2010

A Robust Wireless Mesh Access Environment For Mobile Video Users

Fei Xie

University of Central Florida

A ROBUST WIRELESS MESH ACCESS ENVIRONMENT FOR MOBILE VIDEO USERS

by

FEI XIE
M.S., University of Central Florida, 2008
B.S., University of Science and Technology of China, 2004

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
2010

Major Professor: Kien A. Hua

ABSTRACT

The rapid advances in networking technology have enabled large-scale deployments of online video streaming services in today's Internet. In particular, wireless Internet access technology has been one of the most transforming and empowering technologies in recent years. We have witnessed a dramatic increase in the number of mobile users who access online video services through wireless access networks, such as wireless mesh networks and 3G cellular networks. Unlike in wired environment, using a dedicated stream for each video service request is very expensive for wireless networks. This simple strategy also has limited scalability when popular content is demanded by a large number of users. It is desirable to have a robust wireless access environment that can sustain a sudden spurt of interest for certain videos due to, say a current event. Moreover, due to the mobility of the video users, smooth streaming performance during the handoff is a key requirement to the robustness of the wireless access networks for mobile video users. In this dissertation, the author focuses on the robustness of the wireless mesh access (WMA) environment for mobile video users. Novel video sharing techniques are proposed to reduce the burden of video streaming in different WMA environments. The author proposes a cross-layer framework for scalable Video-on-Demand (VOD) service in multi-hop WiMax mesh networks. The author also studies the optimization problems for video multicast in a general wireless mesh networks. The WMA environment is modeled as a connected graph with a video source in one of the nodes and the video requests randomly generated from other nodes in the graph. The optimal video multicast problem in such environment is formulated as two sub-problems. The proposed solutions of the sub-problems are justified using simulation and numerical study. In the case of online video streaming, online video server does not cooperate with the access networks. In this case, the centralized data sharing technique fails since they assume the cooperation between the video server and the network. To tackle this problem, a novel distributed video sharing technique called Dynamic Stream Merging (DSM) is proposed. DSM improves the robustness of the WMA environment without the cooperation from the online video server. It optimizes the per link sharing performance with small time complexity and message complexity. The performance of

DSM has been studied using simulations in Network Simulator 2 (NS2) as well as real experiments in a wireless mesh testbed. The Mobile YouTube website (http://m.youtube.com) is used as the online video website in the experiment. Last but not the least; a cross-layer scheme is proposed to avoid the degradation on the video quality during the handoff in the WMA environment. Novel video quality related triggers and the routing metrics at the mesh routers are utilized in the handoff decision making process. A redirection scheme is also proposed to eliminate packet loss caused by the handoff.

ACKNOWLEDGMENTS

This dissertation would not have been possible without the guidance of my committee members, helps from my friends, and support from my parents and wife.

My deepest gratitude goes to my advisor, Dr. Kien A. Hua, for his stimulating advice, encouragement, and patience during my research and study. It is pleasure to thank Dr. Mostafa A. Bassiouni and Dr. Cliff C. Zou for their guidance on my research of network communications. I would like to thank Dr. Sheau-Dong Lang for his guidance on the algorithms. Special thanks go to Dr. J. Michael Moshell who adjusted his tight schedule to attend my defense.

I am grateful for the help and suggestion from my friends during my doctoral study. I would like to thank Dr. Ning Jiang, Dr. Yao H. Ho, Dr. Wenjing Wang, and Ai H. Ho whom I have collaborated with in the field of wireless networks. I would also like to thank Dr. Tai T. Do for the constructive discussions on video streaming. I would like to express my gratitude to Richard Zhou, Steven Nichols, and William Cecil for their contributions to the prototype system included in this dissertation. Many thanks go to the following members of the Data System Group for their help and support. They are Dr. Danzhou Liu, Dr. Hao Chang, Ning Yu, Fuyu Liu, Rui Peng, and Sirikunya Nilpanich.

I would like to thank my parents for their invaluable life guidance which has the positive influence on anything that I have achieved so far.

