Search And Delivery Techniques In Peer-to-peer Networks

Tai Do

University of Central Florida

Part of the Computer Sciences Commons, and the Engineering Commons

Find similar works at: https://stars.library.ucf.edu/etd

University of Central Florida Libraries http://library.ucf.edu

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations, 2004-2019 by an authorized administrator of STARS. For more information, please contact STARS@ucf.edu.

STARS Citation

https://stars.library.ucf.edu/etd/3994
SEARCH AND DELIVERY TECHNIQUES IN PEER-TO-PEER NETWORKS

by

Tai T. Do
B.S. University of Oklahoma, 2001
M.S. University of Central Florida, 2005

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the School of Electrical Engineering and Computer Science in the College of Engineering and Computer Science at the University of Central Florida Orlando, Florida

Summer Term
2009

Major Professor: Kien A. Hua
ABSTRACT

The presence of millions of interconnected personal computing devices has given rise to a new class of decentralized networking applications, which are loosely labeled as peer-to-peer (P2P) applications. These P2P applications leverage resources such as processing cycles, storage, content, and network bandwidth available to the user devices, which are also known as peers. A number of current systems - SETI@home, Napster, BitTorrent, and Pastry - are examples of these emerging P2P systems. To fully realize the potential of the peer-to-peer technology, there is a need to define and provide a set of core competencies, serving as the basic services upon which various peer-to-peer applications can be built on. Among these core competencies, this dissertation focuses on two fundamental services, which are search and delivery.

In the first part of the dissertation, delivery techniques to support video-on-demand services in wireline and wireless P2P networks are investigated. Video services are considered due to two reasons. First, video services are the pivotal basis for many other multimedia applications. Second, it is challenging to provide on-demand video services due to asynchronous playback progresses at peers. The proposed techniques enable efficient video sharing between peers with asynchronous playback progresses, and maximize peer bandwidth utilization.

In the second part of the dissertation, the problem of supporting continuous moving range queries in wireless mobile peer-to-peer networks is studied. Continuous moving range queries have a number of applications when a moving object wants to monitor its surrounding environment for a period of time. When a fixed network infrastructure is not available, wireless mobile peer-to-peer networks become a viable option to support the continuous query system. The proposed distributed solution ensures the accuracy of the query results under realistic assumptions, and incurs much less overhead than alternative solutions.
ACKNOWLEDGMENTS

Getting a Ph.D. degree undoubtedly ranks among one of the greatest challenges I have taken to date. Fortunately, I did not have to tackle this challenge alone, but rather with generous support from many people. I take this opportunity to thank those who have supported me in this endeavor.

As my academic advisor, Dr. Kien A. Hua receives the utmost gratitude from me. I am forever indebted to him for his guidance with my research, financial support for my research assistantship, and advices in other matters. I also thank my doctoral committee members, Dr. Chen, Dr. Lang, and Dr. Zou for their helpful comments on my research work and for being very accommodating with my numerous requests. Thanks also to the staff in the School of EECS, especially Jenny Shen, for making seemingly insurmountable administrative tasks become effortless for me.

I thank members of the Data Systems Lab. Duc Tran introduced me to Dr. Hua and UCF, and also let me collaborate on several of his research projects. I enjoyed the time spent discussing research ideas, working on software development projects, and debating various subjects with Alex Aved, Angel Ho, Danny Ho, Danzhou Liu, Fuyu Liu, Fei Xie, Himanshu Pagey, Ning Jiang, Ning Yu, Rui Peng. Outside of the research lab, Duc Nguyen, Hoang Nguyen, Minh Le, Thanh Tran and other friends have kept me entertained by welcoming me to their organized social events, and generally helping me become a wiser individual.

Many thanks also go to my wife Phuong, who has been patient and understanding through this process. Her constant encouragement and unwavering belief in me showed me what persistence really means. I thank my parents for always being my biggest fans, for teaching me the value of learning and education, and for showing me how to live a good life. Thanks also to my two brothers, Thanh and Tuan, and my sister, Van, for taking care of the family matters while I was away and for encouraging words throughout the years. I also thank my parents-in-law for their care and concern to my wife and me in the past few years.
# TABLE OF CONTENTS

**LIST OF FIGURES**  

**LIST OF TABLES**  

**LIST OF ACRONYMS**  

**CHAPTER 1: INTRODUCTION**  

1.1 Motivations .......................................................... 1  
1.2 Main Contributions .................................................. 2  

**CHAPTER 2: VIDEO-ON-DEMAND STREAMING IN WIRELINE PEER-TO-PEER NETWORKS**  

2.1 Introduction .......................................................... 4  
2.2 The Proposed Technique: P2VoD .................................. 6  
2.2.1 Preliminary ......................................................... 6  
2.2.2 Control Protocol .................................................. 8  
2.2.2.1 A Hierarchy of Clusters of Peers .......................... 8  
2.2.2.2 Distributed Directory Service ............................... 9  
2.2.2.3 Data Structures ............................................. 11  
2.2.2.4 Control Messages ....................................... 12  
2.2.3 Selecting Forwarding Peer ..................................... 13  
2.2.4 Join Algorithm .................................................. 15  
2.2.5 Failure Recovery Algorithm .................................. 19  
2.2.5.1 Detecting failures ....................................... 19  
2.2.5.2 Recovering from failures .................................. 20  
2.3 Analysis of P2VoD .................................................. 21
3.2.4 Loading and Playing Video at Peers ............... 53
3.2.5 Caching at Forwarding Buffers .................. 54
3.2.6 Recovery Procedures ............................. 55
   3.2.6.1 Recovery Procedure for Node Mobility .......... 55
   3.2.6.2 Recovery Procedure for Node Departure ......... 56
3.2.7 An Example ..................................... 57
3.2.8 Optimizations and Discussions .................... 58
   3.2.8.1 Discovery of LastRegularTime ................. 58
   3.2.8.2 Ad-hoc Assisted Multicast Routing ............. 59
   3.2.8.3 Multicasting Patching Stream from a Patching Stream Provider 60
   3.2.8.4 Handover with Multiple Base Stations .......... 60
3.3 Performance Study ................................ 61
   3.3.1 PatchPeer with Stationary Nodes ............... 62
   3.3.1.1 Cache Issues ............................... 62
   3.3.1.2 Video Characteristics ....................... 63
   3.3.2 Patching vs. PatchPeer ......................... 65
3.3.3 PatchPeer with Moving Nodes ..................... 67
3.4 Related Work .................................... 69
3.5 Summary ..................................... 73

CHAPTER 4: CONTINUOUS MOVING RANGE QUERIES IN MOBILE PEER-TO-PEER NETWORKS

4.1 Introduction .................................... 75
4.2 The Proposed Technique: ExtRange .................. 77
   4.2.1 System Assumptions ............................ 77
   4.2.2 Models ....................................... 77
   4.2.3 Initial Step .................................. 78
4.2.4 Monitoring Step ......................................................... 80
4.2.5 Refreshing Step ......................................................... 81
4.2.6 Data Structures for ExtRange ................................. 82
  4.2.6.1 Data Structures at Mobile Objects ............... 82
  4.2.6.2 Data Structures at Query Objects ........... 83
4.2.7 Proof of Correctness ............................................. 84
4.2.8 Optimization ......................................................... 84
  4.2.8.1 Extending $SP$ ............................................. 85
  4.2.8.2 Delaying Location Update ....................... 87
  4.2.8.3 Sharing Location Update among Multiple Queries ... 87
  4.2.8.4 Refreshing Multiple Queries .................. 88
4.2.9 Reliability Issues .................................................. 89
4.3 Performance Evaluation of ExtRange .............................. 91
  4.3.1 Result Accuracy ............................................... 92
  4.3.2 Computation Load at Query Objects ............... 93
  4.3.3 Messaging Cost .............................................. 94
  4.3.4 Computation Load at Mobile Objects ............. 96
  4.3.5 Comparing ExtRange against the Baseline Technique ... 96
4.4 Related Works .......................................................... 98
4.5 Summary ............................................................... 99

CHAPTER 5: CONCLUSION

REFERENCES
## LIST OF FIGURES

2.1 A snapshot of the P2VOD system at time 60. ........................................... 7  
2.2 A snapshot of P2VoD at time 9. ................................................................. 23  
2.3 A snapshot of P2Cast at time 9. ................................................................. 23  
2.4 Network topology used in the simulation. ................................................... 28  
2.5 Effect of buffer size on P2VoD. ................................................................. 30  
2.6 Effect of server bandwidth on P2VoD. ....................................................... 31  
2.7 Effect of backbone link bandwidth on P2VoD. ........................................... 31  
2.8 Effect of buffer size on P2Cast. ................................................................. 32  
2.9 Effect of server bandwidth on P2Cast. ....................................................... 32  
2.10 Comparison of P2Cast and P2VoD. ......................................................... 32  
2.11 Number of peers contacted for recovery when $p_f = 0.1$. ....................... 33  
3.1 The Unified Cellular and Ad-hoc Network Architecture ............................ 42  
3.2 Collaboration Diagram of PatchPeer .......................................................... 45  
3.3 Flowchart Diagram for a Requesting Peer in PatchPeer ......................... 46  
3.4 Flowchart Diagram for the Server in PatchPeer ....................................... 46  
3.5 Carrier-Sensing Range of a Node $N_i$ ......................................................... 48  
3.6 Flow Bandwidth Consumption for a pair $(S, D)$ ...................................... 48  
3.7 Peer Locations ......................................................................................... 57  
3.8 Multicast Streams ..................................................................................... 57  
3.9 Ad-hoc Assisted Multicast Routing ............................................................. 59  
3.10 Effect of replacement policy with buffer size of 20 minutes ...................... 63  
3.11 Effect of replacement policy with buffer size of 40 minutes ...................... 64  
3.12 Effect of replacement policy with buffer size of 60 minutes ...................... 64  
3.13 Effect of video characteristics .................................................................. 65
3.14 Effect of number of videos on PatchPeer and Patching .......................... 66
3.15 Effect of request arrival rate on PatchPeer and Patching ........................ 67
3.16 Effect of video length on PatchPeer and Patching ................................. 67
3.17 Effect of mobility on PatchPeer with different buffer sizes ..................... 68
3.18 Effect of mobility on PatchPeer with different average video lengths .......... 68
3.19 Effect of mobility on PatchPeer with different node densities ................... 68

4.1 Illustration of Extended Range .............................................................. 79
4.2 Mobile object is inside the query result. .................................................. 86
4.3 Sharing Location Update among Multiple Queries ..................................... 88
4.4 Two queries issued by the same query object .......................................... 89
4.5 Refresh Schedule for Each Query ............................................................ 89
4.6 Effect of node speed on result accuracy. .................................................. 94
4.7 Effect of node speed on computation load at query objects. ....................... 94
4.8 Effect of extended range on computation load at query objects. .................. 94
4.9 Effect of node speed on messaging cost. .................................................. 95
4.10 Effect of extended range on messaging cost. ........................................... 95
4.11 Effect of extended range on computation load at mobile objects. ............... 95
4.12 Result accuracy for ExtRange and Baseline. .......................................... 97
4.13 Computation load on query objects for ExtRange and Baseline. ............... 97
4.14 Messaging cost for ExtRange and Baseline. .......................................... 98
LIST OF TABLES

2.1 Parameters used for performance evaluation .................................. 28
3.1 Comparing two types of wireless networks .................................... 42
3.2 Parameters used for performance evaluation .................................... 62
4.1 Parameters used for performance evaluation .................................... 92
4.2 Comparison of existing techniques for continuous range queries ...... 99
# LIST OF ACRONYMS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSMA/CA</td>
<td>Carrier Sense Multiple Accesses/Congestion Avoidance</td>
</tr>
<tr>
<td>FIFO</td>
<td>First-in First-out</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>GT-ITM</td>
<td>Georgia Tech Internet Topology Model</td>
</tr>
<tr>
<td>HDR</td>
<td>High Data Rate</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>ITS</td>
<td>Intelligent Transportation System</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>LAR</td>
<td>Location-Aided Routing</td>
</tr>
<tr>
<td>LFU</td>
<td>Least Frequently Used</td>
</tr>
<tr>
<td>LRU</td>
<td>Least Recently Used</td>
</tr>
<tr>
<td>MANET</td>
<td>Mobile Adhoc Network</td>
</tr>
<tr>
<td>MBMS</td>
<td>Multimedia Broadcast Multicast Services</td>
</tr>
<tr>
<td>MP2P</td>
<td>Mobile Peer-to-Peer</td>
</tr>
<tr>
<td>P2P</td>
<td>Peer-to-Peer</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality-of-Service</td>
</tr>
<tr>
<td>UCAN</td>
<td>Unified Cellular and Ad-hoc Network</td>
</tr>
<tr>
<td>V2V</td>
<td>Vehicle-to-Vehicle</td>
</tr>
<tr>
<td>VoD</td>
<td>Video-on-Demand</td>
</tr>
<tr>
<td>WLAN</td>
<td>Wireless Local Area Network</td>
</tr>
<tr>
<td>WWAN</td>
<td>Wireless Wide Area Network</td>
</tr>
</tbody>
</table>
CHAPTER 1: INTRODUCTION

1.1 Motivations

The presence of millions of Internet-connected personal computers has given rise to a new class of decentralized networking applications, which are loosely labeled as peer-to-peer (P2P). These P2P applications take advantages of resources - processing cycles, storage, content, and network bandwidth - available at the edges of the Internet. A number of current systems - SETI@home, Napster, Gnutella, Freenet, and BitTorrent - are examples of these emerging P2P systems [40]. The success and controversy of a number of recent P2P applications have made P2P a popular term, but it is not always clear exactly what qualifies an application as peer-to-peer. In this document, a P2P application has the following characteristics: 1) participants (or peers) in the application are at the edges of the network, i.e. Web servers or personal computers in the wired Internet, and mobile devices in wireless networks, and 2) participants play an active role and have significant autonomy in the application. The above P2P definition is an augmented version of the definition suggested by Clay Shirky in [40]. Active role means participants in a P2P application do not just passively retrieve and display data, but they also perform some or all of the following tasks: sharing local contents, computing a portion of a global computing job, and communicating with other participants. Significant autonomy means that a participant can join and leave the application at will, and it has control over how many resources (content, storage, cycles, and network bandwidth) it is going to contribute to the application.
1.2 Main Contributions

The contributions of this dissertation are a set of search and delivery techniques for P2P applications, as it is envisioned that efficient search and seamless communication and connectivity are the basic building blocks of many future P2P applications.

The first part of the dissertation focuses on video delivery techniques, and proposes two techniques. The first technique is a video-on-demand streaming technique, called *P2VoD*, over wireline P2P networks. P2VoD achieves high peer bandwidth utilization, and makes the system more resilient and robust against peer failures. P2VoD has two major features. First, a caching scheme is implemented at peers to handle asynchronous playback progresses. Second, a control protocol is proposed to provide robust failure recovery and quick join. P2VoD is evaluated analytically and experimentally. The result shows that P2VoD is sound and efficient, and it also outperforms a recently proposed P2P-based VoD streaming system in a number of important performance metrics.

The second technique in the first part of the dissertation addresses the problem of supporting video-on-demand in wireless networks, an important step to achieving the goal of providing video services anywhere anytime. Typically, carrier mobile networks are used to deliver videos wirelessly. Since every video stream comes from the base station, regardless of what bandwidth sharing techniques are being utilized, the media stream system is still limited by the network capacity of the base station. The key to overcome the scalability issue is to exploit resources available at mobile peers in a peer-to-peer setting. It is observed that it is possible to have a carrier mobile network and a mobile peer-to-peer network co-exist in a wireless environment. A feature of such hybrid environment is that the former offers high availability assurance, while the latter presents an opportunistic use of resources available at mobile peers. The proposed technique, called *PatchPeer*, leverages this network characteristic to allow the VoD system scale beyond the bandwidth capacity of the server. Mobile peers in PatchPeer are no longer passive receivers, but also active senders of video streams.
to other mobile peers. The performance study shows that PatchPeer can accept more clients than the current state-of-the-art technique, while maintaining the same Quality-of-Service to clients.

In the second part of the dissertation, a technique is proposed to support continuous moving range queries (CMRQ) in wireless mobile peer-to-peer networks (MP2P). The desirable goals when processing CMRQ in MP2P networks include the followings: 1) the query results have to be accurate at all times, except under communication failures such as network partitioning; 2) the query monitoring task should be distributed; and 3) the overall communication cost should be relatively low. The proposed solution, \textit{ExtRange} (Extended Range), has two key ideas. For a specific query range, ExtRange monitors nodes in a limited extension of the query range. ExtRange also allows mobile nodes to have a Safe-Period, during which mobile nodes need not to report their locations to query nodes. The performance study shows that ExtRange satisfies all three aforementioned goals.

The rest of this document is organized as follows. The P2VoD technique is presented in chapter 2. The PatchPeer technique is described in chapter 3. The ExtRange technique is discussed in chapter 4. Finally, the dissertation draws some conclusions in chapter 5.
CHAPTER 2: VIDEO-ON-DEMAND STREAMING IN WIRELINE PEER-TO-PEER NETWORKS

2.1 Introduction

This chapter addresses the problem of Video-on-Demand (VoD) streaming over the Internet using peer-to-peer (P2P) networks. A P2P-based approach can be used to address several serious problems posed in existing VoD systems including (1) the infeasibility of IP Multicast; (2) network bottleneck at the video server; and (3) the high maintenance and deployment of dedicated overlay routers. Recently, there have been several research projects on live streaming using P2P approach [48, 2, 14, 41]. However, applying these techniques into VoD streaming is not a trivial task due to the following subtle differences between the two types of streaming. First, end-to-end delay is more important to live streaming than VoD streaming. In live streaming, the shorter the end-to-end delay is, the more lively the stream is perceived by the users (defined as liveness in [48]). In VoD streaming, liveness is simply irrelevant because the video stream is already pre-recorded. This fact implies that while a short tree rooted at the video server and spanned over peers is desirable in live streaming, it is not a necessary condition for the case of VoD streaming. Second, a peer joining an ongoing live streaming session is only interested in the stream starting from his/her joining time, while in the VoD streaming case the whole video must be delivered to the new peer. As such, a good VoD system must find an efficient way to provide the initial missing part of the video to the latecomers. Moreover, the correlations between various variables are different for the two types of streaming. For example, a peer will likely stop watching a VoD stream when its QoS degrades, but the peer may not do the same thing for a live stream because he/she doesn’t have an option of watching it again in the future [51]. Therefore, it is expected that if the QoS of the video stream reduces, there will be many more peers leaving the system.
in VoD streaming case than the case of live streaming. This last observation stretches the importance of a robust failure recovery protocol in a VoD streaming system.

Based on the above discussion, the main challenges in designing a P2P-based VoD streaming technique include:

- **Robust failure recovery**: Failures are expected to occur more often in a P2P system. The failure recovery protocol should reconnect the abandoned peers smoothly and quickly, so that there are no loss of frame (*jitter*) and no long delay (*glitch*) when a peer playbacks the video. In addition, the effect of the failure should be localized to ensure that only a small number of peers are affected.

- **Quick join**: The system must allow a new peer to join the system fast.

- **Effective handling of peers’ asynchronous playback progresses**: Because peers request the same video at different times, they have different playback progresses. Streaming video to peers with asynchronous playback progresses is a challenge for existing group communication techniques such as multicast, because multicast is designed to support synchronous group communication.

