improving Video-on-Demand Server Efficiency Through Stream Tapping. In *Proc. of 6th Int'l Conf. on Computer Communications and Networks*, September 1997.

[11] M. Castro, P. Druschel, A.M. Kermarrec, A. Nandi, A. Rowstron, A. Singh Split-Stream: High-bandwidth content distribution in a cooperative environment In *Proceedings of IPTPS 2003.*

[12] Z.Chen, H.Kung, and D.Vlah. Ad hoc relay wireless networks over moving vehicles on highways. In *Proceedings of MobiHoc 2001.*

[13] Y.H. Chu, S.G. Rao and H. Zhang A Case For End System Multicast In *Proceedings of ACM SIGMETRICS*, June 2000,

[14] Y.H. Chu, S.G. Rao and H. Zhang Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture In *Proceedings of ACM SIGCOMM*, August 2001.

[15] B. Cohen. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, Jun 2003.

[16] C. Cordeiro, H. Gossain and D.P. Agrawal Multicast over Wireless Mobile Ad Hoc Networks: Present and Future Directions. In *IEEE Network*, Vol.17, No.1, January 2003.

[17] http://www.fhwa-tsis.com/, Dec., 2004

[18] Y. Cui, K. Nahrstedt. Layered Peer-to-Peer Streaming. In *Proceedings of NOSSDAV 2003.*

[19] A.Dan, D.Sitaram, and P.Shahabuddin. Scheduling policies for an on-demand video server with batching. In *Proc. of 2nd ACM Multimedia Conference*, Oct. 1994.

[20] H.D. Dykeman, M.H. Ammar, J.W. Wong. Scheduling algorithms for Videotex systems under Broadcast Delivery. In *Proceedings of ICC'86.*

[21] D. Eager and M. Vernon Dynamic skyscraper broadcasts for video-on-demand. In *Proc. 4th Intl. Workshop on Multimedia Information Systems*, September 1998.

[22] D. Eager, M.Vernon, and J. Zahorjan, Optimal and Efficient Merging Schedules for Video-on-Demand Servers, In *Proc. 7th ACM Intl. Multimedia Conf. (ACM MULTIMEDIA 99)*, 1999.

[23] D.L. Eager, M.K. Vernon, and J. Zahorjan. Minimizing Bandwidth Requirements for On-Demand Data Delivery, In *IEEE Trans. on Knowledge and Data Engineering* Sept./Oct. 2001.

[24] K. Fall A Delay-Tolerant Network Architecture for Challenged Internets. In *proceedings SIGCOMM 2003.*

[25] S. Jain, K. Fall, R. Patra. Routing in a Delay Tolerant Networking. To appear in *proceedings SIGCOMM 2004.*

[26] Z. Fei, S. Bhattacharjee, E. W. Zegura, and M. H. Ammar. A novel server selection technique for improving the response time of a replicated service. In *Proceedings of IEEE INFOCOM*, 1998.

[27] Z. Fei, M. Ammar, E. Zegura. Optimal Allocation of Clients to Replicated Multicast Servers. In *Proceedings of the 1999 IEEE International Conference on Network Protocols*, November 1999.

[28] Z. Fei, M. Ammar, E. Zegura. Efficient Server Replication and Client Re-Direction for Multicast Services. In *Proceedings of SPIE/ITCOM Conference on Scalability and Traffic Control in IP Networks*, August 2001.

[29] Z. Fei, M. Ammar, E. Zegura. Multicast Server Selection: Problems, Complexity and Solutions. To appear in *IEEE Journal on Selected Areas in Communication.*

[30] P. Francis. Yoid: Extending the multicast Internet architecture. White paper http://www.aciri.org/yoid/, 1999.

[31] D.L. Gall. MPEG: a video compression standard for multimedia applications. In *Communications of the ACM*, Volume 34, Issue 4, April 1991.

[32] L. Gao, D. Towsley. Supplying Instantaneous Video-on-Demand Services Using Controlled Multicast In *Proc. of IEEE Multimedia Computing Systems*, June 1999.

[33] L. Gao, D. Towsley. Threshold-Based Multicast for Continuous Media Delivery In *IEEE Transactions on Multimedia*, Dec. 2001

[34] L. Golubchik, J.C.S. Lui, and R. Muntz. Reducing I/O demand in Video-On-Demand storage servers. In *Proc. 1995 ACM Sigmetrics Join Int'l Conf. on Measurement and Modeling of Computer Systems*, May 1995.

[35] V.K. Goyal, J. Kocevic, R. Arean, and M. Vetterli. Multiple Description Transform Coding of Images In *Proceedings of ICIP*, 1998

[36] M. Guo, M.H. Ammar, E.W. Zegura. Selecting among Replicated Batching Video On Demand Servers. In *proceedings of NOSSDAV 2002*.