I would like to give my sincere gratitude to my wife, Wenjing Lai, who has dedicated her full support to my academic study since the first day I entered the PhD program.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

A picture is worth a thousand words, and the addition of sound and motion can breathe life into a picture. As a result, video data has become an inseparable part of many applications with the rapid advances in networking technology. Specifically, online video service has become one of the killer applications on the Internet. Video sharing websites such as Youtube [1] and Hulu [2] have attracted millions of daily access. Emerging online streaming applications such as web conferencing (e.g., the webinar) and IPTV services are also good examples demonstrating the tight coupling between the advanced video streaming techniques and networking techniques. In recent years, we have witnessed a dramatic increase in the number of users who access online video service through access networks employing different wireless technologies, such as wireless mesh networking [3], 3G access technologies (e.g., [4], [5]). Due to the wireless feature of these access networks, many users in these environments are mobile users.

In a typical video streaming application, a video server launches a dedicated stream for each video request. The data rate of today's online video stream is usually up to hundreds kilo bit per second (kbps). For example, the data rate of the video stream (resolution 176×144 pixels, encoded by h.263) from Mobile Youtube [6] is about 400 kbps. Using a dedicated stream for each user is very expensive in the wireless environment. This simple strategy also had limited scalability when popular content is demanded by a large number of users. It is desirable that a robust wireless access environment should be able to sustain a sudden spurt of interest for certain videos due to, say current event. For example, the events such as the earthquake in Haiti and the death of Michael Jackson incurred a period of great interest in the related videos worldwide. When an event like this arises, we do not want the substantial increase in demand for the related videos to significantly impact the normal access to other regular videos. Addressing this problem is also highly desirable for applications whose access pattern follows the so called 80:20 rule. That is, 80% of the demand is for 20% of the videos. For instance, most of the news video accesses are for recent big events, such as sports, scandal, and disaster.

There are generally two approaches to addressing the aforementioned issues. One is to reduce the data rate of each video stream by adopting advanced video coding technique such as the H.264/AVC [7]. As the per-stream data rate is reduced, the overall demand in the access network declines. Moreover, the scalable video coding [8] technique provides different levels of video quality which corresponds to different data rate. Heterogeneous video users are able to choose the affordable level of quality based on various bandwidth capacities of the access networks. Although this strategy is very promising, it could incur more power consumption at the video client due to the execution of more complicated decoding algorithms. Another way to reduce stress on the network is through data sharing. That is, the users are allowed to share video streams to conserve network bandwidth. In practice either advanced coding or data sharing techniques, or both can be used. We focus on data sharing in this paper.

Video streaming applications can be classified into two categories, namely live video streaming (LVS) and video-on-demand (VOD). For VOD, users can request a video at any time and they generally playback the entire video from the beginning. Examples of VOD application include movies on demand and video services at various social networks. In contrast, users of a live video streaming service are not interested in playing back the entire video from the beginning. Examples include live streaming of a sport event and video conferencing. The users, in this case, want to see what is currently happening, not actions occurred some time ago. In this environment, all the users are effectively always at the same play point in the video, with negligible differences due to network delay. It is easier to facilitate data sharing among LVS sessions than VOD sessions. This is because the LVS sessions always require the same content of the video at any given time. Video sharing in LVS, therefore, can be realized by broadcasting the live video data to all the users. Video sharing is more challenging for VOD applications due to the asynchronous nature of VOD sessions. That is, the users are generally at different play points in a given video.

It is easier to facilitate data sharing among LVS sessions than VOD sessions. This is because the LVS sessions always require the same content of the video at any given time. Video sharing in LVS, therefore, can be realized by broadcasting the live video data to all the users. Video sharing is more

challenging for VOD applications due to the asynchronous nature of VOD sessions. That is, the users are generally at different play points in a given video. Techniques such as Periodic Broadcast ([9], [10]), Overlay Multicast ([11], [12], [13]), and Peer-to-Peer (P2P) Streaming ([14], [15], [16]) can be used to facilitate sharing of video streams in different VOD environments. Specifically, Periodic Broadcast is designed for multicast-capable networks such as local area networks or cable TV networks; Overlay Multicast targets general IP networks; while P2P Streaming is amenable to Internet-scale applications. Recent research has also revealed that network coding techniques can be adopted for video streaming applications ([17], [18]). Since LVS session is a special case of VOD, techniques such as Overlay Multicast and P2P Streaming can also be applied for LVS applications.