The proposed technique, called P2VoD, addresses all of the above challenges. P2VoD only assumes IP unicast at the network layer. Each peer in P2VoD has a *FIFO buffer* (First-In First-Out buffer) to cache the most recent content of the video stream it receives to cope with asynchronous peer requests. Specifically, the peer buffer allows early peers to assist the server by serving later peers; peers in P2VoD can forward the video stream to a new peer as long as they have enough out-bound bandwidth and still hold the first data block of the video in the buffer. To address other challenges, the key is to organize peers into a structure that enables fast lookup of peers with appropriate data content in their buffers. The structure is a distributed directory service, created and maintained by a control protocol. The control protocol employs a concept of *generation*. Peers arriving
to the system within a time threshold are grouped into a generation. Each generation has a directory, consisting of one or more peers, to keep track of the current buffer content of each peer in that generation. Directory nodes in different generations coordinate with each other to form a distributed directory service. The distributed directory service allows a peer to find peers with desirable data content in their buffers through probing only a constant number of peers. A constant number of peer probes imply a new peer can join the system quickly and a disrupted peer can recover fast from a failure.

The rest of this chapter is organized as follows. Section 2.2 presents the P2VoD technique. In section 2.3, P2VoD is analyzed and compared against P2Cast. Section 2.4 evaluates P2VoD against P2Cast using simulation-based study. Section 2.5 discusses the related works. Finally, the chapter is concluded in section 2.6.

2.2 The Proposed Technique: P2VoD

This section describes the control protocol of P2VoD in section 2.2.2, and methods to select a forwarding peer for a requesting peer in section 2.2.3. Algorithms to perform a join request and recover disrupted peers from failures are presented in sections 2.2.4 and 2.2.5, respectively.

2.2.1 Preliminary

In P2VoD, a streaming connection is assumed to be constant bit-rate, which equals to the playback rate of the video player. A retrieval block (R-block) is defined as a data unit of the video, which is also equivalent to one unit of playback time. R-blocks of a video are numbered from 1 to the length of the video file according to their temporal position in the video [47]. There is a server, or a group of servers, hosting the video. A client who wants to watch the video is called a peer; the server is treated as a special peer. A peer \( p \) has the following attributes \((\text{pid}, t, b)\), where \( \text{pid}, t \) and \( b \) are the unique identification number (such
as IP address), the arrival time, and the buffer size of \( p \). A peer always wants to watch the video from the beginning. For boost trap purpose, every peer knows the IP address of the server. Furthermore, peers are collaborative; while enforcing collaboration in peer-to-peer networks is an important research area, it is beyond the scope of this chapter.

Figure 2.1 shows a snapshot of two video sessions \( S_1 \) and \( S_2 \) in the P2VoD system at time 60. The starting times of the two sessions are at time units 0 and 39. Each peer in the figure is represented by a circle, where the peer’s id is written inside the circle and the peer’s arrival time is written next to the circle. Session \( S_1 \) has three generations, while session \( S_2 \) has two generations. Peers arriving to the system within a threshold \( T \) are grouped into a generation. In the example, \( T \) is equal to 10. A peer can forward to another peer if the latter arrives to the system no later than \( T \) time units after the former. Each peer has the same buffer size and equals to \( T \) R-blocks.
2.2.2 Control Protocol

The control protocol in P2VoD handles all the peer management tasks. The first two sections discuss in details the two main components of the control protocol, namely the generation concept and the distributed directory service. Next, the data structures and control messages are described. These data structures and control messages are used to ensure the desired functionality of the control protocol.

2.2.2.1 A Hierarchy of Clusters of Peers

When a new peer $p$ arrives to the system, an existing peer $p'$ is eligible to serve $p$ if the first R-block of the requested video is still in the buffer of $p'$. Formally, the eligibility test is defined as follows.

**Definition 2.1** Given two peers $p$ and $p'$. $p'$ is eligible to serve $p$ if and only if $0 \leq (p.t - p'.t) \leq p'.b$, in which case $p'$ is said to pass the eligibility test of $p$.

In other words, there is a time window, dependent on the peer’s buffer size, which defines the usefulness of an existing peer to a newly arriving peer. As a result, two peers arriving to the system at times far apart from each other do not need to be aware of each other’s presence in the system, and any effort to coordinate them is unnecessary. This localized relationship along the time axis between peers suggests that peers can be grouped together for better peer management. Specifically, P2VoD uses a *session* to group all peers, which share a single video stream from the video server. The time localization effect is then applied again inside a session. Peers that join the session within a time threshold are clustered together as a *generation*. All generations of the session, in turn, form a hierarchy. Formally, generation and session are defined as follows.

**Definition 2.2** A generation $G_i$ in a video session $S$ is a collection of peers $p_j$: $G_i = \{ p_j | T \ast (i - 1) \leq (p_j.t - S.sStartT ime) < T \ast i \}, i \geq 0.$
where $T$ is the system-defined threshold, $S.sStartTime$ is the starting time of the session, and $G_0$ contains only the video server.

The threshold $T$ is related to the buffer sizes of peers. For ease of description, it is assumed that every peer uses the same buffer size to cache the incoming video stream, and the buffer size is equal to $T$ R-blocks. Since every peer has the same buffer size, the attribute $b$ in a peer’s attributes is omitted from now on.

For every $i > 0$, $G_{i-1}$ is called the parent of $G_i$, and $G_i$ is called a child of $G_{i-1}$. The generation with the highest index number is called the youngest generation of the video session. A generation with a higher index number is called younger than a generation with a lower index number.

A video session can be defined recursively as follows.

**Definition 2.3** Let $S$ be a video session in P2VoD

$$S = \{G_i | G_i \leftarrow G_{i-1}, G_{i-1} \in S\}, i \geq 0.$$  

$G_i \leftarrow G_{i-1}$ means peers in $G_i$ receive the video stream from either $G_i$ or $G_{i-1}$ of the video session $S$.

### 2.2.2.2 Distributed Directory Service

When a new peer is joining or a disrupted peer is reconnecting to the system, it is important for this requesting peer to identify quickly which existing peer can be the forwarding peer. P2VoD uses a distributed directory service to index existing peers for fast look-up of forwarding peers. Designing a distributed directory service for P2VoD represents two challenges: 1) what is the appropriate tradeoff between conflicting factors such as update cost, look-up cost, and storage cost, and 2) how can the workload be distributed evenly among peers?
The first challenge is an issue for every index service, not just P2VoD’s. This tradeoff means an index service usually provides fast look-up at the expense of higher update and storage costs. The look-up cost is the number of peers a requesting peer has to query when looking for a forwarding peer. The update cost means the non-data messages exchanged in the system, which is discussed in more details in section 2.2.2.4. The storage cost is the amount of memory used to store control data structures as shown in section 2.2.2.3. To address this first challenge, P2VoD focuses on achieving fast look-up.

The second challenge is specific to distributed indices. P2VoD utilizes a concept similar to the cluster head concept in ZIGZAG [48]. Each generation has a directory head, responsible for indexing peers in that generation. The server and the directory heads of all generations in the system form the backbone of the distributed directory service.

Each generation has a directory to index peers in that generation. The directory is made of one or more peers, called directory peers, which are also members of the generation. The index content is replicated in every directory peer. One directory peer is designated as the directory head of the directory. When the very first peer joins the generation, that peer is also made the directory head. Over time, the directory head may recruit other peers to act as directory peers. When a peer \( p \) joins the generation, it reports its attributes to the directory head, namely \((\text{pid}, t)\). The directory head periodically sends the updated index content to other directory peers. The directory head role can be rotated among directory peers.

**Theorem 2.1** Given a peer \( p \) in a generation \( G_i \) (\( i > 0 \)) in a session \( S \), all eligible peers of \( p \) in this video session are either in generation \( G_{i-1} \) or \( G_i \).

**Proof:** Without loss of generality, assume \( S.s\text{StartTime} = 0 \). Since \( p \) is a member of \( G_i \), by definition of a generation, \( T * (i - 1) \leq p.t < T * i \). There are two cases:

- Case 1: If \( i > 1 \), a peer \( p' \) passes the eligibility test of \( p \) if \( 0 \leq (p.t - p'.t) \leq T \); hence \( T * (i - 1) - T \leq p'.t < T * i \) or \( T * (i - 2) \leq p'.t < T * i \). By definition of a generation,
$p'$ is either with generation $G_{i-1}$ or $G_i$.

- Case 2: If $i = 1$, $p$ is in generation $G_1$; hence, $0 \leq p.t < T$. A peer $p'$ passes the eligibility test of $p$ if $0 \leq (p.t - p'.t) \leq T$; or $-T \leq p'.t < T$. Therefore, $p'$ is either the server itself (a sole member of $G_0$), or a peer in $G_1$.

From theorem 2.1, when a peer in generation $G_i$ searches for a forwarding peer, it only needs to look into the directories of $G_i$ and $G_{i-1}$. As a result, for a $G_i$ ($i \geq 1$), its directory only needs to be connected to directories of $G_{i+1}$ and $G_{i-1}$.

### 2.2.2.3 Data Structures

The server has a *Server Session Table (SST)*, and a *Server Directory Table (SDT)*. The SST stores the session information, and has the following table schema: $(sid, sStartTime)$, where $sid$ and $sStartTime$ are the unique identification number and starting time of a session. The SDT stores the directory service information of each session, and has the following table schema $(sid, gid, pid, gStartTime)$, where $sid$ is the session id, $gid$ is the generation id in that session, $pid$ is the id of the directory head of the generation’s directory, and $gStartTime$ is the starting time of the generation.

Each peer has a *Peer Directory Table (PDT)* to keep the information of directory peers of the same generation. PDT has the following schema: $(pid, isDirectoryHead)$, where $pid$ is the id of a directory peer, and the boolean variable $isDirectoryHead$ is set to true if that directory peer is also the directory head.

If a peer is also a directory peer, it has the following additional tables: *Peer Attribute Table (PAT)*, and *Peer Neighbor Table (PNT)*. PAT stores attributes of peers in the same generation, and has the following schema: $(pid, t)$, where $pid$ and $t$ are the id and starting time of a peer. PNT stores the directory information of the neighboring generations, namely the parent and the child of the generation, and has the following schema: $(pid, isParent)$,
where pid is the id of the directory head of the neighboring generation, isParent is 1 if the neighboring generation is the parent and 0 if the neighboring generation is the child of the generation.

Similar to a peer’s attributes, attributes for a generation G and a video session S are also defined for ease of references in later discussions. A generation G has the following attributes (sid, gid, dirHead, gStartTime), where sid is the session id, gid is the generation id, dirHead is the directory head of the generation, and gStartTime is the starting time of the generation. Note that many times the gid of the generation G is indicated in the subscript as G_{gid}. A video session has the following attributes (sid, sStartTime), where sid is the session id, and sStartTime is the starting time of the session.

### 2.2.2.4 Control Messages

Control messages are exchanged to maintain proper functionality of the control protocol. This section describes the exchange of control messages between peers in P2VoD when there is no peer joining or leaving P2VoD. Sections 2.2.4 and 2.2.5 will discuss how the control messages are used when a peer is joining or leaving P2VoD.

In a directory, the role of the directory head can be rotated among directory peers. The purpose of the rotation is to balance the workload among directory peers, since a directory head exchanges more control messages with other peers than other directory peers do. A simple rotation scheme is to use Round Robin, where each directory peer takes turn to be the directory head for some pre-defined time period, while more elaborate schemes, drawn from the existing literature on load balancing, are also possible. When a new directory head is selected, the directory head sends the server a UpdSDT message including (sid, gid, pid), which are the session id, generation id, and peer id of the new directory head. Upon receipt of the message, the server uses (gid, sid) in the message to update pid of the matching entry in table SDT accordingly. The new directory head also broadcasts to all peers in
its generation a $UpdPDT$ message, and sends a $UpdPNT$ message to directory heads of neighboring generations found in the directory head’s $PNT$ table. When a peer receives the broadcast message $UpdPDT$, it updates the $PDT$ by setting the flag $isDirectoryHead$ properly for the old and new directory heads. Upon the receipt of the message $UpdPNT$, directory heads of the parent and child generations update their $PNT$ accordingly.

Within a directory, the directory head also periodically sends other directory peers update messages $SynPAT$, and $SynPNT$ to keep their index contents in tables $PAT$ and $PNT$ in sync. Message $SynPAT$ contains the difference between the directory head’s current $PAT$ and previous $PAT$ from the last sync attempt. When a directory peer receives the update messages, it updates its tables $PAT$ and $PNT$ accordingly.

When a peer no longer wants to serve as a directory peer, it can request the directory head to release it from the responsibility. The directory head in turn looks for a replacement from non-directory peers in the generation. If a replacement is found, the directory head transfers the contents of $PAT$ and $PNT$ to the new directory peer, and notifies the requesting directory peer that it can leave.

### 2.2.3 Selecting Forwarding Peer

A peer $p$ can have multiple peers that pass its eligibility test. This section discusses how peer $p$ selects a forwarding peer among its eligible peers. To shorten the notations, forwarding peers are called forwarders.

Assume that the requesting peer has enough inbound bandwidth to receive the video stream. VoD service has a stringent bandwidth requirement, but is relatively insensitive to the delay [19]; hence, it is preferable to select a peer with abundant unused bandwidth to be the forwarding peer of the requesting peer. Taking into account the bandwidth requirement, P2VoD considers two stopping conditions for the forwarder selection process:
• First Eligible: when an eligible peer with enough unused bandwidth has been found, and this eligible peer is selected as the forwarder.

• Best Eligible: when all eligible peers have been considered, and the eligible peer with the largest unused bandwidth is selected as the forwarder.

The naive method to select a forwarding peer is to let the requesting peer probe all eligible peers, until the stopping condition has been met. This probing technique has several drawbacks [36]: 1) when many large-scale services using their own probing techniques, this step becomes redundant, and prevents new application from leveraging information already gathered by other applications, and 2) implementing this probing technique is often quite subtle, because large-scale probing of end-hosts can raise intrusion alarms in edge networks as the traffic can resemble a DDOS attack.

Several research efforts have attempted to provide a common measurement infrastructure for distributed applications. P2VoD adopts the iPlane (Information Plane) technique in [36]. Different from other existing methods, iPlane can provide pair-wise bandwidth measurement between end-hosts. iPlane provides a user interface, in which given the ids of two end-hosts, iPlane can output the estimated available bandwidth of the path. Depending on the stopping condition, P2VoD can utilize iPlane to 1) compute the available bandwidth between the requesting peer and each eligible peer one by one until a satisfactory path is found, or 2) compute the available bandwidth between the requesting peer and every eligible peer.

Algorithm 2.1 FindForwardPeer(C, p)

Require: The requesting peer p, the set of eligible peers C, and the stopping condition
Ensure: Return a peer p’ in C that meet the stopping condition, or NULL if there does not exist such peer
1: p sends iPlane the list of eligible peers C
2: iPlane returns p’ or NULL
2.2.4 Join Algorithm

When a peer successfully joins P2VoD, there are two possibilities, either joining an existing video session or being the first member of a new session. In order for a new peer to join an existing session, there must be at least one peer in the session that is eligible to serve the new peer. If a new peer, somehow, knows that an existing session does not have any eligible peer to serve the new peer, it can safely skip probing peers in that session to find a forwarder. Theorem 2.2 states the condition for a session to be considered closed with respect to a requesting time; in other words, the condition guarantees that no peer in a closed session is eligible to serve the new peer. Note that the opposite of a closed session, namely an open session, only means that there may exist an eligible peer in the session for a requesting peer.

Theorem 2.2 A session cannot accept a new peer at time \( t \) if \( (g\text{Start} + 2T) < t \), where \( g\text{Start} \) is the start time of the youngest generation of the session; such session is called a closed session.

Proof: From Definition 2.2, if a peer \( p' \) is in the youngest generation, its arrival time must satisfy the condition \( p'.t \leq (g\text{Start} + T) \). Since the buffer size of \( p' \) is \( T \), \( p' \) passes the eligibility test of a new peer \( p \) if \( p.t \leq (p'.t + T) \). Therefore, if a new peer can find an eligible peer in the youngest generation, the following inequality must hold \( p.t \leq p'.t + T \leq (g\text{Start} + 2T) \). In other words, a new peer \( p \) cannot find any eligible peer in the youngest generation of the session if \( (g\text{Start} + 2T) < p.t \). Since the youngest generation of a session contains the latest arriving peers of the session, there will be no peer in the whole session that can pass the eligibility test of \( p \) if \( (g\text{Start} + 2T) < p.t \); this implies that the session cannot accept a new peer at time \( t \) if \( (g\text{Start} + 2T) < t \).

When a peer \( p \) wishes to join P2VoD, \( p \) contacts the server to start the joining process. The server \( s \) in turn executes Algorithm 2.2. The server first looks for an open session with respect to (w.r.t) \( p.t \), which \( p \) can possibly join (step 2 to step 10). If none of the existing
Algorithm 2.2 Joining Algorithm in P2VoD

Require: A peer $p$ wants to join P2VoD
Ensure: Connect $p$ to P2VoD or reject the joining request from $p$

The server $s$ carries the following steps when $p$ contacts $s$

1: initialize $\text{connectFlag}$ to FALSE
2: while there are more entries in $\text{SST}$ AND $\text{connectFlag}$ is FALSE do
3: let $S_m$ be the video session in the current entry in $\text{SST}$
4: let $G_n$ be the youngest generation of $S_m$ (i.e., by looking up in $\text{SDT}$)
5: if $p.t < (G_n.g\text{StartT ime} + T)$ then
6: $\text{connectFlag} \leftarrow \text{JoinExistingGeneration}(G_n.\text{dirHead}, p)$
7: else if $(G_n.g\text{StartT ime} + T) \leq p.t < (G_n.g\text{StartT ime} + 2T)$ then
8: $\text{connectFlag} \leftarrow \text{CreateNewGeneration}(G_n.\text{dirHead}, p)$
9: end if
10: end while
11: if $\text{connectFlag}$ is FALSE then
12: $\text{connectFlag} \leftarrow \text{CreateNewSession}(p)$
13: end if
14: if $\text{connectFlag}$ is FALSE then
15: the server rejects the joining request from $p$
16: end if

sessions can accommodate $p$, $s$ will try to create a new session for $p$ to join as shown in step 11 to step 13. Finally, if $p$ is still not connected to P2VoD, $s$ has to reject $p$ since both the server and existing peers can not accommodate $p$ at the moment.

There are two cases that peer $p$ can join an existing session:

- Case 1: $p$ joins the youngest generation of the session (step 6) according to Algorithm 2.3, if $p$ satisfies the membership requirement of the generation (step 5).

- Case 2: $p$ tries to join a newly created generation of the session (step 8) according to algorithm 2.4, if $p$ cannot join the current youngest generation but the session is still open w.r.t to $p.t$ (step 7).