[37] M. Guo, M.H. Ammar Scalable live video streaming to cooperative clients using time shifting and video patching. *Technical Report GIT-CC-03-40*, College of Computing, Georgia Tech, 2003.

[38] M. Guo, M.H. Ammar. Scalable live video streaming to cooperative clients using time shifting and video patching. In *proceedings of INFOCOM 2004*.

[39] M. Guo, M.H. Ammar, E.W. Zegura. Cooperative Patching: A client based P2P Architecture for supporting continuous live video streaming. In *proceedings of ICCCN 2004*.

[40] M. Guo, M.H. Ammar, E.W. Zegura. Supporting continuous live video streaming in cooperative overlay networks. Submitted to *Special Issue of Computer Networks on overlay distribution structures and their applications*

[41] M. Guo, M.H. Ammar, E.W. Zegura. V3: a vehicle-to-vehicle video streaming architecture. In IEEE PERCOM 2005.

[42] Y. Guo, K. Suh, J. Kurose, D. Towsley. P2Cast: peer-to-peer patching scheme for vod service. In *Proceedings of the 12th World Wide Web conference*, 2003.

[43] Y. He, F. Wu, S. Li, Y. Zhong, and S. Yang. H.26L-based fine granularity scalable video coding. In Proc. ISCAS-2002 IEEE International Symposium on Circuits and Systems,, 2002.

[44] K.A. Hua, S. Sheu. Skyscraper broadcasting: a new broadcasting scheme for metropolitan video-on-demand systems. In *Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, 1997.

[45] K.A. Hua, Y. Cai, S. Sheu. Patching: a multicast technique for true video-on-demand services . *Proc. of the 6th ACM international conference on Multimedia*, 1998.

[46] Duc A. Tran, Kien Hua, Tai Do ZIGZAG: An Efficient Peer-to-Peer Scheme for Media Streaming In *Proceedings of InfoCom 2003*.

[47] A. Iwata, C.-C. Chiang, G. Pei, M. Gerla, and T.-W. Chen, Scalable Routing Strategies for Ad Hoc Wireless Networks. In *IEEE Journal on Selected Areas in Communications, Special Issue on Ad-Hoc Networks, Aug. 1999.*

[48] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O'Toole. Overcast: Reliable multicasting with an overlay network. In *Proc. of OSDI 2000.*

[49] Shudong Jin and Azer Bestavros. Cache-and-Relay Streaming Media Delivery for Asynchronous Clients. In *Proceedings of NGC 2002*

[50] David B. Johnson, Davis A. Maltz. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks. In *Proceedings of SIGCOMM,* 1996.

[51] S. Kandula, J.-K. Lee, and J. C. Hou. LARK: A light-weight, resilient application-level multicast protocol. In Proc. of IEEE CCW'03.

[52] B. Kim, Z. Xiong, and W.A. Pearlman. Low bit-rate scalable video coding with 3-D set partitioning in hierarchical trees (3-D SPIHT). In *IEEE Trans. on Circuits and Systems for Video Technology, vol. 10,* Dec. 2000.

[53] M. S. Kim, S. S. Lam, D.Y. Lee Optimal Distribution Tree for Internet Streaming Media. *Technical Report* , U.T. Austin, 2002

[54] Y.B. Ko, N.H. Vaidya. Geocasting in Mobile Ad Hoc Networks: Location-Based Multicast Algorithms. In *Proceedings of IEEE WMCSA 1999.*

[55] Y.B Ko, N.H. Vaidya. GeoTORA: a protocol for geocasting in mobile ad hoc networks. In *Proceedings of ICNP 1999.*

[56] Y.B. Ko, N.H. Vaidya. Flooding-based geocasting protocols for mobile ad hoc networks. In *Mobile Networks and Applications*, Volume 7, Issue 6, December 2002.

[57] W. Li Overview of fine granularity scalability in MPEG-4 video standard In IEEE Trans. on Circuits and Systems for Video Technology, vol. 11, March 2001.

[58] J. C. Lin and S. Paul, RMTP: A Reliable Multicast Transport Protocol. IEEE INFOCOM 96, March 1996.

[59] Li, X., Ammar, M. H., Paul, S Video Multicast over the Internet. In *IEEE Network Magazine*, Vol. 13. No. 2, March/April 1999, pp46-60.

[60] [13] M.G. Luby, M. Mitzenmacher, M.A. Shokrollahi, D.A. Spielman, and V. Stemann. Practical lossresilient codes, In *Proceedings of Annual ACM Symposium on the Theory of Computing, 1997.*

[61] C. Maihofer A survey of Geocast routing protocols In *IEEE Communications Magazin*, Volume 6, Number 2, 2004.