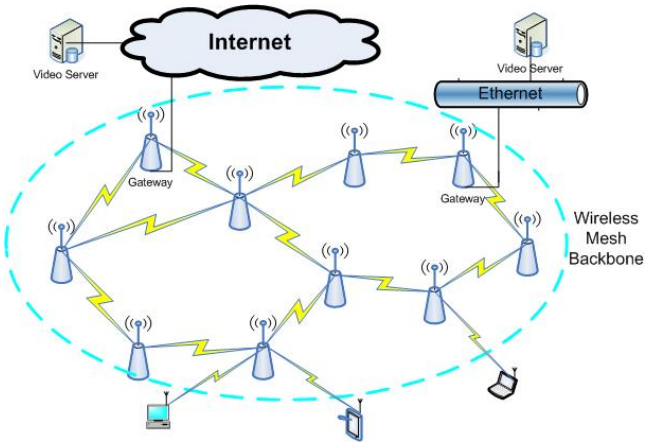


Figure 1. Video streaming in the wireless mesh access environment

In this dissertation, we are interested in exploiting the possibility of video sharing in wireless mesh access networks. By taking advantage of the broadband wireless technologies, wireless mesh networks (WMN), a special case of mobile *ad hoc networks* (MANET) with mesh topology, have been used as an edge technology to provide broadband Internet access in residential, business, and even city-wide networks. In a WMN, the wireless mesh routers form a mesh-like wireless backbone network that allows end users to access the services in the Internet or Local Area Network (LAN) through mesh gateways. For convenience sake, we refer to such a network environment as a Wireless Mesh Access (WMA) environment. Figure 1 illustrates the video streaming in the WMA environment.

Unlike the mobile routers in MANET, wireless mesh routers are deployed in fixed location. However, since the end users have mobility, mesh routers have to support the end user roaming and provide seamless handoff. Existing implementations (e.g., [19], [20]) of WMN aim to present the network as a conventional WLAN to the end users. The underlying mesh networking technology is transparent to the end users. This strategy facilitates the easy adoption of WMN since the end users do not need to install extra software to join the network. Besides user roaming, problems such as routing, rate adaptation, as well as power control in the wireless mesh backhaul have drawn attentions of the research communities in the past few years.

Many researches ([21], [22], [23], [24]) have been conducted to improve the quality of unicast video streams in WMN by taking the video coding and the network parameters into account. Some of them ([21], [22]) utilize the Rate-Distortion model to capture the relationship between video quality and the underlying network conditions, such as delay and packet loss. These works are complementing to the video sharing techniques as shown in [25], where a joint rate-distortion optimized video multicast scheme is discussed. Nevertheless, it has not been well studied on leveraging the video sharing technique to improve the robustness of the WMA environment. On the other hand, video sharing for VOD has been thoroughly studied in the wired environment (e.g., [9], [14], [26], [27]). However these schemes are not designed to take advantage of the broadcast capability of wireless transmission. Furthermore, due to the large scale of the Internet, these schemes adopt the high level overlay network and avoid the complexity associated with the physical topology of the Internet. In contrast, it is possible to take into account the near stationary topology of WMN in order to leverage the full potential of the wireless networks. Handling the dynamics of the wireless links and user mobility brings another great challenge unique to WMN. One of the important design issues for video sharing is that the content provider (e.g., the video server side) has certain cooperation with the media (e.g., the access networks) where sharing happens. Specifically, in the case of video sharing in the WMA environment, the conventional video sharing techniques require the video servers control over the WMN. This kind of cooperation might not be feasible for online video service.