In Algorithm 2.3, the directory head $\text{dirHead}$ of the generation $G_n$ attempts to take $p$ into this generation, given that $p$ satisfies the membership requirement of $G_n$. The directory head first looks in either $G_n$ or the parent generation of $G_n$ for existing peers, which can pass the eligibility test of $p$ (from step 4 to step 8). Among the eligible candidates, $p$ looks
Algorithm 2.3 JoinExistingGeneration(dirHead, p)

Require: The directory head dirHead and joining peer p
Ensure: Return 1 (or 0) if p is (or is not) admitted to the current generation {the following steps are executed at the directory head dirHead}
1: initialize isConnectedFlag to 0
2: \( C \leftarrow \emptyset \)
3: let pDirHead be the directory head of the parent generation (i.e., using dirHead.PNT)
4: for each peer \( p_i \) from dirHead.PAT and pDirHead.PAT do
5:   if \( p.t < p_i.t + T \) then
6:     \( C \leftarrow p_i \)
7:   end if
8: end for
9: if \( C \neq \emptyset \) then
10:   isConnectedFlag \( \leftarrow \) FindForwardPeer(C, p)
11: end if
12: if isConnectedFlag is TRUE then
13:   dirHead updates its PAT to include p in the index
14: end if
15: return isConnectedFlag

for the forwarding peer by calling Algorithm 2.1. If p is successfully connected to P2VoD, dirHead updates its index table PAT to include the information about p (steps 12 to 14).

In Algorithm 2.4, the directory head attempts to create a new generation for the joining peer p, given that p cannot be in the same generation \( G_n \) with dirHead. First, dirHead checks if any peer in \( G_n \) passes the eligibility test of p (steps 3 to 7). Among the eligible candidates, p looks for the forwarding peer by calling Algorithm 2.1. If p finds a forwarding peer, p becomes the directory head of the new generation, \( G_{n+1} \), and proceeds to update its data structures and the server’s data structures (steps 11 to 15).

When none of the existing sessions can accommodate p, using Algorithm 2.5, the server attempts to create a new session and let p join it. Steps 1-7 shows the case when the available bandwidth at s is still enough to support p. In this case, p is connected to s, and becomes the directory head of the first generation of the newly created session. s and p also update their data structures accordingly.
Algorithm 2.4 CreateNewGeneration($dirHead, p$)

Require: The directory head $dirHead$ and joining peer $p$

Ensure: Create a new generation, whose first member is $p$. Return 1 (or 0) if the generation is (or is not) created {the following steps are executed by the directory head $dirHead$}

initialize $isConnectedFlag$ to 0

$C \leftarrow \emptyset$

for each peer $p_i$ from $dirHead.PAT$ do
  if $p.i.t \geq p_i.t + T$ then
    $C \leftarrow p_i$
  end if
end for

if $C \neq \emptyset$ then
  $isConnectedFlag \leftarrow \text{FindForwardPeer}(C, p)$
end if

if $isConnectedFlag$ then
  $p$ becomes the directory head of the new generation
  $p$ and $dirHead$ update their data structures ($PNT$ for $dirHead$; $PDT$, $PAT$ and $PNT$ for $p$)
  $s$ updates its server-side data structures ($SST$) based on messages sent by $p$
end if

return $isConnectedFlag$

Algorithm 2.5 CreateNewSession($p$)

Require: A joining peer $p$

Ensure: Create a new video session for $p$. Return 1 (or 0) if $p$ is allowed (or rejected) to join P2VoD {the following steps are executed by the server $s$}

if server $s$ is still capable to serve $p$ then
  $p$ is connected to $s$
  $p$ becomes the directory head of the new generation $G_1$ of the new session
  $s$ updates its data structures, $SST$ and $SDT$
  $p$ updates its data structures, $PDT$, $PAT$ and $PNT$
  return TRUE
end if

return FALSE
2.2.5 Failure Recovery Algorithm

In a peer-to-peer environment like P2VoD, failures are expected to happen often due to the unpredictable behavior of users as well as the congested traffic in the underlying network. P2VoD uses a two-phase failure recovery protocol. The first phase requires a peer to detect a failure between it and the forwarding peer. Then the abandoned peer enters the second phase to recover from the failure by finding a new forwarding peer.

2.2.5.1 Detecting failures

Failures in P2VoD are categorized into two kinds: graceful and unexpected. Graceful failure refers to the case when a peer intentionally chooses to leave the system. On the other hand, unexpected failure accounts for network failure and software crash (e.g., buffer overflow). These two kinds of failures require different detection mechanisms.

- **Graceful failures:** When a peer decides to leave the system, it informs its receivers by sending a *Leave* message. These abandoned peers will then invoke the failure recovery algorithm in section 2.2.5.2 to find a new forwarder.

When a failure happens, P2VoD also needs to update its control data structures depending on the role of the leaving peer in its generation:

**Case 1:** If the leaving peer is a non-directory peer, the leaving peer informs the directory head of its intention, and the directory head updates its *PAT* accordingly.

**Case 2:** If the leaving peer is a directory peer, but not a directory head, the directory head has to: 1) updates its *PAT*, *PDT*, and 2) promotes a peer to directory peer using a similar process as shown in section 2.2.2.4 when a directory peer wants to become a non-directory peer.
Case 3: If the leaving peer is the directory head, the leaving directory head select a
directory peer to become the new directory head. In addition to steps in Case 2, the
new directory peer informs the server and neighboring generations of the change.

• Unexpected failures: Since failures in this category happens unexpectedly without any
explicit warning, peers in P2VoD are required to monitor their incoming traffic con-
stantly in order to detect failures. Three messages are used: TestError message,
NetworkRecovery message, and Wait message. If the quality of the video stream
received is below a threshold, a peer sends a TestError message to its forwarder. If a
peer doesn’t receive any response from its forwarder for three consecutive TestError
messages, then the connection is deemed as a failure. In case the forwarder can receive
the TestError message, there are two possibilities. If the forwarder has already used
up all available out-bound bandwidth, it sends a NetworkRecovery message suggest-
ing that receiver to initiate the failure recovery procedure in section 2.2.5.2 to connect
to a better forwarder. The second possibility is that the forwarder itself is involving in
another recovery process, it sends a Wait message asking the receiver to wait until the
recovery is finished. Note that when the connection is deemed as a failure, the receiver
will inform the directory head or one of the directory peers (in case the directory head
is the forwarder) to update the control data structures as shown in the case of grateful
failures.

2.2.5.2 Recovering from failures

When there is a failure at a peer \( p \) of a generation \( G \), the whole sub-tree under \( p \) is affected.
\( p \) initiates the recovery process, while the rest of the sub-tree are informed to wait through
the use of the Wait message. If \( p \) succeeds, the whole sub-tree is recovered. If \( p \) fails to find
a new parent, then \( p \) is rejected. The receivers of \( p \) will invoke the recovery process to try
to recover their own sub-tree. The process repeats recursively through the whole sub-tree. A disrupted peer $p$ uses the following steps to find a new forwarder for itself.

- **Step 1**: Using its PDT, $p$ contacts the directory head of its generation, $dirHead$. The directory head will attempt to reconnect $p$ by running a procedure similar to the Algorithm 2.3. If $p$ is reconnected, the recovery stops, else $p$ follows the following second step.

- **Step 2**: $p$ contacts the server $s$. $s$ will run a procedure similar to the Algorithm 2.5 to reconnect $p$. If $p$ is reconnected, the recovery stops, else $p$ will be ejected from the system. Note that these special sessions do not allow newly arriving peers to join as regular sessions.

2.3 Analysis of P2VoD

2.3.1 Network Capacity Amplification

The network bandwidth in a P2P-based VoD system available to serve new peers includes both that of the server and of other peers. Hence, it can be said that the network capacity of the traditional server-client system has been amplified in the P2P-based system. To facilitate the comparison in term of capacity amplification between P2VoD and P2Cast [19], assume that both systems have the same system threshold $T$. In other words, peers in P2VoD and P2Cast use the same buffer size $T$ to cache the video stream.

**Theorem 2.3** A new peer $p$ in P2VoD can select its forwarding peer from a set, which consists of the server and all peers, whose arrival times are no more $T$ time units earlier than $p$’s.
**Proof:** The theorem can be proven by tracing the join algorithm in section 2.2.4. According to the join algorithm, there are two cases when \( p \) joins P2VoD:

- **Case 1:** if \( p \) joins a new session, the server will be the forwarding peer for \( p \).

- **Case 2:** if \( p \) joins an existing session, \( p \) will join either the currently youngest generation or a new generation. As already shown in theorem 2.1, through the directory head of the currently youngest generation, all peers that pass the eligibility test of \( p \) are available for \( p \) to select its forwarding peer. In other words, all peers, whose arrival times are no more \( T \) time units earlier than \( p \)'s, are participants in the forwarder selection for \( p \). From theorem 2.3, it is concluded that given a threshold \( T \), P2VoD has maximized the capacity amplification of the system by fully utilizing all eligible peers.

On the contrary, there exist instances in P2Cast in which a new peer cannot select an existing peer to be the new peer's forwarder, even though the existing peer's arrival time is also no more \( T \) time units earlier than the new peer's. This statement about P2Cast can be proven by using a counter-example, which is shown below.

Assume \( T = 5 \), and there are five peers in the two systems when the systems’ snapshots are captured at time 9. The arrival times of the five peers are as follows: \( p_1.t = 0, p_2.t = 2, p_3.t = 4, p_4.t = 6, p_5.t = 8 \). In Fig. 2.2, the server only needs to forward the stream to one peer in P2VoD, while in Fig. 2.3 the server has to forward the base stream to two peers. When \( p_4 \) arrives to P2VoD at time 6, peers \( p_2 \) and \( p_3 \) still have the first video segment in their buffers, and \( p_2 \) is selected to forward the stream to \( p_4 \). On the other hand, when \( p_4 \) arrives to P2Cast at time 6, \( p_4 \) cannot join the same video session with \( p_1, p_2, \) and \( p_3 \), because \( p_3.t - p_1.t = 6 > T \). As a result, the server has to stream the video to \( p_4 \). This example shows that both P2VoD and P2Cast utilize peers’ network capacity to amplify the traditional server-client system, but P2VoD maximizes peers’ utilization while P2Cast does
2.3.2 Join Overhead

The join overhead indicates the delay it takes before a new peer $p$ can join P2VoD. The main contributing factor to the join overhead is the time it takes for $p$ to set up a communication session with an existing peer. To measure the join overhead, we measure the number of peers $p$ has to contact during the joining process.

Assume that if $p$ just needs to estimate the network delay between it and an existing peer, $p$ can query a centralized service as shown in section 2.2.3. Since $p$ does not need to contact an existing peer directly, the network delay measurement does not constitute a peer contact in this analysis.

From the join algorithm in section 2.2.4, there are two cases:
- Case 1: If all sessions are closed, the server will proceed to create a new session and make $p$ the first member of the first generation of the new session. In this case, the number of peers $p$ has to contact is 0, excluding the server.

- Case 2: If there are some open sessions, $p$ contacts each open session sequentially until $p$ is connected to P2VoD. For each open session, $p$ only needs to communicate with the directory head of the youngest generation of that session; the number of peers $p$ contacts in an open session is 1. The next question is what is the maximum number of open sessions $p$ has to contact? To answer this question, assume that 1) every peer in P2VoD can forward the stream to at least another peer, and 2) new peers join the system sequentially. The first assumption means that even though a new peer consumes some network bandwidth of the session, it also contributes enough network bandwidth to at least offset the consumption. In other words, the joining of a new peer to a video session does not decrease the overall network capacity of the session. The second assumption ensures that the session is never temporarily overwhelmed by concurrent join requests.

**Lemma 2.1** The start times of two generations in one session is different by at least $T$ time units.

**Proof:** The result of this lemma comes directly from the definition of a generation as shown in Definition 2.2.

Using lemma 2.1 and the above assumptions, we have:

**Lemma 2.2** The start times of the two youngest generations of any two sessions are different by more than $T$ time units.

**Proof:** A new session is created when none of the existing sessions can accommodate a new peer $p$. This means either 1) none of the existing sessions have enough unused bandwidth to serve the new peer or 2) there is not any peer in any session that passes the eligibility test of
the new peer \( p \), or for every peer \( p' \) in the system \( p'.t + T < p.t \). From the above assumptions, 1) cannot happen, so 2) is the case. For an existing session, \( gStart\_Time\_Current ≤ p'.t \), where \( gStart\_Time\_Current \) is the start time of the youngest generation and \( p' \) is a peer in that generation. For the new session, \( gStart\_Time\_New = p.t \), where \( gStart\_Time\_New \) is the start time of the youngest generation of the new session and \( p \) is the first peer of the generation. It follows that \( gStart\_Time\_Current + T < gStart\_Time\_New \). Therefore, the start times of the two youngest generations of any two sessions are different by more than \( T \) time units.

**Theorem 2.4** The maximum number of open sessions in P2VoD at any time will be 2.

**Proof:** At an arbitrary time \( t \), there are the following cases in P2VoD. If there exists some open sessions in P2VoD, the \( gStart\_Time \) of the youngest generation of these sessions must satisfy the following inequality \( t - 2T ≤ gStart\_Time ≤ t \). From Lemma 2.2, there can be no more two sessions, whose start time of the youngest generation satisfies the above inequality. Therefore, the maximum number of open sessions in P2VoD at any time will be 2.

From Theorem 2.4, it can be inferred that the maximum number of sessions a new peer \( p \) has to contact during a join is 2, or the number of peers \( p \) has to contact during a join is 2.

### 2.3.3 Failure Recovery Overhead

Similar to the join overhead, failure recovery overhead measures the delay it takes for a disconnected peer to reconnect to P2VoD under a failure. The failure recovery overhead is measured as the number of peers the disconnected peer has to contact during a recovery. From the failure recovery algorithm in section 2.2.5, a disconnected peer only needs to contact the directory head of its generation to look for a new forwarding peer. If the attempt is unsuccessful, the server streams the video to the disconnected member using a special
recovery channel. Therefore, the number of peers the disconnected peer has to contact during a recovery is 1.

### 2.3.4 Control Overhead

The control overhead accounts for non-data messages in P2VoD between two peers to maintain the control protocol described in section 2.2.2. From sections 2.2.2.4, 2.2.4, and 2.2.5, there are two kinds of control messages: the first kind is to maintain the integrity of the control protocol, and the second kind is to improve the robustness of the control protocol. The first kind of control messages comes from the join and recovery procedures 2.2.4, and 2.2.5. For each join or failure recovery, the peers that have to update their control data structures are: the server, the directory head, and the directory head of the neighboring generations.

The second kind of control message comes from the communication between the directory head and other directory peers in a generation 2.2.2.4. In each event, the peers a directory head has to contact are: the server, the directory peers in the same generation, and directory heads of the neighboring generations.

Therefore the control overhead in P2VoD is small, because for each event, the number of control messages exchanged is localized within each peer’s generation.

### 2.4 Performance Study

This section evaluates the performance of P2VoD, and how P2VoD compares against P2Cast [19] using the simulation-based approach.
2.4.1 Simulation Objectives

The following performance metrics are used in the simulation:

- **Network capacity amplification**: indicates how much peer bandwidth is utilized. The amplification can be measured by two parameters, the *server stress* and the *rejection probability*. The server stress measures the number of direct streams from the server. The rejection probability means the probability that a peer tries to join the system but cannot get the service.

- **Join overhead**: represents the time a new peer has to wait before receiving the video service. The join overhead is measured using the number of peer contacts during a join. In this simulation, when bandwidth estimation between the new peer and an existing peer is needed, the new peer will initiate the measurement and this measurement is counted as one contact.

- **Failure overhead**: represents the time a disrupted peer has to wait before reconnecting to another upstream peer. The failure overhead is measured using the number of peer contacts during a failure recovery.

2.4.2 Simulation Setup

A discrete-event simulator is developed for each of the two techniques, and is written in C++. In the simulation, the underlying network topology, as illustrated in Fig. 2.4, is created using the GT-ITM utility [60]. The whole network consists of one transit network (with 4 nodes), and 12 stub domains (with 96 nodes in total). Note that the 4 transit nodes are in the center of Fig. 2.4. Assume that each node in this network represents a local area network (LAN) with the ability to host unlimited number of peers, and to have enough bandwidth to support media streaming. Routing between two nodes is determined by using the shortest path algorithm. The bandwidth capacity of links in the network is assigned by default as
Figure 2.4: Network topology used in the simulation.

Table 2.1: Parameters used for performance evaluation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Variation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation time (minutes)</td>
<td>100</td>
<td>N/A</td>
</tr>
<tr>
<td>Video length (minutes)</td>
<td>100</td>
<td>N/A</td>
</tr>
<tr>
<td>Buffer size (percent of the video length)</td>
<td>10</td>
<td>5-50</td>
</tr>
<tr>
<td>Request arrival rate (requests/sec)</td>
<td>0.1</td>
<td>0.001-0.5</td>
</tr>
<tr>
<td>Backbone link bandwidth (number of concurrent streams)</td>
<td>20</td>
<td>12-25</td>
</tr>
<tr>
<td>Edge link bandwidth (number of concurrent streams)</td>
<td>5</td>
<td>N/A</td>
</tr>
<tr>
<td>Server location</td>
<td>Transit node</td>
<td>Transit or stub node</td>
</tr>
</tbody>
</table>

follows: a backbone link (at least one end point of the link is a transit node) can support 20 concurrent media streams, and an edge link (any link in the network, which is not a backbone link) can support 5 concurrent media streams. By default, the video server is placed at a transit node. Peers arriving to the system follow the Poisson distribution with arrival rate $\lambda$, and are randomly placed into one of the network nodes. There is one video available at the server with length of 100 minutes. Each simulation setting lasts for 100 minutes, and is repeated 100 times with different seed numbers. Each reported result is the average of the 100 simulation runs. Table 2.1 summarizes the parameters and their values used in the simulation.
2.4.3 Sensitivity Analysis

The sensitivity study considers the effect of the following variables:

- **Workload**: The *normalized workload* is defined as the total number of peers arriving to the system during a period equal to the length of the video.

- **Buffer size**: indicates the percentage of the video data each peer can cache in its buffer, which is also the threshold \( T \) discussed earlier.

- **Underlying network capacity**: considers different values for the maximal bandwidth of backbone links.

- **Server bandwidth**: The available bandwidth at the video server is varied by placing the server at either a stub node or a transit node.

2.4.4 P2VoD without Peer Failures

This section studies the effect of each of the three variables (buffer size, network link bandwidth, and server bandwidth) on the performance of P2VoD under different normalized workload sizes. The two forwarder selection methods, Best Eligible and First Eligible, perform similarly in terms of rejection probability and server stress, while the Best Eligible method requires more join overhead than the First Eligible method does. Hence, in this section, only results using the First Eligible selection method are reported.

Fig. 2.5 shows the effect of the buffer size on P2VoD. As expected, Fig. 2.5.a indicates that the rejection probability decreases when the buffer size increases, since more peers become eligible to serve an incoming peer. There are significant changes in the rejection probability when the buffer size increases from 5 percent to 10 percent, and to 15 percent. When the buffer size is 15 percent or larger, the improvement in rejection probability is small. Hence, the buffer size should be chosen to achieve a small rejection probability while also maintaining a small storage requirement on peers. From Fig. 2.5.b, the number of direct
Figure 2.5: Effect of buffer size on P2VoD.

streams from the server is always less than 8, indicating that most peers receive the data stream from other peers. The small server stress value also means that P2VoD can become almost self-sustained without the help from the server. Fig. 2.5.c shows that in all cases, the average number of peers contacted when a new peer joins P2VoD is less than 5 percent of the total number of peers in the system. Note that the total number of peers in the system is smaller than the normalized workload, since some peers are denied the service.

Fig. 2.6 shows the effect of the server bandwidth on P2VoD is negligible, because most peers receive the video stream not from the server, but from other peers. Fig. 2.7 indicates that the effect of the bandwidth of backbone links on P2VoD is also not significant. From the simulation setting in section 2.4.2, multiple peers can be found in one network node, and each network node represents a LAN with the ability to host unlimited number of peers. Therefore, many video stream connections consume bandwidth of neither a backbone link nor an edge link. This characteristic of the underlying network also helps to explain another observation in Fig. 2.5.a that for large enough buffer sizes (≥ 15), the rejection probability also decreases when the normalized work actually increases.