[62] S.W. Man, S.N. Lin, S.S. Panwar, Y. Wang, and E. Celebi. Video Transport Over Ad Hoc Networks: Multistream Coding With Multipath Transport. In *JSAC Vol. 21, No. 10.* 2003.

[63] V. N. Padmanabhan, H. J. Wang, P. A. Chou, K. Sripanidkulchai. Distributing Streaming Media Content Using Cooperative Networking. In *Proceedings of NOSSDAV 2002.*

[64] V. N. Padmanabhan, H. J. Wang, P. A. Chou. Resilient Peer-to-Peer Streaming In Proceedings of ICNP, November, 2003.

[65] VD Park and MS Corson A highly adaptive distributed routing algorithm for mobile wireless networks In *Proceedings of INFOCOM'97, Apr. 1997.*

[66] C.E. Perkins and P. Bhagwat, Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers. In *Computer Communication Review*, Oct. 1994.

[67] Charles E. Perkins, Elizabeth M. Royer, Samir R. Das. Ad Hoc On-demand Distance Vector Routing. In *2nd IEEE Workshop on mobile computing systems and applications,* 1999.

[68] R. Rejaie, M. Handley, and D. Estrin. Layered quality adaptation for Internet video streaming. In *IEEE J. Select. Areas Commun.*, December 2000.

[69] PALS: Peer to Peer Adaptive Layered Streaming. In *Proceedings of NOSSDAV 2003.*

[70] L. Rizzo, Effective erasure codes for reliable computer communication protocols, In *ACM Computer Communication Review, vol. 27, April 1997.*

[71] J.P.Singh, N.Bambos, B.Srinivasan, D.Clawin. Wireless LAN Performance under Varied Stress Conditions in Vehicular Traffic Scenarios In *IEEE Vehicular Technology Conference*, Fall 2002.

[72] V. Stankovic, R. Hamzaoui, Z. Xiong Live video streaming over packet networks and wireless channels In *Proceedings of PV 2003.*

[73] D. A. Tran, K. A. Hua, T. T. Do Peer-to-Peer Streaming Using A Novel Hierarchical Clustering Approach To appear *at IEEE JSAC Special Issue on Advances in Service Overlay Networks*, 2003

[74] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. Service location protocol. *RFC 2165*, June 1997.

[75] S.Viswanathan , T. Imielinski. Metropolitan area video-on-demand service using pyramid broadcasting. In *Multimedia Systems*, August, 1996.

[76] C.S. Wu, G.K. Ma A real-time transport scheme for wireless multimedia communications. In MONET, Vol.6(6), 2001.

[77] S. Wu and S. Banerjee. Improving the performance of overlay multicast with dynamic adaptation. In *Proceedings of IEEE CCNC, 2004.*

[78] H. Wu, R.M. Fujimoto, G. Riley Analytical models for information propagation in Vehicle-to-Vehicle networks. In *proceedigns of VTC 2004-Fall.*

[79] H. Wu, R. Fujimoto, R. Guensler, M. Hunter. MDDV: mobility-centric data dissemination algorithm for vehicular networks. In *VANET*, 2004.

[80] B. Zhang, S. Jamin, and L. Zhang. Host multicast: A framework for delivering multicast to end users. In *Proceedings of IEEE INFOCOM 2002.*

[81] W. Zhao, M. Ammar Message Ferrying: Proactive Routing in Highly-Partitioned Wireless Ad Hoc Networks. In *Proceedings of the IEEE Workshop on Futrure Trends in Distributed Computing Systems, 2003*.

[82] W. Zhao, M. Ammar, E. Zegura. A Message Ferrying Approach for Data Delivery in Sparse Mobile Ad Hoc Networks. In *Proceedings of ACM Mobihoc 2004*.

[83] E. Zegura, M. Ammar, Z. Fei, and S. Bhattacharjee. Application-layer anycasting: A server selection architecture and use in a replicated web service. *IEEE/ACM Transactions on Networking*, August 2000.

VITA

Meng Guo was born in Xi'an, China. He attended Xi'an Jiaotong University, majoring in electrical engineering & computer science. He then went to Institute of Automation, Chinese Academy of Sciences for graduate study. He obtained a Master of engineering degree in computer engineering. He joined the Ph.D. program in College of Computing, Georgia Institute of Technology, Atlanta, Georgia in 1999. Along the way in the Ph.D. program, he received a Master of Science degree in Computer Science in 2001. He also worked as a summer intern student in AT&T, Avaya Labs, and Microsoft Research Asia Labs.