In this dissertation, several novel techniques are proposed to tackle the aforementioned problems. The remaining part of the dissertation is organized as follows. The related works are discussed in Chapter 2. A cross-layer framework for scalable VOD service in multi-hop WiMax relay networks is proposed in Chapter 3. The optimization problems for video multicast in a general wireless mesh networks are studied in Chapter 4. In Chapter 5, a novel distributed video sharing technique called Dynamic Stream Merging (DSM) is presented. The implementation of DSM in a wireless mesh testbed is also covered in this chapter. A cross layer handoff scheme for mobile video users is studied in Chapter 6. The concluding remarks are presented in Chapter 7.

2. RELATED WORKS

Our literature survey indicates that existing related works can be classified into three categories, namely video sharing in wired networks, video streaming in WMN, and multicast routing in WMN.

2.1 Video Streaming in Wired Networks

A video server typically can sustain only a very limited number of concurrent video streams. This problem, known as *server or network–I/O bottleneck*, limits the scalability of video services. Several periodic broadcast schemes have been proposed to address this problem (e.g., [9], [10]). In this approach, a video is fragmented into a number of segments, each periodically broadcast on a dedicated channel. It is up to the client software to tune to these channels to pick up the required segments just in time for the video playback. This strategy is highly scalable as it can serve a very large community of users requesting the same video with minimal server bandwidth. A limitation of this approach is that it requires instance occupation of the network bandwidth. To reduce this cost, one can leverage IP multicast to allow multiple clients to share a server stream ([26], [28]). Unfortunately, the deployment of such a facility beyond local area networks has been shown to be difficult. Alternatively, end hosts can implement multicast service at the application layer, assuming only IP unicast at the network layer, therefore called application-layer multicast (ALM). Existing ALM protocols can be classified into two categories: the infrastructure-based (IB) approach ([13], [29]) and the P2P approach ([14], [15], [16], [27], [30]). In the IB approach, a set of dedicated machines called overlay nodes act as software routers with multicast functionalities. These overlay nodes form a certain overlay topology (e.g., tree) to facilitate the multicast in the network. Due to the high cost of deployment and maintenance of the IB approach, the P2P approach, was proposed to simplify the implementation of ALM and has been widely used in today's Internet. Examples include *PPLive* [31], *PPStream* [32], and *Coolstreaming* [33],. These techniques, however, cannot take full advantage of the network topology; nor are they well suited to leverage the broadcast nature of wireless

transmission. Moreover, they are not designed for a mobile-user environment. Recent research has also revealed that network coding techniques can be adopted in the P2P streaming applications.

2.2 Video Streaming in WMN

Besides video sharing, another class of research focuses on using cross-layer design approach to improve the quality of the video streaming in the wireless environment. The works in this category are complementing to the video sharing. Under the cross-layer design framework, problems with routing, rate control, and scheduling are solved by considering together the video coding schemes (e.g., scalable video coding and multiple descriptions video coding) and the parameters of the wireless network. In these works (e.g, [21], [22], [23], [34]), the rate-distortion model is usually used to capture the relationship between the quality of the video and the quality of the network connectivity. In [21], the authors encode the video into multiple streams and distribute them over multiple paths in an ad hoc network. This scheme can take advantage of the path diversity in the mesh networks. Authors of [22] propose a multi-source multi-path video streaming system. Their assumption is that the client may keep a local copy of the video. The video files have multiple copies in different clients. A video session can get data from multiple sources. Authors of [23] solve the joint routing and rate allocation problem for multiple video streams in ad hoc networks. Other works, such as [24] and [35], try to improve the video streaming quality in a one hop scenario. Recent work [36] has studied real-time video multicast in cognitive radio networks.

2.3 Multicast Routing in WMN

Multicast routing can be leveraged to efficiently share video data in WMN. This category of research has been studied from both the theory and protocol aspects. Finding a multicast tree with a minimum sum of edge weights in a random weighted graph is the famous Steiner Tree problem [37]. However, the authors of [38] have recently pointed out that the optimal multicast tree in wireless mesh networks is not a Steiner tree. The actual number of transmissions required to multicast a packet is smaller than the number of links of the tree due to the broadcast nature of wireless transmissions. Broadcast, a

special case of multicast, has been studied in many works. For example, the authors of [39] proved that finding a set of broadcasting nodes with minimum energy consumption is NP Complete in both the general graph and unit disk graph. However, it is still interesting and important to study the complexity of general multicast in WMN.