### 2.4.5 Comparison of P2VoD and P2Cast without Peer Failures

To verify the consistency of the reported results for P2Cast, simulation settings similar to those found in the P2Cast paper [19] are used in this section for both P2Cast and P2VoD.
Fig. 2.8 and Fig. 2.9 show the effect of the buffer size and the server bandwidth on the rejection probability of P2Cast. It is observed in Fig. 2.8 that the rejection probability of P2Cast decreases when the buffer size increases. Fig. 2.9 shows that the effect of the server bandwidth on the rejection probability of P2Cast is insignificant when the normalized workload is heavy, i.e. the number of peers join P2Cast is more than 500 during 100 minutes. In any case, the rejection probability always increases when the normalized workload is heavier.

Fig. 2.10 compares P2VoD and P2Cast when the buffer can store up to 15 percent of the video data. It is clear that P2VoD outperforms P2Cast on rejection probability and server stress. While the join overhead in P2VoD is higher than that in P2Cast, it should be noted that the number of peers in P2VoD is also much higher due to the low rejection probability.
2.4.6 P2VoD with Peer Failures

In this simulation, a peer failure is modeled by $p_f$, the probability a peer leaves the system early before it receives the entire video stream or before it finishes the streaming assignments to downstream peers. The request arrival rate is fixed at 0.05/second, and the client buffer can cache up to 15 percent of the video data. Fig. 2.11 shows the probability of number of peers contacted for recovery. Most peers (with a probability of 0.8612) are not disrupted at all by peer failures. The probability a peer has to contact more than 10 peers during a recovery is less than 0.04. The probability a forced departure happens is also measured (not shown in a figure) for different values of $p_f$, ranging from 0.1 to 0.9. The result shows that the probability of a forced departure is always less than 0.05.
Figure 2.11: Number of peers contacted for recovery when $p_f = 0.1$.

2.5 Related Works

IP Multicast as an extension of the Internet is designed for one-to-many and many-to-many communications at the network layer. As a result, IP Multicast is a natural choice as the underlying technology for VoD services. A numerous number of VoD techniques assuming IP multicast have been proposed. Based on video popularity, existing IP Multicast-based VoD techniques can be categorized into periodic broadcast approach and batching approach.

Periodic broadcast is proposed for delivering popular videos, and examples of this category include staggered broadcasting, Skyscraper, Client-Centric Approach, Stripping, Cautious Harmonic, and Pagoda [26]. In the periodic broadcast approach, each video is repeatedly broadcast on a number of network channels; a client tunes in one or few of these channels to watch the video. A limitation of the periodic broadcast approach is its non-zero startup delay. If the number of clients watching a video is small, the utilization of network channels broadcasting the video is low.

For less popular videos, it is more efficient to multicast the requested video on-demand instead of broadcasting it repeatedly [26]. However, in IP multicast, clients in a multicast group receive the same content; on the other hand, in VoD services, due to different arrival
times, it is likely that clients want different parts of a video at any given time. The Batching technique [26] proposes to group together multiple requests, within a short period of time, and serve them using the same multicast session. Even though Batching overcomes the mis-matching problem between IP Multicast and VoD services, one problem with Batching is the long startup delay for early clients in the batch. Patching technique is proposed to address the start up delay problem in order to provide videos truly on-demand [26]. Patching allows a late client to join an ongoing multicast session and still receives the video in full. The client gets the initial missing part of the video (a patch) from the server through an IP unicast connection. A problem with Patching is that the server can be overwhelmed with patching requests from clients.

Besides the aforementioned limitations of IP Multicast-based techniques, the deployment of IP Multicast itself on the Internet has been very slow due to several fundamental issues [11, 3]. To cope with the infeasibility problem of IP Multicast, researchers have proposed multicast services at the application layer, assuming only IP unicast at the network layer. Proposals for application layer multicast for multimedia applications, including VoD services, can be classified into infrastructure-based approach and peer-to-peer approach.

In the infrastructure-based approach [61, 43, 49, 29], a set of dedicated overlay routers are used to act as software routers with multicast functionality. The content is then transmitted from the source to a set of clients on a multicast tree consisting of overlay routers. A new client can join an existing media stream by connecting to an overlay router, who belongs to a delivery path to an existing client. This approach can alleviate the bottleneck problem at the source because a client can get services not only from the source but also from overlay routers. A drawback of this approach is the deployment/maintenance cost for the dedicated overlay routers.

To avoid the cost of deploying and maintaining additional hardware, peer-to-peer (P2P) approach does not use overlay routers. The multicast tree is built containing only the source and the clients, or peers. Existing techniques using P2P approach for media streaming can
also be categorized as live streaming and VoD streaming. Live streaming using P2P has been studied by a number of researchers [48,14,2,11,41]. However, these proposals are not readily applicable to P2P VoD streaming due to the following reasons. First, end-to-end delay is more important to live streaming than VoD streaming. In live streaming, the shorter the end-to-end delay is, the more lively, defined as liveness in [48], the stream is perceived by the users. In VoD streaming, liveness is simply irrelevant because the video stream is already pre-recorded. This fact implies that while a short tree rooted at the video server and spanned over peers is desirable in live streaming, it is not a necessary condition for the case of VoD streaming. Second, a peer joining an ongoing live streaming session is only interested in the stream starting from his/her joining time, while in the VoD streaming case the whole video must be delivered to the new peer. Hence, a good VoD system must find an efficient way to provide the initial missing part of the video to the latecomers. Third, the correlations between various variables are different for the two types of streaming. For example, a client will likely stop watching a VoD stream when its QoS degrades, but the client may not do the same thing for a live stream because the client does not have an option of watching it again in the future [51]. Therefore, it is expected that if the QoS of the video stream reduces, there will be many more peers leaving the system in VoD streaming case than the case of live streaming. This last observation stretches the importance of a robust failure recovery protocol in a VoD streaming system. In the following paragraphs, we discuss in more detail existing papers in the VoD streaming category [47,19,41,57].

CoopNet [41] employs a multi-description coding method for media content, and is intended to support both live and VoD streaming. In CoopNet, the media signal is decoded into several streams, or so called descriptions, in which every subset of them is decodable. CoopNet then builds multiple distribution trees spanning the source and the receivers, and each tree is used to transmit a single description. Therefore, a peer failure only causes the affected peers to lose some descriptions. This scheme works fine for live streaming case. For on-demand streaming, when the server receives such a request and is already overloaded,
the request is directed to some client, who has downloaded the requested stream (in full or partial) before and is willing to participate in CoopNet. There are several drawbacks with this method. It requires a huge buffer at the client to store the entire video. In case only partial of the video is found at the serving client, the requesting client has to look for the missing part from other clients, which increases the client’s startup delay. Another problem is that CoopNet puts heavy control overhead on the source because the source is required to maintain full knowledge of all distribution trees.

Xu et al [57] considers media streaming while taking peers heterogeneity in bandwidth capacity into consideration. With the limitation of outbound bandwidth, multiple supplying peers are needed to provide service for one receiving peer. They first look at the problem of media data assignment, determining which data segment of the video each supplying peer needs to send to the receiving peer. Their solution is optimal in term of minimizing the buffer delay. The paper also proposes a technique to amplify the capacity of the streaming system. While [57] assumes many-to-one relationship between supplying-peer and requesting-peer, we suppose that the relation is one-to-one in our P2VoD system. In other words, a supplying peer in P2VoD can provide the entire video stream to a requesting peer without the help of additional supplying peers.

The closest works to our technique are Chaining [47] and P2Cast [19].

- **Chaining** [47]: This is the first technique applying P2P concept in VoD streaming. Each client in Chaining has a fixed-size buffer to cache the most recent content of the video stream it receives. A new client can receive the video stream from an early client, as long as the first data block of the video is still in the buffer of the early client.

- **P2Cast** [19]: This technique applies the idea of Patching [26] to the application layer. As in Patching, when a new client joins P2Cast, it receives both a base stream and a patch stream. The patch stream stops when the new client catches up with the starting point of the base stream. In order to provide patch streams to new clients, clients in
P2Cast cache the initial part of the video. P2Cast offers a new implementation of the Patching technique at the application layer, and also makes use of clients as patch stream servers.

Both Chaining and P2Cast exploit the client resources, network bandwidth and storage, to reduce the demand on the server substantially. Both techniques, however, still have several drawbacks. Chaining is completely ignorant of the transience in client behavior. As a result, Chaining does not provide a recovery protocol when client failure happens. Moreover, the topology of Chaining is a chain, which makes the system very susceptible to failures; when a failure happens at a client near the start of the chain, most clients in the chain are affected. Even though P2Cast does take client transience into account in its design, P2Cast has other limitations too. A new client in P2Cast generally needs two streams at the beginning, a patch stream and a base stream, putting higher network bandwidth requirement on both serving and requesting clients. More importantly, P2Cast allows fewer clients to share the same server resource than Chaining does. In P2Cast, the extension of a session depends on the arrival time of the first client of that session. On the other hand, in Chaining, the extension of a chain only depends on the arrival time of the latest client of the chain.

2.6 Conclusion

This chapter presents P2VoD, a new technique for video-on-demand streaming in a P2P network. The main concepts in P2VoD are the caching scheme, the distributed directory service and generations. The analysis in section 2.3 has proven that the design of P2VoD is sound and efficient. Specifically, P2VoD maximizes peers’ utilization, while a competing technique, P2Cast, does not. Moreover, under certain assumptions, it is shown that the join and failure recovery overheads are small, in the order of $O(1)$. Additionally, P2VoD achieves these desirable characteristics with small control overhead. The simulation results
also confirm that P2VoD performs better than P2Cast in terms of important performance metrics.
3.1 Introduction

Due to the popularity of video services, users expect to be able to access videos anywhere anytime. There are two basic mediums for the users to access video services, namely wireline and wireless networks. While video services in wireline networks have been researched extensively [25], less attention has been paid to video services in wireless networks. Providing efficient video services in wireless networks is one step closer to achieving ubiquitous video services. This chapter presents a video-on-demand technique in wireless P2P networks.

On-demand video service in wireless networks has a number of potential applications. One example is to support the Advanced Traveler Information Systems (ATIS), which is one important component of Intelligent Transportation Systems (ITS) [1]. The broad goals of the ATIS are to provide collision warning technologies, traveler information, and driver convenience. In the context of ATIS, a wireless on-demand video service can be used to 1) disseminate captured scenes of an accident to serve as a collision warning or an indication of possible road congestion ahead, or 2) broadcast popular video clips to keep car passengers entertained. In a university campus setting, students can watch archived lectures with their portable video playback devices anywhere on campus. Another example is in a sport event setting such as a stadium or an arena, where users can use their personal wireless devices to watch replays, highlights, interviews, advertisements.

In [35], the authors argue that the vast majority of commercial wireless applications need high availability assurance; therefore, these applications are best supported through the managed infrastructure of carrier mobile networks. The simplest approach to stream video
data to users is to create a unicast connection from the server to each requesting user, where the video server is situated behind the base station. Given the high-bandwidth requirement of video data, this approach will quickly consume all available bandwidth of the downlink communication when multiple users request the video service. Multicast as a stream sharing mechanism can improve the spectrum utilization of wireless communication in multipoint services. The Multimedia Broadcast Multicast Service (MBMS) has been introduced as one possible multicast technique for 3G-cellular networks [16]. This solution, however, has its limitation [22]. Specifically, two important spectrum saving techniques, fast power control and packet scheduling based on channel quality used in 3G unicast, are not available in a multicast channel. To overcome these limitations in MBMS, another multicast technique for 3G-cellular network is proposed in [22]. The goal in this chapter is not to propose a new multicast service, but how to utilize the available multicast service to efficiently support video-on-demand service in a wireless network.

3.1.1 Description of Patching

Patching [7] is a technique that utilizes the multicast service at the network layer to enable true on-demand service on the Internet for videos with diverse popularity. The basic idea of Patching is to allow a client to join an existing multicast for the remainder of the video, and download the missed portion over a dedicated patching stream. Under Patching, most of the server bandwidth is organized into a set of logical communication channels, each capable of transmitting video data at the playback rate. A logical channel can be used to multicast a video in its entirety called a regular multicast, or to multicast the first portion of the video called a patching multicast. A logical channel is called either a regular channel when it is used for a regular multicast, or a patching channel when it is used for a patching multicast.

1User, node, client, and peer are used interchangeably in this paper
The data streams on a regular channel and a patching channel are called *regular stream* and *patching stream*, respectively.

A straightforward application of the Patching technique is to implement the Patching technique at the media server. This direct application of Patching still has limited scalability because every media stream still comes from the base station. To overcome the scalability issue of Patching in a wireless environment, one approach is to explore the wireless communication architecture. It is observed that a carrier mobile network and a mobile peer-to-peer network can co-exist in a typical wireless environment. This hybrid architecture combines the strength of the two types of networks. The carrier mobile network, also known as *Wireless Wide-Area Network* (WWAN) offers high availability assurance. The mobile peer-to-peer network, also known as *Wireless Local-Area Network* (WLAN), offers an opportunistic use of resources available at mobile clients.

### 3.1.2 Description of the Hybrid Wireless Architecture

Hybrid wireless network architectures have been considered in the past. Wei’s dissertation presents a survey on more than a dozen of recent proposed architectures [53]. Fig. 3.1 shows the general architecture used in this chapter. Each mobile device has two radio interfaces, enabling the device to communicate both in the WWAN and WLAN modes. Through the WLAN interface, mobile devices form a mobile peer-to-peer network by using the IEEE 802.11x protocol in the ad-hoc mode. Through the WWAN interface, mobile devices communicate with the base station using the 1xEV-DO (*Evolution-Data Only*) protocol, or also known as HDR (*High Data Rate*). Table 3.1 compares the main features of these two types of networks.
Table 3.1: Comparing two types of wireless networks

<table>
<thead>
<tr>
<th></th>
<th>WWAN</th>
<th>WLAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coverage Radius</td>
<td>up to 20,000m</td>
<td>up to 250m</td>
</tr>
<tr>
<td>Throughput (kbps)</td>
<td>38.6-2400</td>
<td>1,000-54,000</td>
</tr>
<tr>
<td>Operating Mode</td>
<td>infrastructure</td>
<td>access point or ad-hoc</td>
</tr>
</tbody>
</table>

3.1.3 Challenges and Contributions

This chapter presents PatchPeer, a video-on-demand technique for the wireless environment presented in Fig. 3.1. The basic idea of PatchPeer is to take advantage of the distinct features of the hybrid wireless network to overcome the scalability issue associated with the original Patching technique in a traditional wireless network. A client just joining PatchPeer can receive the regular stream from an existing multicast from the base station for the remainder of the video, and the patching stream from a neighboring peer for the missed portion of the video. The highly available WWAN is used to deliver the long-lasting regular streams, while the opportunity-driven WLAN is utilized to deliver the short-lived patching streams. It is expected that PatchPeer to scale better than the original Patching, as most of the patching streams in PatchPeer are provided by mobile clients themselves, leaving the base station with more downlink bandwidth to serve more clients (i.e., more regular streams). Designing PatchPeer, however, requires solving a number of new technical challenges.
There are two types of challenges in designing PatchPeer. The first type is associated with the integration of WWAN and WLAN in general, and the second type is specific to supporting video-on-demand in the hybrid wireless network. Integrating WWAN and WLAN in an efficient manner is not a trivial problem. In [10], the authors have pointed out important issues in integrating different wireless networks. At the time of this writing, many research problems in the first type of challenges are still active. As the two types of challenges are independent of each other, this chapter focuses on the second type of challenges.

Two fundamental characteristics in a mobile wireless hybrid network that influence the design of PatchPeer are node mobility, and the sharing nature of the wireless medium in single-channel networks such as IEEE 802.11x protocol. Node mobility can break the communication sessions between mobile nodes in the WLAN, and also change the network condition, such as number of on-going communication sessions, in a neighborhood. The sharing nature of the wireless medium dictates that within a neighborhood, there can only be a certain number of concurrent communication sessions if we want to maintain the QoS of the current streams. These two fundamental characteristics help us to understand the key technical challenges in designing PatchPeer.

The challenges for the PatchPeer design are twofold. First, a requesting node has to select a neighboring node to receive the patching stream. If the patching stream is provided by a multi-hop neighbor, the stream is subjected to long delay and frequent link break due to node mobility. On the other hand, if the patching stream is provided by a 1-hop neighbor, the list of patching stream providers is limited for a requesting client. Second, nodes in a WLAN contend with each other for the medium; hence, a neighboring node needs an admission control mechanism when admitting a patch request from a requesting node. Obviously, a neighboring node should not accept a patch request if this new patching stream affects the quality of the ongoing patching streams in the neighborhood. Moreover, since nodes in a WLAN operate in a distributed fashion, naturally the admission control mechanism should also be distributed to avoid overhead.
This chapter makes the following contributions. First, a hybrid wireless architecture is proposed for video-on-demand services. Second, the PatchPeer technique is designed for the hybrid wireless environment. Third, performance study is carried out to measure the performance of PatchPeer under various conditions. Fourth, this chapter can serve as a case study to illustrate the design principles when supporting a specific application in an integrated wireless network.

The rest of the chapter is presented as follows. The detailed design of PatchPeer is described in section 3.2. The results of the performance study is presented in section 3.3. Related works are discussed in section 3.4. Finally, the chapter is concluded in section 3.5.

3.2 The Proposed Technique: PatchPeer

3.2.1 System Overview

Fig. 3.2 shows the typical interactions between a requesting peer with its neighboring peers and the server in PatchPeer. The requesting peer first sends the server the ID of the requested video, and receives from the server the starting time of the latest regular multicast that is streaming the video. The ID of the video could be the video title, or a set of some keywords. The requesting peer then requests a patching stream from a neighboring peer to compensate for the initial missing part of the video. If a neighboring peer can provide the patching stream, the requesting peer receives the patching stream from the neighboring peer and the regular stream from the base station. Otherwise, the requesting peer receives both the patching stream and regular stream from the base station. One question in the design of PatchPeer is the scope of the search when a requesting peer looks for a patching stream provider among other peers. The search scope in PatchPeer is one-hop neighbors of the requesting peer because communicating with multi-hop neighbors has the following disadvantages. Since a multi-hop communication session is made of multiple wireless communication links and any
link break will affect the whole communication session, a multi-hop communication session is more susceptible to link breaks. Moreover, a multi-hop communication session consumes bandwidth of the shared channel in a wider area, causing more contention in the overall wireless network. One concern with the 1-hop search scope is that if the node density is low, a requesting peer may have problem finding a patching stream provider in its one-hop neighborhood. However, the simulation study shows that for many practical settings, the 1-hop search scope is adequate. Fig. 3.3 and Fig. 3.4 highlight the main operations of a requesting peer and the server in PatchPeer, respectively. A requesting peer has three main operations. First, the peer finds out the start time of the last regular multicast of the requested video (section 3.2.2). Second, the requesting peer finds a patching stream provider from its 1-hop neighbors (section 3.2.2). Third, the peer downloads and plays the video (section 3.2.4). The main operation of the server is to decide whether a patching stream or a regular stream is needed for a video request (section 3.2.3). Two other sections are not depicted in these two flow charts are 3.2.6 and 3.2.8. Section 3.2.6 proposes recovery procedures for PatchPeer under node mobility and node transient. Section 3.2.8 describes a number of possible optimization techniques that can improve PatchPeer further.
Figure 3.3: Flowchart Diagram for a Requesting Peer in PatchPeer

Figure 3.4: Flowchart Diagram for the Server in PatchPeer
3.2.2 Patching Stream Provider Search from one-hop Neighbor Peers

Each peer in PatchPeer caches the prefix of the video it is watching in a forwarding buffer (fb). The idea is early peers can use the video content in their fb to provide the patching stream to neighboring peers, who request the same video at a later time. Suppose a peer \( p \) wants to request a video \( v \). By contacting the server, \( p \) knows the time the last regular multicast of the video started. If there is no available regular multicast of the video, \( p \) issues a VideoRequest message to the server, and will be served the entire video by the server as shown later in section 3.2.3. Otherwise, \( p \) starts looking for a patching stream provider in its neighborhood by broadcasting a PatchRequest message to its one-hop neighbors. In order to be a patching stream provider for \( p \), a one-hop neighbor \( p' \) has to satisfy two conditions:

- **Data Condition**: the fb of \( p' \) must hold enough data of the requested video to compensate for the skew, defined as \( \text{currentTime} - \text{LastRegularTime} \), where \( \text{currentTime} \) is the current time and \( \text{LastRegularTime} \) is the time the last regular multicast of the requested video started.

- **Network Condition**: there must be enough bandwidth in the WLAN to support the new data flow from \( p' \) to \( p \) without compromising the QoS of existing data flows.

While the first condition is easy to check, the second condition warrants a separate research topic of its own [31, 58, 46]. In [58], the authors point out that existing approaches supporting QoS in wired networks or multichannel wireless networks are not applicable to single-channel wireless networks, such as IEEE 802.11 networks. The chief problem comes from the shared nature of the wireless medium in single-channel networks; any node with a wireless transmitter can transmit and contend for the wireless channel. To cope with the problem, the authors in [58] propose a contention-aware admission control for new flows in ad hoc networks, called CACP. CACP introduces a concept called \( c\)-neighborhood (carrier-sensing area) to represent the contention area of a node. The \( c\)-neighborhood of a node is
usually larger than the node’s transmission range. Fig. 3.5 illustrates the transmission range and carrier-sensing range for a mobile node. This c-neighborhood affects resource allocation at individual nodes and admission control of flows in two ways. First, allocation decisions at an individual node require bandwidth information of nodes outside of its communication range. Second, contention for resources may involve multiple nodes along a route. Note that strict QoS as in wired networks cannot be guaranteed in CACP, due to node mobility. Node mobility introduces two major issues: 1) routes can be broken frequently, and 2) there is no guarantee that resources will remain available since available bandwidth may decrease when communicating nodes move into the communication range of each other. The QoS commitment CACP provides is that no node intentionally breaks the QoS commitment to the flows in the network by admitting too many flows. PatchPeer adapts CACP solutions as follows.

![Carrier-Sensing Range of a Node](image1)

![Flow Bandwidth Consumption for a pair (S, D)](image2)

### 3.2.2.1 Prediction of Available Bandwidth

The first step is to estimate the c-neighborhood available bandwidth for a mobile node. C-neighborhood available bandwidth is the maximum amount of bandwidth that a node can use for transmitting without depriving the reserved bandwidth of any existing flows in its carrier-sensing range (c-neighborhood). Available bandwidth is measured as a weighted function of
the last estimation of the available bandwidth and the amount of unused bandwidth observed
during the period from the time of the last estimation to the current time.

Specifically, each mobile node uses the following formula to estimate the c-neighborhood
available bandwidth:

\[
B_{\text{neighbor}} = \alpha B_{\text{neighbor}} + (1 - \alpha) \frac{T_{\text{neighbor idle}}}{T_p} B_{\text{channel}}
\]

where weight \( \alpha \in [0, 1] \), \( B_{\text{channel}} \) is the channel capacity, \( T_{\text{neighbor idle}} \) is the amount of channel idle time during every period of time \( T_p \). Measuring \( T_{\text{neighbor idle}} \) is not trivial as c-neighbors may not be able to directly communicate with each other. One method to measure \( T_{\text{idle}} \) is the passive approach. In the passive approach, a \textit{Neighbor-carrier-sensing Threshold} is used. When the mobile node detects the signal strength of the wireless medium smaller than the \textit{Neighbor-carrier-sensing Threshold}, the channel is marked as idle.

3.2.2.2 Prediction of Bandwidth Consumption

Fig. 3.6 shows a source node \( p' \), a destination node \( p \) and their corresponding carrier-sensing ranges. Node \( p' \) is going to establish a communication session with node \( p \) to send the patching stream to \( p \). The bandwidth \( B_c \) consumed by the connection between \( p' \) and \( p \) is simply the video streaming rate, which equals the video playback rate. In Fig. 3.6, peers \( n_1, n_2, \) and \( p \) are in the c-neighborhood of \( p' \). If a communication session between \( p' \) and \( p \) is established for the patching stream, the available bandwidth for \( n_1, n_2, \) and \( p \) will be reduced by the amount of \( B_c \). On the other hand, even though \( n_3 \) is in the c-neighborhood of \( p \), since \( p \) only receives the patching stream, the available bandwidth at \( n_3 \) is not affected by the communication session between \( p' \) and \( p \). Since a communication session in PatchPeer only involves a peer and its one-hop neighbors, predicting the bandwidth consumption of a communication session in PatchPeer is much simpler than the case in CACP, where a communication session involves a multi-hop route.
3.2.2.3 Selection of a Patch Provider

Upon receipt of the *PatchRequest* message from the requesting peer $p$, a neighboring peer $p'$ determines if it satisfies the Data Condition, which is a simple check. Next $p'$ determines if the Network Condition can be met as follows. $p'$ first computes its c-neighborhood available bandwidth, $B_{neighbor}$, as shown in section 3.2.2.1. If $B_{neighbor} > B_c$, $p'$ passes the Network Condition; otherwise, $p'$ fails the Network Condition. If both conditions are met, $p'$ issues a *PatchRequestAck* message to $p$ with the *status* flag set to *TRUE*.

After a timeout period, $p$ checks all replies from its neighbors. If there is no positive response, i.e. no response with the *status* flag set to *TRUE*, $p$ sends a *VideoRequest* message to the server, and will be served by the server according to the procedure described in section 3.2.3. Otherwise, $p$ selects a candidate from the set of responding neighbors to be its patching stream provider, and sends a *JoinRegularMulticast* message to the server. When the server receives the *JoinRegularMulticast* message from the client, it simply enters the client to the list of clients that share the on-going regular multicast.

To select a patching stream provider from the set of candidate peers, $p$ can use one of the following heuristic selection methods.

- **Random Selection**: $p$ randomly selects the patching stream provider in the set of candidate peers. The advantage of this selection method is the workload is balanced among peers.

- **Earliest Response**: $p$ selects the neighboring peer who responds to the *RequestPatch* message at the earliest time. The advantage of this selection method is $p$ does not have to wait for the timeout period to expire to start receiving the patching stream.

- **Closest Peer**: assuming mobile nodes have knowledge of their locations, i.e. mobile nodes are equipped with positioning devices such as Global Positioning System (GPS).
In this selection method, \( p \) selects the candidate that is closest to \( p \). The idea is the close distance keeps two mobile nodes within the transmission range longer.

- **Most Available Bandwidth**: \( p \) selects the patching stream provider with the most available bandwidth among the set of candidate peers. When other communicating nodes move into the communication range of the patching stream provider, the data rate of the patching stream may be affected. With ample available bandwidth at the provider, this selection method will likely maintain the data rate of the patching stream under dynamic conditions of the network.

### 3.2.3 Server Main Routine

A token of the *VideoRequest* message contains \((pID, vID, |pb|)\), where \( pID \) and \( vID \) are the identification of the requesting peer and the requested video, \(|pb|\) is the size of the playback buffer employed at the requesting peer. There is a subtle difference between the two buffers, forwarding buffer and playback buffer, at a peer. The forwarding buffer \( fb \) is used by each peer to cache the prefix of the video, such that a peer can *provide* a patching stream to another peer later. On the other hand, the playback buffer \( pb \) is employed by each peer to cache the regular stream of the video, while the peer is *receiving* the patching stream from either the video server or another peer. When the server receives the *VideoRequest* message, if the server does not have any free channel, the request will be rejected. Otherwise, the server executes Algorithm 3.1. We describe the main components of Algorithm 3.1 as follows.

If there is no regular multicast currently serving video \( v \), the free channel is used to start a new regular multicast for \( v \) (steps 2-5). Otherwise, *skew* and *workload* are computed (steps 6, 7-11). If *skew* exceeds the size of the optimal patch window, the free channel is used for a new regular multicast (steps 12-13). Otherwise, the free channel is used for a patching multicast for a duration of *workload* (steps 14-17). Finally, the server’s data structures are
updated (step 18), and the requesting peer is notified of the ids of the downloading channels (step 19). The server uses Alg. 3.2 to update its data structures when a new client joins an on-going regular multicast, or when a batch of clients are served the requested video in its entirety.

**Algorithm 3.1 Server Routine**

**Require:** v: the requested video  
W(v): size of the optimal patch window for v  
|pb|: size of the playback buffer  
LastRegularID, LastRegularTime: channel ID and start time of the last regular multicast of v  
FreeID: ID of the free channel  

**Ensure:** Send IDs of the downloading channels to the requesting peer

1: Set RID = NULL, PID = NULL, and Workload=0  
2: if none of the existing regular multicast currently serving v then  
3: RID = FreeID, PID = NULL  
4: Jump to step 18  
5: end if  
6: skew = currentTime − LastRegularTime  
7: if skew < |pb| then  
8: tempWL = skew  
9: else  
10: tempWL = |v| − min{|pb|, |v| − skew}  
11: end if  
12: if skew > W(v) then  
13: RID = FreeID, PID = NULL  
14: else  
15: RID = LastRegularID, PID = FreeID  
16: Duration of the patch stream: Workload = tempWL  
17: end if  
18: Send UpdFrmServer message to Alg. 3.2  
19: Issue a message containing (RID, PID) to the requesting peer
Algorithm 3.2 Update Server Data Structures

Require: JoinRegularMulticast message: message from a peer to join an existing regular multicast
UpdFrmServer message: update message from the server

Ensure: Update the client list of appropriate server channels
Update LastRegularChannel
1: if JoinRegularMulticast message then
2: The server extracts the following variables from the message:
   RID: ID of the regular channel
   PeerID: ID of the requesting peer
3: Add PeerID to the client list of RID
4: else {UpdFrmServer message}
5: The server extracts the following variables from the message:
   RID: ID of the regular channel
   PID: ID of the patching channel
   PeerList: the list of peers in the selected batch
6: if RID! = NULL then
7: Add peers in PeerList to the client list of RID
8: if RID! = LastRegularChannel then
9: LastRegularChannel = RID
10: end if
11: end if
12: if PID! = NULL then
13: Add peers in PeerList to the client list of PID
14: end if
15: end if

3.2.4 Loading and Playing Video at Peers

A peer p executes Algorithm 3.3 to download and play the video. The peer uses two loaders, $L_p$ and $L_r$, for loading the patching stream and the regular stream, respectively. There are the following cases:

- Case 1: The patching stream comes from a neighboring peer, and the regular stream comes from the server.

- Case 2: Both the patching stream and regular stream come from the server.

- Case 3: p receives the entire video from the server’s new regular multicast.
The VideoPlayer playbacks the data as follows. Data received by loader $L_p$ from the patching stream is rendered immediately by the VideoPlayer. At the same time, data received by loader $L_r$ from the regular stream is stored in the playback buffer $pb$, using First-In-First-Out replacement policy. When there are no more data arriving from the patching stream, the VideoPlayer renders data from $pb$. The VideoPlayer stops when there are no more data from $pb$. The prefix of the video is also saved in the forwarding buffer $fb$.

**Algorithm 3.3 Peer Main Routine**

**Require:** $RID$: the ID of the regular channel $PID$: the ID of the patching channel $L_p$: loader for the patching stream $L_r$: loader for the regular stream $VideoPlayer$: video player at the peer

**Ensure:** Load and play the video
1: Load the regular stream from channel $RID$ using $L_r$
2: if $PID! = NULL$ then
3: Load the patch stream from channel $PID$ using $L_p$
4: end if
5: Play the video using the $VideoPlayer$
6: Save the prefix of the video in the forwarding buffer $fb$

### 3.2.5 Caching at Forwarding Buffers

The forwarding buffer at each peer can store a number of video prefixes. When the buffer is full, a peer needs to evict an existing video prefix in the buffer to create space for the incoming video prefix. The available buffer space of a peer is divided into a number of equal-sized chunks; each chunk can store a video prefix. Peers use one of the following replacement policies to decide which video prefix to be evicted:

- **Least Frequently Used (LFU):** When a peer uses a video prefix in a chunk to send a patching stream to a neighboring peer, the $usedCounter$ associated with that chunk is increased by one. In LFU, a chunk with the smallest $usedCounter$ will be evicted.

- **Least Recently Used (LRU):** When a peer uses a video prefix in a chunk to send a patching stream to a neighboring peer, the $lastTimeUsed$ associated with that chunk
is updated to the current time. In LRU, a chunk with the oldest lastTimeUsed will be evicted.

- First In First Out (FIFO): When a video prefix is assigned to a chunk, the cacheTime associated with that chunk is updated to the current time. In FIFO, a chunk with the oldest cacheTime will be evicted.

- Random: A peer picks a random chunk to evict its current content, and assigns the incoming video prefix to this chunk.

### 3.2.6 Recovery Procedures

While communication between mobile clients and the base station in the WWAN is fairly stable, communication disruptions are expected to happen quite often in the WLAN. Two main sources of communication disruptions are node mobility and node transient.

#### 3.2.6.1 Recovery Procedure for Node Mobility

When two peers participating in a patching stream connection move far apart such that they are no longer in the transmission range of each other, the ongoing patching stream will be disconnected. The receiver \( p \) can perform out-of-range detection by monitoring the transmission rate of the patching stream. To recover from the disconnection, the receiver initiates the search for a new patching stream provider using the following procedure. Denote \( v[a] \) as a prefix of length \( a \) of the requested video \( v \), where \( 0 < a < |v| \). \( p \) computes the missed portion of the patching stream as follows:

\[
remWL = [v[slew], v[dis]]
\]

where \( slew \) is the length of the patching stream computed in section 3.2.2, and \( dis \) is the length of the portion of the patching stream that \( p \) has received up to the disconnecting time.
broadcasts a RecoverRequest message to its one-hop neighbors to look for a new patching stream provider. Upon receipt of the RecoverRequest message from $p$, a neighboring peer $p'$ determines if it satisfies the Data Condition, i.e. whether the $fb$ of $p'$ contains the missed portion $remWL$ of the patching stream. Next $p'$ determines if the Network Condition can be met as shown in section 3.2.2.3. If both conditions are met, $p'$ is a candidate to be the new patching stream provider for $p$. After a timeout period, $p$ selects the new patching stream provider from the set of candidates. If there is no available candidate, $p$ contacts the server to receive this missed portion of the patching stream. If the server has a free channel, $p$ will be served the remaining patching stream. Otherwise, video service for peer $p$ is deemed as not recoverable.

3.2.6.2 Recovery Procedure for Node Departure

Peer departure in PatchPeer can be categorized as graceful and graceless departures. A graceful departure is when a peer leaves the system voluntarily, for example when the peer finishes watching the video or the peer is no longer interested in the video service. A graceless departure is when a peer leaves the system involuntarily, due to problems such as software failures. Detecting peer departure in two cases can be done as follows. In graceful departure, the leaving peer sends a Leave message to the server and to affected peers, i.e. peers that are either sending or receiving the patching stream to or from the leaving peer. In graceless departure, affected peers can find out the status of the leaving peer by monitoring their one-hop neighborhood. The server can also detect a graceless departure if each peer sends the server a KeepAlive message periodically. After a peer departure is detected, the next step is how the server and affected peers respond to the detected departure. For the server, it just simply removes the departing peer from the list of clients of any channel the leaving peer is in. Similarly, the providing peer just needs to remove the connection between itself...
Figure 3.7: Peer Locations

3.7.a: At time 9

3.7.b: At time 11

Figure 3.8: Multicast Streams

3.8.a: At time 9

3.8.b: At time 11

and the departing peer. For a peer receiving the data stream from the departing peer, the former recovers by using a procedure similar to what is described in section 3.2.6.1.

3.2.7 An Example

This section shows a simple scenario to illustrate the main operations of PatchPeer. Assume that 1) all peers in this example request the same video, 2) their playback and forwarding buffers have the same size, capable of storing five time units of one video, and 3) the optimal patch window size is also equal to five time units of video data. Note that the time unit is measured in term of playback time, say one minute of playback time.
At time 0, peer $p_1$ joins the system. Since $p_1$ is the first peer requesting the video, $p_1$ receives the entire video from a regular stream $R_1$ from the video server. At time 6, peer $p_2$ joins the system. Since there isn’t any peer in the neighborhood of $p_2$, and the patch window size for this video is $W = 5$, $p_2$ also receives the entire video from a regular stream $R_2$ from the video server. At time 9, peer $p_3$ joins the system. This time, $p_3$ receives the patching stream from $p_2$, and the regular stream from $R_2$. Fig. 3.7.a and Fig. 3.8.a show the relative locations of peers, and the different multicast streams at time 9. At time 11, $p_2$ moves out of the communication range of $p_3$, but $p_3$ has not finished downloading the patching stream yet. $p_3$ first searches in its neighborhood to find a replacement for $p_2$, and it finds $p_1$. $p_3$ then resumes the downloading of the remaining of the patching stream from $p_1$. Fig. 3.7.b and Fig. 3.8.b show the locations and multicast streams of PatchPeer at time 11.

### 3.2.8 Optimizations and Discussions

#### 3.2.8.1 Discovery of $LastRegularTime$

The discovery of the start time of the last regular multicast described in section 3.2.2 can be categorized as client-pull, as each peer probes the server for the information when they need it. An alternative approach is server-push, in which the server periodically broadcasts the start time of the last regular multicast of each video the server is currently serving. When a peer wants to find out the start time of the last regular multicast of a video, instead of querying the server, the peer listens on the broadcast channel for the needed information. The broadcast channel can use any of the existing techniques for efficient indexing on air channels [28]. Whether the client-pull or server-push approach is more efficient depends on a number of factors, including the number of requests and the popularity of a video. This chapter does not focus on evaluating these two approaches, and simply assumes that the client-pull approach is used.
3.2.8.2 Ad-hoc Assisted Multicast Routing

In section 3.2, the regular streams are broadcast from the server, and reach every node in the system. The broadcast radius has two effects on the wireless system: 1) base station’s power: since wireless signal is weaken as distance to the source increases, larger broadcast radius requires more power from the base station, and 2) radio resource: larger broadcast radius causes wireless interference to other communicating channels in a wider area. The authors in [22] propose to use a smaller broadcast radius, and utilize the local area network to relay the broadcast message to nodes outside the broadcast coverage area. Fig. 3.9 illustrates the technique proposed in [22], in which nodes inside the MZONE (Multicast Zone) act as gateways to relay broadcast message to outside nodes.