For large scale WMN such as city-wide WMN, distributed multicast routing protocols are necessary. Many multicast routing protocols have been proposed for MANET [40]. A topology-based approach (e.g., [41], [42], [43], [44], [45], [46]) aims to establish an efficient multicast tree or mesh topology to facilitate the multicast. Due to mobility in MANET, a topology-based approach incurs control overhead associated with the maintenance of the topology overtime. To address this problem, stateless multicast (e.g., DDM [47]) is proposed, wherein a source explicitly mentions the list of destinations in the packet header. Many multicast routing protocols (e.g., [41], [42], [48]) leverage WMA by using link layer multicast. In [49], the authors point out that the routing metrics in multicast routing protocols are substantially different from those used in unicast routing protocols due to the use of WMA. However, in many wireless technologies, the link layer broadcast is less reliable than link layer unicast. For instance, 802.11 MAC unicast data transmission involves RTS/CTS/ACK to insure successful transmission; but there is no such guarantee in 802.11 MAC broadcast. The performance of these multicast routing protocols for video streaming and VOD applications is unknown.

3. VOD IN WIMAX-BASED WMA

In this chapter, a cross-layer framework is proposed to facilitate the VOD service in multi-hop WiMax mesh networks. A joint solution of admission control and channel scheduling scheme is proposed to guarantee that the required data rate is achieved for video streams. This feature is crucial for multimedia streaming applications. An efficient and light-weight multicast routing technique is also proposed to minimize the bandwidth cost for a video user to join a multicast tree. Furthermore, the Patching technique is adopted in the application layer to improve the capacity of the video server. Overall, the quality of the VOD service is dramatically improved with the help of the efficient cooperation between the techniques proposed in different layers of the network. Simulation study shows that with the proposed approach, true VOD in WiMax mesh networks can be achieved under high video request arrival rate.

3.1 Introduction

Rapidly growing bandwidth demand for multimedia streaming services has spurred the development of new broadband access technologies over recent years. The emerging wireless mesh networks are expected to deliver high data rate wireless connectivity and provide significant extension of coverage for next generation ubiquitous multimedia streaming service. The IEEE 802.16 standard [50], also commonly known as WiMax, is a standards-based technology that enables the delivery of last mile wireless broadband access as an alternative to cable and DSL. Compared to wired solutions, WiMax provides more ubiquitous access with lower deployment and maintenance costs. Besides the advantage of lower deployment expense, WiMax also offers ample bandwidth resource which enables future high-speed data and telecommunication services.

We are interested in supporting VOD service in WiMax-based WMN. VOD is a critical technology for many important multimedia applications, such as home entertainment, digital video libraries, distance learning, company training, news on demand, electronic commerce, to name just a few. A typical VOD service allows remote users to playback any video from a large collection of videos stored on one or

more servers. In response to a service request, a video server delivers the video to the user in an isochronous video stream.

In this work, we try to improve different layers of the network to favor the VOD applications. More precisely, we introduce a novel joint channel allocation and admission control scheme in MAC layer, a light-weight multicast routing in the network layer and application layer VOD technique. In the rest of this section, we first elaborate some preliminary knowledge of the WiMax networks and then briefly introduce the challenging parts of channel scheduling and routing for video streams in a WiMax based mesh network. We highlight our contributions at the end of this section.