PatchPeer can use a similar scheme described above to multicast the regular streams. A peer in PatchPeer is now responsible for sending both the patching stream and regular stream; the added responsibility means a heavier workload on peers. Moreover, a peer failure has a bigger impact on the PatchPeer system, as a disrupted peer now has to recover both the patching and regular stream.
3.2.8.3 Multicasting Patching Stream from a Patching Stream Provider

Wireless communication is inherently broadcast. If a patching stream provider is a qualified candidate for multiple requesting nodes, it is more efficient to multicast the patching stream from the patch provider to all of these nodes in one broadcast instead of using multiple separate unicast connections to serve them.

3.2.8.4 Handover with Multiple Base Stations

In practice, mobile nodes in PatchPeer can move from the coverage area of a base station (BS\textsubscript{1}) into that of another base station (BS\textsubscript{2}). Our design for one base station can be easily extended to support multiple base stations. The primary concern when handover happens is to ensure that mobile nodes receive video streams from the new base station without disruption. Assume a mobile node \( p \) moves from BS\textsubscript{1} to BS\textsubscript{2}. BS\textsubscript{1} identifies all channels that have \( p \) in their client list. BS\textsubscript{1} transfers information of these channels to BS\textsubscript{2}. For each channel, the transferred information includes a token \((\text{videoID}, \text{startTime}, \text{workLoad}, \text{clientList})\), where \text{videoID} is the ID of the video being served by this channel, \text{startTime} is the time the channel starts multicasting the video, \text{workLoad} is the portion of the video the channel needs to send, \text{clientList} is the list of IDs of clients receiving the video stream from this channel. The state of a channel can be identified by \((\text{videoID}, \text{startTime}, \text{workLoad})\); two channels are considered to be identical if they have the same state. When BS\textsubscript{2} receives the information of the transferred channel, BS\textsubscript{2} first checks whether any of its busy channels is identical to the transferred channel. If there is an identical channel, the server just simply informs the new mobile node \( p \) to join the existing channel. Otherwise, if BS\textsubscript{2} still has a free channel, it uses this free channel to multicast the remaining workload for \text{videoID}, and informs \( p \) to join the newly formed multicast channel. Finally, if BS\textsubscript{2} does not have any free channel, \( p \) will be denied the video service.
3.3 Performance Study

The number of accepted requests is used as the performance metric in this simulation study. A request is considered to be accepted if the requesting peer receives and plays the entire video without loss of frame (jitter) and long delay (glitch).

The first study considers the case when nodes are stationary in order to verify the performance of the technique in a less dynamic environment. Next, PatchPeer is compared against the original Patching technique [7]. Finally, PatchPeer is evaluated when nodes are moving.

The sensitivity analysis study is done with respect to the below simulation parameters:

- Cache issues: include buffer size, chunk size and cache replacement policy.
- Video characteristics: include number of videos, video length, and video popularity.
- User characteristics: include request arrival rate, and user mobility.
- Other system variables: include operating area and node density.

Table 3.2 summarizes the parameters used in the simulation. The downlink bandwidth for the WWAN (base station) is fixed at 2400 kbps. The communication between the base station and a mobile node is error-free. The bandwidth for the WLAN is fixed at 6000 kbps. The transmission range of a node in the WLAN is 250m. The carrier-sensing range is equal to the transmission range. For the WLAN, if two nodes are within each other’s transmission range, the transmission is error-free. Nodes in PatchPeer move according to the Random Waypoint mobility model. User requests follow the Poisson distribution. Video length and video popularity follow the log normal distribution and zipf distribution, respectively [30]. The scheduling algorithm used by the server is First Come First Serve. The Closest Peer is used as the selection method to select the patching stream provider from a set of candidate peers.

The simulator is a discreet event simulator, written in C++, and simulates PatchPeer and Patching [7]. The simulator only simulates the media stream level, and does not simulate the
Table 3.2: Parameters used for performance evaluation

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Variation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation Time (hours)</td>
<td>10</td>
<td>N/A</td>
</tr>
<tr>
<td>Operating Area (mxm)</td>
<td>1000x1000</td>
<td>1000² – 3000²</td>
</tr>
<tr>
<td>Mean Request Inter-arrival Time 1/λ (seconds)</td>
<td>5</td>
<td>5-20</td>
</tr>
<tr>
<td>Number of Mobile Nodes</td>
<td>100</td>
<td>100-1600</td>
</tr>
<tr>
<td>Transmission Range (m)</td>
<td>250</td>
<td>N/A</td>
</tr>
<tr>
<td>WLAN Bandwidth (kbps)</td>
<td>6000</td>
<td>N/A</td>
</tr>
<tr>
<td>Client Forwarding Chunk Size (minutes)</td>
<td>20</td>
<td>5-60</td>
</tr>
<tr>
<td>Number of Chunks</td>
<td>2</td>
<td>1-4</td>
</tr>
<tr>
<td>Client Playback Buffer Size (minutes)</td>
<td>optimal</td>
<td>N/A</td>
</tr>
<tr>
<td>WWAN Bandwidth (kbps)</td>
<td>2400</td>
<td>N/A</td>
</tr>
<tr>
<td>Number of Videos</td>
<td>30</td>
<td>1-200</td>
</tr>
<tr>
<td>Normal Playback Rate (kbps)</td>
<td>300</td>
<td>N/A</td>
</tr>
<tr>
<td>Average Video Length (minutes)</td>
<td>45</td>
<td>10-60</td>
</tr>
<tr>
<td>Skew Factor for Video Popularity (zipf distr.)</td>
<td>0.271</td>
<td>0.1-0.7</td>
</tr>
<tr>
<td>Mean Speed (mps)</td>
<td>3</td>
<td>1-10</td>
</tr>
<tr>
<td>Speed Delta (mps)</td>
<td>0.3</td>
<td>0.1-1</td>
</tr>
<tr>
<td>Mean Pause Time (seconds)</td>
<td>5</td>
<td>N/A</td>
</tr>
<tr>
<td>Pause Time Delta (seconds)</td>
<td>1</td>
<td>N/A</td>
</tr>
</tbody>
</table>

network protocol stack. While detailed simulation of the network protocol stack certainly resembles realistic scenarios very closely, it is unnecessary for the simulation objectives and also too time-consuming in many simulation experiments with large number of nodes and long simulated time. Each simulation setting is run 100 times with different seed numbers, and the reported result is the average value of these 100 runs.

3.3.1 PatchPeer with Stationary Nodes

3.3.1.1 Cache Issues

Different replacement policies are studied with varying buffer sizes and chunk sizes. Figures 3.10, 3.11, and 3.12 correspond to buffer sizes of 20, 40, and 60 minutes, respectively. Each of these figures shows the results for three different cache replacement policies, namely LFU, LRU, and FIFO in sub-figures a, b and c. Each sub-figure displays the results for a
Figure 3.10: Effect of replacement policy with buffer size of 20 minutes

specific cache replacement policy with different chunk sizes. For all figures, the x-axis is the simulation hours, and the y-axis is the total number of accepted requests.

For the same setting in terms of the forwarding buffer size and the number of chunks in the buffer, all three cache replacement policies behave similarly. For this reason, one cache replacement policy, LFU, is used to explain these figures. If the number of chunks in the forwarding buffer is the same, PatchPeer performs better when the size of the buffer increases. As the buffer size increases, the size of each chunk in the buffer also increases; hence, each chunk can store a longer video prefix. With a longer video prefix, an early peer can satisfy the Data Condition to serve many more peers who request the same video later. The result is PatchPeer can accept more requests. Another observation is for the same buffer size, PatchPeer accepts more requests when there are fewer chunks in the buffer. With the same buffer size, a buffer with fewer chunks means less number of video prefixes is cached in the buffer, but each chunk can store a longer video prefix.

3.3.1.2 Video Characteristics

This section investigates how video characteristics affect the performance of PatchPeer. Figures 3.13.a, 3.13.b, and 3.13.c show the effect of number of videos, skew factor of video popularity distribution, and video length. Note that the default number of videos in this section is 100.
The performance of PatchPeer decreases as the number of videos increases as shown in Fig. 3.13.a or the skew factor increases as shown in Fig. 3.13.b. In both cases, for a requesting peer the likelihood of finding in its one-hop neighbors a peer, who has the prefix of the video being requested, is smaller.

Fig. 3.13.c shows that PatchPeer serves more requests when the average length of videos is shorter. With shorter videos, the workload of each patching stream is also shorter. A short patching stream connection releases the network bandwidth it uses quicker. Hence, more patching streams can share the WLAN channel during the same period of time, resulting in more accepted requests.
3.3.2 Patching vs. PatchPeer

PatchPeer is compared against Patching [7] in this section. The performance of Patching can be affected by the client playback buffer size, inter-arrival time of requests, and video length [7]. The optimal patching window as shown in [7] is used to determine the size of the client playback buffer size; hence, PatchPeer is only compared against Patching under the effect of the inter-arrival time of requests, video length, and number of videos.

For all figures, the x-axis is the number of simulation hours, and the y-axis is the number of accepted requests. Figures 3.14, 3.15, and 3.16 show that PatchPeer consistently outperforms Patching in all conditions, confirming that the additional resources contributed by peers indeed improve the scalability of the overall system.

Fig. 3.14 indicates that the performance of both PatchPeer and Patching is sensitive to the number of videos. When the number of videos increases from 1 to 100, the number of accepted requests after 10 simulated hours in PatchPeer reduces from 7,000 to around 450, while that number in Patching reduces from 225 to 117. When there are more videos in Patching, it is less likely to find a regular channel that is multicasting the same video a user is requesting. Hence, much less number of requests can share a regular multicast, resulting in a quicker depletion of the server network bandwidth and less number of accepted requests. PatchPeer can mitigate the problem in Patching. When there is no regular channel multicasting the video that is being requested, PatchPeer will not be able to help. On the
other hand, if there is an on-going regular multicast of the requested video, and the server has already used up its network bandwidth, the requesting peer can still get a patching stream from a neighboring peer. However, when there are more videos, there will be less number of neighboring peers that satisfy the Data Condition in section 3.2.2 because the probability a peer stores a certain video prefix in its forwarding buffer is smaller. As a result, the performance of PatchPeer also decreases when there are more videos.

While Patching’s performance is limited by the WWAN, PatchPeer’s performance is upper-bound by the total network capacity of the WWAN and WLAN. Fig. 3.15 confirms that PatchPeer is able to scale better than Patching under heavier workload.

In Fig. 3.16, both Patching and PatchPeer can receive more requests when the average video length is shorter. In the case of Patching, with shorter videos, the time each regular and patching multicast occupies a server channel is shorter, resulting in faster release of the server resources. Consequently, Patching can accept more requests. Similarly each patching stream established between two peers in PatchPeer is shorter for shorter videos; hence, the WLAN bandwidth can be shared by more patching streams, resulting in more accepted requests in PatchPeer too.
3.15.3 PatchPeer with Moving Nodes

A number of experiments are carried out to evaluate the effect of node mobility on PatchPeer with different forwarding buffer sizes, video lengths, and node densities. Fig. 3.17 shows that when the buffer size increases, PatchPeer’s performance improves the most when node mobility is low. This improvement is intuitive, because larger buffer size means an early peer can satisfy the Data Condition to serve many later requests, and low node mobility means smaller probability of disconnection for a patching stream between two peers. The combined effect results in more number of accepted requests.

In Fig. 3.18, for the same average video length, PatchPeer accepts more requests when node mobility is smaller. For the same average node speed, PatchPeer accepts more requests when the average video length is shorter.
In Fig. 3.19, for the same average speed, PatchPeer has similar performance for different node densities. The indication is that even with low density (100 nodes), PatchPeer can already reach its peak performance.
3.4 Related Work

This section first surveys video delivery techniques in wireless environments. Next, admission control mechanisms for WLANs are discussed. Finally, a number of different applications in hybrid wireless environments are presented.

The two systems that are most closely related to PatchPeer are [23, 59]. In [23], the paper uses the WLAN in two modes, access point and ad hoc. The base layer of the media stream is delivered through the access point, while the enhancement layers are delivered through multiple paths in ad hoc mode. The hybrid architecture in [23] only utilizes the WLAN while PatchPeer uses both WWAN and WLAN. In [59], the paper focuses on finding an energy efficient solution in supporting media streaming applications in wireless hybrid peer-to-peer systems. Since the WWAN interface consumes significantly higher energy than the WLAN interface, the paper proposes two collaborating modes, master-slave and peer-to-peer, where clients collaborate to help stream the media content from the server. Both of these previous works focus on improving the unicast connection between the media server and mobile clients, while PatchPeer considers multicast connections.

In [9], the authors propose a stadium-scale wireless media streaming using WLANs. The paper utilizes three key technologies in their paper: multi-channel WLANs, power control, and cache-and-relay. Using multi-channel WLANs and power control, the paper arranges the network in a hierarchical architecture. Each level of the system hierarchy uses a different channel for communication, and the coverage area of a proxy in an upper level includes the coverage area of children proxy nodes in the lower level. Comparing to PatchPeer, this paper considers unicast streaming connections only and it considers a different architecture, utilizing only the WLANs.

MobiVoD, proposed in [50], is a broadcasting video system in wireless local area networks. The MobiVoD architecture includes a video server, a number of local forwarders, and clients. Local forwarders receive video stream broadcast from the video server, and re-broadcast the
video stream to clients. The local forwarder is just another mobile node in MobiVoD. As staggered broadcasting has high start-up delay, MobiVoD reduces this high service delay by using client patching. A newly arriving client uses two channels to download the video; the first channel is used to retrieve the on-going broadcast from the local forwarder, and the second channel is used to download the missing part of the first video segment from a neighbor of the client. MobiVod focuses on popular videos, while PatchPeer handles videos with diverse popularity. Moreover, MobiVoD is using only the wireless local area networks, while PatchPeer operates in a hybrid wireless environment. There are few other works that also consider periodic broadcasting in wireless environment for video delivery. The authors in [5] presents an adaptive broadcasting approach, in which the requirement on the server bandwidth adapts with the number of clients. Another paper [27] extends the idea in [5] by taking into account clients’ heterogeneous capabilities. In [45], the authors propose to do periodic broadcast in a pure MANET.

In [18], the authors present V3, an architecture to support live video streaming in a vehicle-to-vehicle (V2V) network. In V3, vehicles on streets form an ad hoc network. When a vehicle in V3 wants to receive a video capture of the road network region ahead, it sends the message through the vehicle network to some destination vehicle inside the wanted region. The destination vehicle does a video capture of the region, and sends the video back to the requesting vehicle through the vehicle network. While V3 focuses on live video streaming, PatchPeer concentrates on video-on-demand. Moreover, V3 only utilizes the V2V network, while PatchPeer uses the WWAN for availability assurance and WLAN such as V2V networks for opportunistic communication.

In [54], the author addresses the problem of multipath unicast and multicast video communication over wireless ad hoc networks. The author proposes Inference aWare Multipath Routing (IWM), which minimizes the Packet Drop Probability (PDP) of two node-disjoint paths in a wireless ad hoc network. Another problem considered in the dissertation is Multiple tree Video Multicast. The basic idea is to split the video into multiple parts and send
each part over a different tree, which are ideally disjoint with each other so as to increase robustness to loss and other transmission degradations. The author proposes two solutions for Multiple tree Video Multicast, a Serial Multiple Disjoint Tree Multicast Routing (Serial MDTMR) and a Parallel Multiple Nearly-Disjoint Tree Multicast Routing (Parallel MNTMR).

WiVision is a real-time multimedia distribution system, capable of delivering both real-time video and on-demand playback, developed at SUNY Stony Brook [13]. The system consists of three main components: 1) acquisition component which digitizes the captured media content into compressed streams, 2) the storage, management, and distribution component that stores the metadata and the content for future playback, as well as is designed to enable live streaming, and 3) the client software which is designed to receive the multimedia streams over wireless channels and display them. The most interesting point of the system is the streaming of video data to mobile wireless clients over wireless links. The paper points out that there are three methods to deliver video data to clients, unicast, multicast and broadcast. Citing that unicast is not scalable and multicast for the lack of well documented support by WiFi hardware vendors, the paper chooses the broadcast approach. The paper concludes that up to two broadcast channels can be supported reliably using IEEE 802.11b protocol. In WiVision, live video is broadcast, while on-demand video is unicast to clients.

In [12], the authors propose a patching algorithm for an overlay multicast scheme for supporting VoD in wireless networks. Client nodes communicate using wireless medium. The basic idea of the paper is as follows. Nodes employ FIFO cache to buffer the video. Late clients can receive the stream from early clients if the latter still have the first video block in its cache. The paper points out that wireless medium is unstable; hence link error is expected. The link error can downgrade the quality of the video playback at clients. The paper proposes to use patch streams from the server to fill in the ”’gaps’” in client buffer that are caused by link error. To support analytical study, and simulation modeling, the authors use a two-state Markov chain to model link error. Nodes in [12] are stationary. Hence, the
paper does not have to deal with node mobility. Moreover, nodes in [12] cache the most recent part of the video, while nodes in PatchPeer only cache the prefix of the video.

In [31], the authors present models, algorithms and evaluation of VoIP on wireless mesh networks. The paper also points out that admission control is important in wireless meshes when supporting voice streams. There are at least two relevant works that propose solutions for admission control in ad hoc wireless networks. In [58], the paper considers the problem of resource allocation and admission control for multi-hop wireless routes in ad hoc networks. In their model, the authors consider interference from carrier-sensing area, which is typically larger than the transmission range of a node. Another paper [46] considers admission control problem for single-hop ad hoc wireless networks.

Wei’s dissertation presents a survey on more than a dozen of recent proposed hybrid wireless architectures [53]. UCAN (Unified Cellular and Ad-hoc Network) architecture is proposed in [35]. The goal of UCAN is to improve the cell’s aggregate throughput, while maintaining fairness [35]. The key idea that allows UCAN to achieve both of these contradicting goals is the *opportunistic* use of the IEEE 802.11 interfaces to improve the wide-area cell throughput. When receiving data from the base station, if a destination client experiences low HDR downlink channel rate, instead of transmitting directly to the destination, the base station transmits the data frames to another client (proxy client) with a better channel rate. The authors in [10] outline the challenges of integrating cellular networks with MANETs. The paper looks at open research problems at different layers of the network protocol stack.

A number of recent papers look at efficient methods to support multicast in hybrid wireless architectures. The document in [16] is a technical report on the vision of broadcast and multicast. The report explains the importance of multicast in both wireless and wireline networks, and discusses the 3G Cellular MBMS (Multimedia Broadcast Multicast Service). The authors in [33] consider QoS issues when multiple groups are present in a wireless hybrid network. The basic idea is to minimize the load of base stations by selecting appropriate
multicast groups to be handled by ad hoc networks. The group selection problem is transformed into a knapsack problem. In [21, 20, 22], the authors argue for a hybrid architecture to support high-data rate services. The authors propose an architecture for multicast, in which the 3G cellular infrastructure is assisted by multiple local mobile ad hoc networks in forwarding the multicast packets. The key idea behind the paper is to reduce the 3G coverage needed for multicast streaming by utilizing local MANETs. In [20], the authors propose two multicast routing protocols, one strongly centralized and one distributed protocol. The work in [42] has similar goal to [21, 20, 22], that is one-to-many communication with the assistance from ad hoc networks. This paper [42] considers the effect of ad hoc path interference. Since clients in a multicast group experience different channel conditions, it is hard for a multicast sender to send at a rate that is suitable to all receivers. The main idea of the proposed solution is to use client proxies, who are experiencing good cellular channel condition, to forward multicast data to other clients, who are experiencing poor cellular channel condition. The general idea of using client proxies has been proposed in UCAN [35], but UCAN considers unicast communication while this paper studies multicast communication. The technical highlight of the paper is to find ad hoc paths from receivers experiencing poor channel condition to proxies while taking into account ad hoc path interference, and specific features of the CSMA/CA mechanism.

3.5 Summary

PatchPeer is a scalable video-on-demand delivery technique for wireless environments that overcomes the limitations of the original Patching technique in a traditional wireless network. Utilizing the available resources from mobile clients, PatchPeer allows the video-on-demand system to scale beyond the bandwidth capacity of the server. The design of PatchPeer offers solutions to several challenging issues, most notably the handling of node mobility, and admission control in shared single-channel wireless medium networks. The performance
study indicates that PatchPeer is a clear winner over Patching on acceptance ratio under a wide range of environment settings.
CHAPTER 4: CONTINUOUS MOVING RANGE QUERIES IN MOBILE PEER-TO-PEER NETWORKS

4.1 Introduction

A location-based service allows the users query for information based on their own locations and/or other users’ locations. The explosive growth rate of the number of location-aware mobile wireless devices, ranging from navigational systems in vehicles to handheld devices and cell phones, means future location-based services need to employ scalable architectures to support a large and growing number of users and more complex queries. The research community [44, 6, 17, 39, 34, 52] has spent considerable efforts in finding scalable solutions to support continuous spatial queries, an important class of these location-based services. A continuous spatial query is a location-related query that is evaluated continuously for a specified time period.

One common assumption in most of previous works on continuous spatial queries is the availability of some fixed network infrastructure, such as a carrier mobile network. A centralized server, situated at the base station of the mobile network, is used to communicate with mobile nodes for location updates, and to manage queries. Some recent works [8, 17, 34, 15, 56] propose to offload the centralized server by distributing query monitoring task to mobile nodes. In reality, a fixed network infrastructure is not always available in many practical situations, such as in a battlefield or a remote area. In some other cases, bidirectional communication is not supported by the existing network infrastructure, such as in satellite communication.

This chapter explores new networking environment to support continuous spatial queries when a fixed network infrastructure is not available. Specifically, mobile nodes themselves form an ad-hoc network, or a Mobile Peer-to-Peer (MP2P) network, to communicate with
each other. A scalable technique, called ExtRange (Extended Range), is proposed to answer continuous moving range queries in a MP2P network. Such queries have many applications in real life. One example is a soldier in a battlefield queries for his friendly units in his neighborhood while he is moving to a target. Another example is a taxi driver can monitor the list of potential customers while the driver is driving on a street.

The proposed ExtRange is a distributed technique, where each mobile node manages its own issued queries and neighboring nodes participate to monitor those queries. ExtRange introduces two notable concepts, extended range and safe period. For each query with a given query range, the extended range of that query is a limited extension of the query range; mobile nodes that are within the extended range of a query monitor the query, and communicate significant changes to the query-issuing node (query node) to ensure the accuracy of the query result. Mobile nodes determine when to communicate with a query node using the concept of safe period. A safe period is a time period during which movement of a mobile node is not going to affect the result of a monitored query. By combining the two concepts judiciously, ExtRange ensures the accuracy of the query results as well as low associated overheads.

In summary, the main contribution in this chapter is the proposed technique ExtRange, a scalable solution to continuous moving range queries in a MP2P network. Concrete proofs show that ExtRange guarantees the accuracy of the query results when communication in the network is perfect. The extensive simulation study shows that ExtRange delivers high accuracy for query results with low associated costs under realistic simulation settings. The simulation study also introduces a meaningful performance metric to measure result accuracy, where both precision and recall are taken into account. The rest of the chapter is organized as follows. Section 4.2 describes the proposed technique. Section 4.3 presents the evaluation of ExtRange. Section 4.4 discusses works related to ExtRange. Finally, the chapter is concluded in section 4.5.
4.2 The Proposed Technique: ExtRange

4.2.1 System Assumptions

Each mobile node has a mobile device that can store data, and carry out computation tasks. A mobile device is also equipped with a positioning device such as GPS, and a wireless network interface card such as a IEEE802.11x operating in ad-hoc mode. Mobile nodes communicate with each other using broadcast or unicast protocols. A node uses a broadcast protocol to send messages to a group of nodes. Two nodes communicate with each other using a unicast routing protocol. The maximum speed of all mobile nodes, $MaxSpeed$, is known.

4.2.2 Models

A mobile object $o$ can be modeled by the following tuple: $<oid, loc_t>$, where $oid$ is the object’s id and $loc_t$ is the location of the object at time $t$. A query object is a mobile object that issues a continuous range query. A continuous range query $q$ can be formally defined using the following tuple: $<qid, qo, r, er, ts, dur>$ where

$qid$ : the id of the query, assigned by the query object.

$qo$ : the query object, which includes object’s id and location

$r$ : the radius or range of the query.

$er$ : the extended range of the query.

$ts$ : the timestamp when the query is issued.

$dur$ : the period the query should last from the issuing time.

---

$^1$ Node and object are used interchangeably in this paper
The result of $q$ is a list of mobile objects, in which the Euclidean distance between each of the objects and the query object is no greater than the query range $r$. The Euclidean distance between two locations $loc_1$ and $loc_2$ is denoted as $d(loc_1, loc_2)$. A circle is denoted as $C(o, r)$, where $o$ and $r$ are the center and radius of the circle, respectively. For ease of description, assuming that there is one query object and one query unless noted otherwise.

### 4.2.3 Initial Step

At time $t_0$, query object $qo$ initiates a query $q$ by broadcasting the query to mobile nodes in the circle $C(qo.loc_{t_0}, er)$. The radius of the broadcast is the extended range of the query $er$. The extended range $er$ is bigger than the query range $r$ and can be defined as follows: $er = r + \epsilon$, where $\epsilon > 0$. The key idea behind the concept of extended range is to allow nodes that are currently outside of the query range but may move into the query range momentarily aware of the query. Another way to view the role of the extended range is that the extra broadcast area provides a protective layer to ensure the integrity of the query result under the dynamic condition of mobile nodes.

To explain the role of the extended range in a specific example, assume that $qo$ is stationary. Fig. 4.1 shows a case when $\epsilon > 0$, in which the outer circle represents the broadcast circle with radius equal to the extended range and the inner circle represents the query circle with radius equal to the query range. If $\epsilon = 0$, or $er = r$, mobile object $o_3$ in Fig. 4.1 is not aware of query $q$ because the initial broadcast from the query object $qo$ does not reach $o_3$. When $o_3$ moves inside the inner circle, the result of $q$ is not updated accordingly because $o_3$ does not report its location to $qo$. Note that in the general case, the broadcast circle and the query circle of a query change over time as the query object moves.

When a mobile object $o_i$ inside the broadcast circle receives the broadcast message, it enters $q$ to the list of monitored queries. Object $o_i$ also computes its Safe Period ($SP$) for
query $q$ as follows:

$$SP = \frac{|d_{t_0}(o_i, qo) - r|}{2 \times \text{MaxSpeed}}$$  \hspace{1cm} (Eq. 4.1)

where $d_{t_0}(o_i, qo)$ is the shorthand for the Euclidean distance between the location of $o_i$ and that of $qo$ at the current time $t_0$ ($d(o_i.loc_{t_0}, qo.loc_{t_0})$).

**Theorem 4.1 (Safe Period)** During $[t_0, t_0 + SP]$, the membership of $o_i$ with respect to the result set of $q$ is unchanged.

**Proof:** During a time $t$, the maximum distance $o_i$ and $qo$ can move far apart from each other (or closer to each other) is $(2 \times t \times \text{MaxSpeed})$, which happens when the two objects move away from (or toward) each other at $\text{MaxSpeed}$. There are two cases:
• $o_i$ is inside the query range of $q$ at time $t_0$, i.e. $d_{t_0}(o_i, q) \leq r$: During $SP$, the maximum distance the two objects can move far apart from each other is:

\[
2 \times SP \times \text{MaxSpeed} = 2 \times \frac{r - d_{t_0}(o_i, q)}{2 \times \text{MaxSpeed}} \times \text{MaxSpeed} = r - d_{t_0}(o_i, q).
\]

Since the distance between the two objects at time $t_0$ is $d_{t_0}(o_i, q)$, at time $(t_0 + SP)$ the maximum distance between the two objects is less than $r - d_{t_0}(o_i, q) + d_{t_0}(o_i, q) = r$, or $o_i$ is still inside the query range of $q$.

• $o_i$ is currently outside the query range of $q$: During the $SP$, the maximum distance the two objects can move closer to each other is $d_{t_0}(o_i, q) - r$

Since the distance between the two objects at time $t_0$ is $d_{t_0}(o_i, q)$, at time $(t_0 + SP)$ the minimum distance between the two objects is greater than $d_{t_0}(o_i, q) - (d_{t_0}(o_i, q) - r) = r$, or $o_i$ is still outside the query range of $q$.

4.2.4 Monitoring Step

Each mobile object $o_i$ monitors the $SP$ associated with each query that $o_i$ is aware of. When the $SP$ associated with query $q$ is expired, $o_i$ initiates a unicast connection with $qo$ to inform the query object of the current location of $o_i$. When the query object receives the new location update, it determines whether $o_i$ should be included in the result set of $q$. The query object in return also replies $o_i$ with the current location of the query object. Upon the receipt of the query object’s location update message, $o_i$ computes the current distance between itself and the query object. If the distance is not greater than $er$, $o_i$ computes the new $SP$ as in Eq. 4.1 and goes back to monitor the query. Otherwise, $o_i$ does not need to monitor query $q$ anymore, and removes $q$ out of its list of monitored queries.
4.2.5 Refreshing Step

After a query broadcast message is sent, while mobile objects inside the broadcast circle of the query are aware of the query and update the query object in a timely manner, objects outside the broadcast circle are not aware of the query and may affect the query result sometime in the future without the knowledge of the query object. To overcome this challenge, the query object periodically sends a query broadcast message to the current broadcast circle of the query. The question is how often a query broadcast message will be sent. A satisfying answer has to account for two conflicting goals, which are to ensure the accuracy of the query result and to save the network resource as much as possible. Theorem 4.2 shows the necessary and sufficient condition to ensure that no mobile node from outside of the broadcast circle at one time enters the query circle at another time without the knowledge of the query object.

Theorem 4.2 (Refresh Rate) Given a query \( q \), and two consecutive query broadcast messages at times \( t_1 \) and \( t_2 \). Define \( WT = t_2 - t_1 \) as the wait time between two consecutive query broadcast messages. The following value for \( WT \) is the necessary and sufficient condition to ensure that no mobile object from outside of the broadcast circle at time \( t_1 \) enters the query circle at time \( t_2 \).

\[
WT = \frac{\varepsilon}{2 \times \text{MaxSpeed}}
\]

(Eq. 4.2)

**Proof:** The necessary condition is proven by contradiction. Assume the time difference between two consecutive query broadcast messages is \( WT' = t_2 - t_1 \), where \( WT' > WT \). Object \( o_i \) is just right outside of the broadcast circle of query \( q \) at time \( t_1 \), i.e. \( d_{t_1}(o_i, qo) = er + \delta \), where \( 0 < \delta < 2 \times (WT' - WT) \times \text{MaxSpeed} \). During period \([t_1, t_2]\), the two objects
move toward each other at the speed of $MaxSpeed$. We have:

$$d_{t_2}(o_i, qo) = d_{t_1}(o_i, qo) - 2 \times MaxSpeed \times WT'$$

$$< d_{t_1}(o_i, qo) - 2 \times MaxSpeed \times WT - \delta$$

$$< er + \delta - \epsilon - \delta$$

$$< er - \epsilon$$

$$< r.$$

or object $o_i$ is in the query circle of query $q$ at time $t_2$.

The sufficient condition is proven as follows. Consider the worst case scenario, an object $o_i$ just outside the broadcast circle of $q$ at time $t_1$, and the distance between $o_i$ and $qo$ is $(er + \delta)$, where $0 < \delta$. During $WT$, $o_i$ and $qo$ move toward each other at $MaxSpeed$. The distance between the two objects after $WT$ is:

$$d_{t_2}(o_i, qo) = d_{t_1}(o_i, qo) - 2 \times MaxSpeed \times WT$$

$$= er + \delta - \epsilon$$

$$= r + \delta.$$

or object $o_i$ is still outside of the query circle of query $q$ at time $t_2$.

4.2.6 Data Structures for ExtRange

4.2.6.1 Data Structures at Mobile Objects

Each mobile object has a Local Query Table ($LQT$) to store the continuous queries that the object is aware of. $LQT$ has the following fields:

- $qid$: id of the monitored query.
• **oid**: id of the query object.

• **loc**: location of the query object.

• **r**: the query range.

• **er**: the extended range.

• **ts**: time the query is issued.

• **dur**: duration of the query.

• **SP**: the safe period of the query, computed by the mobile object.

### 4.2.6.2 Data Structures at Query Objects

Each query object has a Query Table (**QT**) to store the issued queries. **QT** has the following fields:

• **qid**: id of the query.

• **r**: the query range.

• **er**: the extended range.

• **ts**: timestamp when the query was first issued.

• **dur**: duration of the query.

• **WT**: the wait time between two consecutive query broadcast messages.

• **lastSent**: timestamp when the last query broadcast message was sent.

• **resultList**: a set containing ids of mobile objects, who are inside the query circle.
4.2.7 Proof of Correctness

Theorem 4.3 (Result Accuracy) Assuming broadcast and unicast messages get to the intended parties, the query result in ExtRange is always accurate.

Proof: Since the query message is re-broadcast after a period of $WT$, it is enough to show that the query result is accurate between two consecutive query broadcast messages. Assume at time $t$, a query broadcast message is sent by the query object for query $q$. The focus is in the accuracy of the query result between $[t, t + WT)$. At time $t$, after the query broadcast message has been sent to all mobile nodes inside the broadcast circle, nodes can be divided into two sets: 1) the inside set including nodes inside the broadcast circle, and 2) the outside set including nodes outside the broadcast circle. For nodes in the outside set, at time $t$, they are obviously not included in the result set of query $q$. According to Theorem 4.2, during the period $[t, t + WT)$, no node in the outside set can enter the query circle of $q$. In other words, during $[t, t + WT)$, it is guaranteed that no node in the outside set will be included in the result set of $q$. For nodes in the inside set, at time $t$, some nodes in this set are included in the query result and some nodes are not. During $[t, t + WT)$, it is possible that nodes in the inside set change their membership in the query result. Theorem 4.1 ensures that mobile nodes changing their membership with respect to the query result update the query object in a timely manner. In summary, for those nodes unaware of the query, these nodes do not affect the query result during $WT$ anyway. For those nodes that can affect the query result during $WT$, they are all aware of the query and update the query object in a timely manner, ensuring the accuracy of the query result at all times.

4.2.8 Optimization

Previous sections describe the basic operations of ExtRange. There are several ways to possibly enhance the performance of ExtRange. This section describes some of these enhancement techniques as follows.
4.2.8.1 Extending SP

When SP is expired, the mobile object needs to update its location with the query object. Eqn. Eq. 4.1 presents the formula for SP in the worst case scenario, as such it may over-estimate many situations in reality. From this observation, in certain cases when the SP is expired, a location update may not be needed. Specifically, when the SP is expired, the mobile object estimates the current location of the query object and decides whether it needs to send a location update message as follows.

Consider the case when mobile object $o_i$ is in the query result of $q$ during SP. Assume $t_1, t_2$ are the timestamps at the beginning and the end of SP. At time $t_2$, the mobile object estimates the farthest location from the query object using the following formula:

$$MaxDist_{t_2} = d(o_i.loc_{t_2}, qo.loc_{t_1}) + SP \times MaxSpeed$$  \hspace{1cm} (Eq. 4.3)

The second term of the right hand side of the equation is the maximum distance that $qo$ can travel during $SP$. The meaning of Eqn. Eq. 4.3 can be visualized as follows. During the $SP$, the query object can only move to some place inside the circle $C_1 = C(qo.loc_{t_1}, SP \times MaxSpeed)$. The furthest point the query object can be from the mobile object at time $t_2$ is a point, which is on circle $C_1$ and opposite to $o_i$ through the circle center.

If at time $t_2$, the estimated distance between $o_i$ and $qo$ computed by Eqn. Eq. 4.3 exceeds the query range, the mobile object updates its location with the query object as shown in section 4.2.4. Otherwise, $o_i$ can re-compute its $SP$ without contacting the query object. In the second case, $o_i$ re-computes $SP$ as follows:

$$SP = \frac{r - MaxDist_{t_2}}{2 \times MaxSpeed}$$

Fig. 4.2.b shows an example where at the end of $t_2$, $o_i$ does not need to send location update to the query object.
4.2.a: At time $t_1$

4.2.b: At time $t_2$ ($x = SP \times MaxSpeed$)

Figure 4.2: Mobile object is inside the query result.

Similarly, for the case when the mobile object is outside the query result of $q$ during $SP$, at time $t_2$ the mobile object estimates the closest location to the query object using the following formula:

$$MinDist_{t_2} = d(o_i.loc_{t_2}, qo.loc_{t_1}) - SP \times MaxSpeed$$  \hspace{1cm} (Eq. 4.4)

If at time $t_2$, the estimated distance between $o_i$ and $qo$ computed by Eqn. Eq. 4.4 is smaller than the query range, the mobile object updates its location with the query object as shown in section 4.2.4. Otherwise, $o_i$ can re-compute its $SP$ without contacting the query object.

In the second case, $o_i$ recomputes $SP$ using the following formula:

$$SP = \frac{MinDist_{t_2} - r}{2 \times MaxSpeed}$$
4.2.8.2 Delaying Location Update

When sending location update to the query object is absolutely necessary, a mobile object can still delay the location update message, but with the tradeoff that the query result is no longer guaranteed to be accurate. While providing approximate answer is not a focus in this chapter, this technique provides a good calibrating tool, where the system operator can decide the tradeoff between communication cost and result accuracy.

4.2.8.3 Sharing Location Update among Multiple Queries

Assume mobile object \( o_i \) is monitoring queries \( q_1, q_2, ..., q_i \) for the same query object \( q_o \). In the basic algorithm of ExtRange, \( o_i \) manages each query independently. \( o_i \) computes the \( SP \) for each query, and communicates with \( q_o \) when the \( SP \) of a query is expired. Communication saving can be achieved if \( o_i \) manages these queries of \( q_o \) jointly. The idea is when \( o_i \) receives new location information of \( q_o \), \( o_i \) can extend the \( SP \) of all of these queries instead of just one query.

Fig. 4.3 shows an example how queries share one location update to extend their \( SP \) and the benefit of sharing. The upper rectangle of Fig. 4.3 shows three queries with their respective \( SP \) during period \([t_0, t_2]\), when \( o_i \) manages each query independently. The lower rectangle of Fig. 4.3 shows how \( o_i \) can extend the \( SP \) of queries \( q_2 \) and \( q_3 \) after \( o_i \) communicates with \( q_o \) when the \( SP \) of \( q_1 \) is expired, if \( o_i \) manages queries jointly. During period \((t_1, t_2)\), in the case of independent query management, \( o_i \) needs to communicate with \( q_o \) twice since the \( SP \) of \( q_2 \) and \( q_3 \) are expired in this period. For the same time period, in the case of joint query management, \( o_i \) does not need to communicate with \( q_o \), as \( o_i \) is able to utilize the new location update from \( q_o \) to extend the \( SP \) of \( q_2 \) and \( q_3 \).
4.2.8.4 Refreshing Multiple Queries

Consider the case when multiple queries issued by the same query object have the same WT value. Fig. 4.4 shows two queries issued by the same query object, and Fig. 4.5 shows the periodic broadcast schedule for two queries at the query object. If the query object \(q_0\) manages each query independently, \(q_0\) needs to broadcast a refresh message at times \(t_0, t_1, t_2, t_3, t_4, t_5\). On the other hand, if \(q_0\) manages the two queries jointly, it only needs to send a query broadcast message at times \(t_0, t_1, t_3, t_5\) for the following reason. Mobile nodes in the broadcast circle \(C(q_0.loc, er_1)\) are also inside the broadcast circle \(C(q_0.loc, er_2)\) as \(er_2 > er_1\). At time \(t_1\), when \(q_0\) broadcasts a query message to \(C(q_0.loc_t_1, er_2)\), mobile nodes in \(C(q_0.loc_t_1, er_1)\) also receive the broadcast message. Therefore, at time \(t_2\), \(q_0\) does not need to broadcast a refresh message to mobile nodes inside \(C(q_0.loc, er_1)\). From time \(t_1\), \(q_0\) just needs to execute the broadcast schedule for \(q_2\) since the two refresh schedules have been merged. The only change in the content of the query broadcast message of \(q_2\) is that \(q_0\) needs to include the id of query \(q_1\).
4.2.9 Reliability Issues

Failures are common in MP2P networks. Software and hardware failures may prevent nodes from carrying out computing and networking tasks. Moreover, link breaks during communication between nodes also happen often due to a number of reasons, chief among them is node mobility. When these failures happen, communication between nodes will be disrupted.

If a mobile node fails to deliver the location update message to the query object when a SP is expired, the query result could become stale. If a query broadcast message from a query object does not get to all mobile objects in the broadcast circle, the initial result as well as subsequent results of the query are not guaranteed to be accurate. Keeping the query results accurate at all times under various kinds of failures is not trivial. In some
situations, such as network partition, communication between nodes in two partitions is simply impossible. Therefore, a best-effort approach is adopted when handling reliability issues for ExtRange. In some cases, it is only possible to mark the query results as stale if the problem is detectable, but not fixable.

To cope with dropped or non-delivered location update messages from mobile objects inside the broadcast circle, a query object can identify stale query results as follows. The query object maintains an additional data structure, called Broadcast Circle Table (BCT) to store information of mobile objects inside the broadcast circle. The BCT is updated when mobile objects report to query object in the initial step (section 4.2.3) and refreshing step (section 4.2.5). The BCT has the following schema: \(< qid, oid, rt, loc, ESP >\), where \(qid\) is the id of a query issued by the query object, \(oid\) is the id of a mobile object inside the broadcast circle, \(rt\) is the reporting time of location update, \(loc\) is the location of the mobile object at \(rt\), and \(ESP\) is the estimated safe period. Using the formula in Eqn. Eq. 4.1, the query object can also estimate the \(SP\) of each mobile object in the BCT, and save the estimation in \(ESP\). When the \(ESP\) of a mobile object is expired, and the query object hasn’t received the location update from the mobile object yet, the query result is marked as stale.

Note that the proposed detection method may conflict with the optimization technique in section 4.2.8.1. Mobile nodes using the optimization technique in section 4.2.8.1 knowingly delay their location update messages because they are certain the query result will not be affected; however, the detection technique in this section still marks the query result as stale.

When a query object stops functioning, upon its return the query object immediately sends a query broadcast message to the broadcast circle. While the query object can infer the membership of some mobile objects with respect to the query result set, for some other mobile objects it is not possible to infer their membership with absolute certainty. Therefore, the query result during the down-time period has to be marked as stale.
4.3 Performance Evaluation of ExtRange

A simulation-based study is carried out to evaluate the performance of ExtRange in realistic settings. Four different performance metrics are considered, including query result accuracy, average number of location updates processed by all query objects per second, average number of messages sent in the network per second, and average number of queries monitored by each mobile object. The first metric measures the accuracy of the query results, while the next three metrics estimate the overheads in ExtRange. A sensitivity analysis study is also done with respect to certain workload factors and system design parameters such as number of mobile objects, number of queries, maximum speed of mobile nodes, and the value of extended range.

A discreet event simulator in C++ for ExtRange is developed. To gauge the bottom line performance of ExtRange, the simulator only simulates the basic ExtRange technique without considering any optimization features as proposed in section 4.2.8. Mobile objects move according to the Random Waypoint model [4]. Mobile nodes are picked randomly to be query nodes. All queries are issued at the beginning of the simulation and last the entire simulation period. The simulator uses the distance-based broadcasting technique [55] as the broadcast protocol, and a modification of the Location-Aided-Routing technique (LAR-1) [38] as the unicast protocol. For the broadcast and unicast protocols, the simulator does not simulate the entire network protocol stack, but only simulates multi-hop message relaying based on proximity between nodes. While detailed simulation of the network protocol stack certainly resembles realistic scenarios very closely, it is unnecessary for the simulation objectives and also too time-consuming in many simulation experiments with large number of nodes and long simulated time. Each simulation setting is run 100 times with different seed numbers, and the reported result is the average value of these 100 runs. Each simulation run lasts for 30 simulated minutes. Table 4.1 summarizes the parameters used in the simulation. Unless noted otherwise, the simulation uses default values for various parameters as reported in
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Variation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation Time (minutes)</td>
<td>30</td>
<td>N/A</td>
</tr>
<tr>
<td>Operating Area (mxm)</td>
<td>3000x3000</td>
<td>N/A</td>
</tr>
<tr>
<td>Number of Mobile Nodes</td>
<td>400</td>
<td>300-700</td>
</tr>
<tr>
<td>Number of queries</td>
<td>30</td>
<td>10-50</td>
</tr>
<tr>
<td>Query range (m)</td>
<td>400</td>
<td>320-480</td>
</tr>
<tr>
<td>$\epsilon$ (as a fraction of query range)</td>
<td>0.2</td>
<td>0.2-1.0</td>
</tr>
<tr>
<td>Random access delay (ms)</td>
<td>5</td>
<td>N/A</td>
</tr>
<tr>
<td>Distance threshold (m)</td>
<td>200</td>
<td>N/A</td>
</tr>
<tr>
<td>Transmission Range (m)</td>
<td>250</td>
<td>N/A</td>
</tr>
<tr>
<td>Max speed (m/s)</td>
<td>5</td>
<td>3-10</td>
</tr>
</tbody>
</table>

Table 4.1. Two parameters, random access delay and distance threshold, are used in the distance-based broadcasting protocol.

### 4.3.1 Result Accuracy

In section 4.2, it has been proven that the query results in ExtRange are always accurate when the communication is perfect, i.e. every message is relayed to the intended target without delay. In reality, communication in a MP2P network is far from being perfect; an imperfect communication mechanism leads to less than perfect result accuracy for ExtRange.

First, the precision $p$ and recall $r$ for a query are defined as below:

\[ p = \frac{|C \cap A|}{|A|}, r = \frac{|C \cap A|}{|C|} \]

where $C$ and $A$ are the correct query result and actual query result, respectively. The correct result $C$ is determined by an all-knowing object (god object) inside the simulator. When the denominator in the above formulas is 0, the corresponding precision or recall is assigned the value of 1 if the other result set is also empty, or the value of 0 if the other result set is not empty.
The accuracy of a query result is then defined as follows:

\[ \text{accuracy} = \frac{2 \times p \times r}{p + r} \]

Note that in the information extraction community, our defined result accuracy is also known as the F-measure \([37]\), which is the weighted harmonic mean of the precision and recall. The value of result accuracy ranges from 0 to 1, with 1 being completely accurate.

Two main factors that influence the message delivery mechanism in ExtRange are studied, namely number of nodes and node speeds. Fig. 4.6 plots the result accuracy as a function of node speed for different number of nodes. High node density provides better connectivity in the MP2P network, and highly connected network is expected to provide better result accuracy for ExtRange, as almost all messages are delivered to the intended destinations. This is observed from Fig. 4.6 as it shows that when the number of nodes is no less than 500, the result accuracy is consistently high (at or above 0.9). Even with smaller number of nodes, the result accuracy is no worse than 0.8. Another interesting observation is that result accuracy is not affected much by node mobility. A plausible explanation is that node mobility causes communication disruption by breaking communication links in a communication session, especially for long-lasting communication session. However, each communication session in ExtRange is very short, which effectively minimizes the effect of node mobility on message delivery and result accuracy.

### 4.3.2 Computation Load at Query Objects

This section discusses the effects of several parameters on computation load at query objects. Computation load at query objects is measured by the average number of location updates processed by all query objects per second. Fig. 4.7 plots the number of location updates as a function of node speed for different number of nodes. It is expected that higher node speed increases the number of location updates, as the safe period for each query becomes shorter.
This is observed from Fig. 4.7 as it shows that the number of location updates increases 2-3 times when node speed increases from 3 (m/s) to 10 (m/s). When the number of nodes is bigger, there are more nodes in the broadcast circle of each query. As a result, there will be more location updates. This is also observed from Fig. 4.7 as it shows that the number of location updates is proportional to the number of nodes.

For a given number of queries, Fig. 4.8 studies the effect of $\epsilon$ on the number of location updates. It is observed that the number of location updates increases proportionally to the number of queries. One interesting observation is that the number of location updates is reduced by 30% when $\epsilon$ is increased from 0.05 to 0.2. Increasing $\epsilon$ beyond 0.2 does not appear to lower the number of location updates though. This observation indicates that there is an operating point for $\epsilon$ that the overheads for ExtRange will be smallest. The implication of $\epsilon$ on system overheads will be visited again in section 4.3.3.

### 4.3.3 Messaging Cost

This section studies the effects of several parameters on messaging cost. Messaging cost is measured as the total number of messages sent, including relaying messages, in the network per second. Fig. 4.9 plots the number of messages sent as a function of node speed for different number of nodes. Fig. 4.10 plots the number of messages sent as a function of
Figure 4.9: Effect of node speed on messaging cost.

Figure 4.10: Effect of extended range on messaging cost.

Figure 4.11: Effect of extended range on computation load at mobile objects.

\( \epsilon \) for different number of queries. In section 4.3.2, it has been observed that the number of location updates increases when there is higher node speed, or more number of nodes, or more number of queries. Higher number of location updates leads to higher number of messages sent. Hence, it is expected that higher node speed, or more number of nodes, or more number of queries requires more number of messages sent per second. Fig. 4.9 and Fig. 4.10 confirm this observation. Fig. 4.9 shows that when the number of nodes increases from 300 to 700, the number of messages sent per second is increased from 3-5 times, and when the node speed increases from 3 m/s to 10 m/s, the number of messages sent per second is increased from 3-4 times. Fig. 4.10 also shows that the number of messages sent is proportional to the number of queries. One interesting observation is that even though increasing the extended range does not mean more location updates as shown in section 4.3.2, Fig. 4.10 shows that bigger extended range indeed increases the messaging cost. The reason is that even with the same number of location updates, bigger extended range needs more number of messages to relay the location updates to the query objects. Combining the observations from this section and section 4.3.2, values in the range \([0.15, 0.3]\) are ideal for \( \epsilon \) with respect to the number of queries ranging from 10 to 50, because at these values of \( \epsilon \) both the computation load at query objects and messaging cost are kept low.
4.3.4 Computation Load at Mobile Objects

This section discusses the effects of extended range and number of queries on computation load at each mobile object. The computation load at each mobile object is measured as the average number of queries each mobile node monitors during the simulation. Fig. 4.11 plots the number of monitored queries as a function of \( \epsilon \) for different number of queries. The first observation is that when the number of queries is increased, the number of queries each mobile node has to monitor also increases. Fig. 4.11 also shows that when \( \epsilon \) increases from 0.1 to 1.0 of the query range, the number of queries monitored by each node is doubled. The reason for the effect of \( \epsilon \) is when the extended range is increased, more nodes are required to monitor a given query; hence, the number of queries monitored by each node is also increased.

4.3.5 Comparing ExtRange against the Baseline Technique

This section compares ExtRange against a competitive baseline technique. ExtRange is not compared against the existing centralized server-based solutions as they were designed for a different environment. The baseline technique being compared with ExtRange in this simulation study is also a distributed technique designed for MP2P networks. The baseline technique works as follows. For each query it issues, the query object rebroadcasts the query in every second. The broadcast circle has a radius equal to the query range. When receiving the query broadcast message, mobile nodes inside the broadcast circle reply to the query object with their current location information.

The two techniques are compared in terms of result accuracy, computation load at query objects, and messaging cost. For each performance metric, a sensitivity study is done on three different parameters, including node speed, number of nodes, and number of queries. Under all settings, ExtRange delivers higher result accuracy (Fig. 4.12) and significantly lower overheads (Fig. 4.13 and Fig. 4.14) than the baseline technique does. Like ExtRange,
result accuracy of the baseline technique is also affected by the imperfect communication mechanism in the network. However, unlike ExtRange, even with perfect communication, the baseline technique cannot guarantee perfect result accuracy, because the location update method in the baseline technique is unaware of the queries. The higher the location updating rate is, the more accurate the query results in the baseline technique will be; however, to ensure perfect result accuracy, the location updating rate in the baseline technique has to be, theoretically, infinity. This explains why even with high updating rate (1 per second), the result accuracy for the baseline technique is still lower than that of ExtRange. Fig. 4.13 shows that the computation load on query objects in the baseline technique is 2-6 times higher than that in ExtRange. Similarly, Fig. 4.14 shows that messaging cost in ExtRange is also consistently lower than that in the baseline technique.

Figure 4.12: Result accuracy for ExtRange and Baseline.

Figure 4.13: Computation load on query objects for ExtRange and Baseline.
Figure 4.14: Messaging cost for ExtRange and Baseline.

4.4 Related Works

Table 4.2 categorizes existing works on continuous range queries in terms of 1) the distance function, whether it is Euclidean-based or network-based distance, 2) whether the data objects and queries are stationary or moving, 3) mobility pattern the system assumes, whether it is linear movement or random movement, 4) whether the system architecture is centralized, hybrid or MP2P, 5) how the system defines and addresses scalability issues, which usually include the two predominant costs: wireless communication cost for location update and evaluation cost for query processing. For the column Mobility of Queries and Objects, S is for stationary, M is for moving, Q is for query, and O is for object.

The concept of safe period used in ExtRange has been used before [44, 8]. The three closest works to ExtRange are [17, 34, 32]. Both [17] and [34] consider continuous moving range queries, but they are different in the distance function with the former using Euclidean distance while the latter using network distance. Both of them assume a hybrid system architecture where a centralized server is still utilized, whereas ExtRange considers a MP2P network without the existence of any centralized server. Like ExtRange, the authors in [32] also assume MP2P networks; however, the paper considers continuous \(kN.N\) queries where the query objects are stationary, while ExtRange supports continuous moving range queries with both query objects and mobile objects moving.
Table 4.2: Comparison of existing techniques for continuous range queries

<table>
<thead>
<tr>
<th>Technique Name</th>
<th>Distance Function</th>
<th>Mobility of Queries and Objects</th>
<th>Mobility Pattern</th>
<th>System Architecture</th>
<th>Scalability Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>QIndex [44]</td>
<td>Euclidean</td>
<td>SQMO</td>
<td>Random</td>
<td>Centralized</td>
<td>Processing Cost</td>
</tr>
<tr>
<td>MQM [8]</td>
<td>Euclidean</td>
<td>SQMO</td>
<td>Random</td>
<td>Hybrid</td>
<td>Both</td>
</tr>
<tr>
<td>MobiEyes [17]</td>
<td>Euclidean</td>
<td>MQMO</td>
<td>Random</td>
<td>Hybrid</td>
<td>Both</td>
</tr>
<tr>
<td>SINA [39]</td>
<td>Euclidean</td>
<td></td>
<td>Random</td>
<td>Centralized</td>
<td>Processing Cost</td>
</tr>
<tr>
<td>HXL-Generic [24]</td>
<td>Euclidean</td>
<td>SQMO</td>
<td>Random</td>
<td>Hybrid</td>
<td>Both</td>
</tr>
<tr>
<td>DRQ [34]</td>
<td>Network</td>
<td>MQMO</td>
<td>Random</td>
<td>Hybrid</td>
<td>Both</td>
</tr>
<tr>
<td>ExtRange</td>
<td>Euclidean</td>
<td>MQMO</td>
<td>Random</td>
<td>MP2P</td>
<td>Both</td>
</tr>
</tbody>
</table>

4.5 Summary

The ability to be able to support continuous spatial queries in MP2P networks becomes vital when a fixed network infrastructure is not available or not cost-effective. This chapter takes an important step forward by providing a solution for a useful type of queries, continuous moving range queries, in MP2P networks. The proposed technique ExtRange contains two key ideas, extended range and safe period. Through analytical study and detailed simulation study, it has been shown that ExtRange is a scalable distributed solution to support continuous moving range queries in MP2P networks. ExtRange is also shown to outperform a competitive baseline distributed technique in MP2P networks.
Search and delivery techniques are part of the core competencies from which P2P applications can be built upon. This dissertation presents two video delivery techniques and one spatial range query technique in P2P-based applications with a focus on system scalability.

The first technique, P2VoD, is especially suitable for delivering video-on-demand over the Internet under the following assumptions: 1) IP Multicast is not available, 2) peers do not cache the entire video, and 3) each peer has sufficient inbound and outbound bandwidth for at least one video stream. Due to the last assumption on the outbound bandwidth capacity of each peer, each receiver in P2VoD only needs to receive the video stream from one sender; hence, P2VoD is an example of the one-sender approach. Other researchers also consider the multi-sender approach, when a receiver receives multiple streams from multiple senders for one video (e.g. [57]). The multi-sender approach assumes the following conditions: 1) peers have larger inbound bandwidth than outbound bandwidth, and 2) peers can cache the entire video. One possible research direction is to design a framework to combine the one-sender and multi-sender approaches to support a broader client base and different networking conditions.

The second technique, PatchPeer, is designed for delivering video-on-demand over wireless networks, when there exists a hybrid wireless network and node mobility is not too high. This research can be expanded in several ways. First, the patch streams can be provided by either neighboring peers or infostations, if the latter is available. Second, if multiple-channel wireless communication can be assumed, the contention-aware admission control in section 3.2.2 is not needed as the admission control is used to cope with the single-channel wireless networks. Furthermore, a peer can receive multiple streams simultaneously, each stream on a different channel, which may lead to a different design to utilize peer’s bandwidth.
The third technique, ExtRange, is useful when no fixed wireless communication infrastructure is available. One direction to expand this research is to also support other types of queries, such as k-nearest-neighbor queries. Another direction is to leverage available location services. Since location information of mobile nodes are demanded by many location-based applications, location services have been proposed for mobile P2P networks to manage location information, which can be shared by different location-based applications. Adapting ExtRange to use one such existing location service will eliminate the need of mobile nodes sending location updates directly to query nodes.

Besides the research directions mentioned in the above paragraphs, this dissertation can also be expanded into addressing other core competencies for P2P applications such as collaboration enforcement. The level of peer collaboration has a direct consequence on the scalability of a P2P application; hence, effective collaboration enforcement is one important factor deciding the success or failure of a P2P application. Recent collaboration enforcement techniques have applied economic theories with encouraging results. Clever application of existing economic theories and extensive evaluation and analysis of existing P2P applications will likely lead to more efficient collaboration enforcement techniques.
REFERENCES