A WiMax network consists of a Base Station (BS) and multiple Subscriber Stations (SS). The mesh BS acts as the gateway for SSs to access external networks. Each SS is an access point which aggregates data traffics of end users using other protocols, such as 802.3 or 802.11. A WiMax network can operate under two modes. The first mode is the Point-to-Multipoint (PMP) mode. In PMP, all the SSs directly connect to the BS through a single-hop wireless link. In other words, the PMP mode requires that each SS be within the *Line of Sight* (LOS) of a BS. The second mode is the mesh mode, in which a SS can communicate with either the BS or other SSs through multi-hop routes. The mesh mode extends the coverage of the network and is able to support non-LOS transmission. The mesh mode is an instance of the wireless backhaul networks. In this paper, we focus on the mesh mode. In the mesh mode, SSs and BSs use scheduling trees for routing. A scheduling tree is a spanning tree built upon the physical topology of the mesh network. The root of a scheduling tree is the mesh BS. A scheduling tree is built and maintained as follows. Active nodes in a mesh network periodically broadcast Mesh Network Configuration (MSH-NCFG) messages. Each MSH-NCFG message contains a Network Descriptor that includes configuration information of the mesh network. A new node searches for active networks by listening to MSH-NCFG messages. From all the possible neighboring nodes that advertise MSH-NCFG messages, the new node chooses one as its sponsor node. Then the new node sends a Mesh Network Entry (MSH-NENT) message with registration information to the mesh BS through its sponsor node. Upon receiving the registration

message, the mesh BS adds the new node as the child of the sponsor node and then broadcasts the updated network configuration to all the SSs in the network. Figure 2 depicts a WiMax-based WMN and the corresponding scheduling tree.

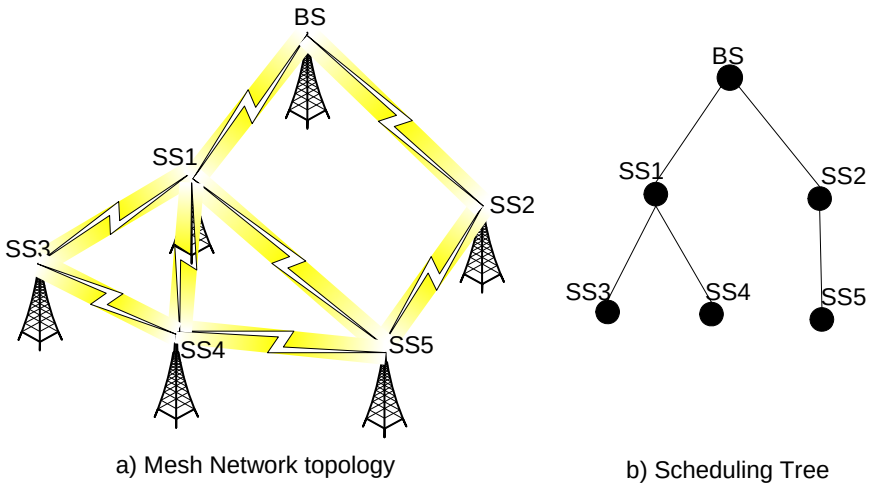


Figure 2. An example of WiMax WMN and its scheduling tree

We consider a typical scenario of providing VOD service in residential or business networks with a wireless backhaul, where video servers are connected to a mesh BS with high speed wired link. Streaming requests are generated from end users of each SS. Each stream requires certain data rate for continuous video playback. Since the link between video servers and the mesh BS is not a bottleneck, the capacity of the server is dominated by the capacity of the wireless mesh network. Therefore, we consider the admission control on the video server and the admission control on the mesh BS as identical problems. In the rest of this paper, we refer to both of them as the admission control problem.

The channel scheduling mechanisms of WiMax are as follows. WiMax uses Time Division Multiple Access (TDMA) for BSs and SSs to access radio channels. A channel is divided into frames. A frame consists of a control subframe and a data subframe. Each frame is further divided into mini-slots for transmission of user data and control message. The data subframe mini-slot allocation is carried in the last control subframe. The mesh mode has two types of channel scheduling schemes: distributed scheduling and centralized scheduling. In distributed scheduling, SSs are peers and they compete for transmission

opportunities based on pseudo-code random election algorithm. In the centralized scheduling, BSs determine the allocation of the minislots dedicated for centralized scheduling among all the stations. The centralized and distributed scheduling algorithms can work simultaneously and independently if both are assigned dedicated mini-slots. More details of the messages and signaling mechanism for transmission scheduling can be found in [50].

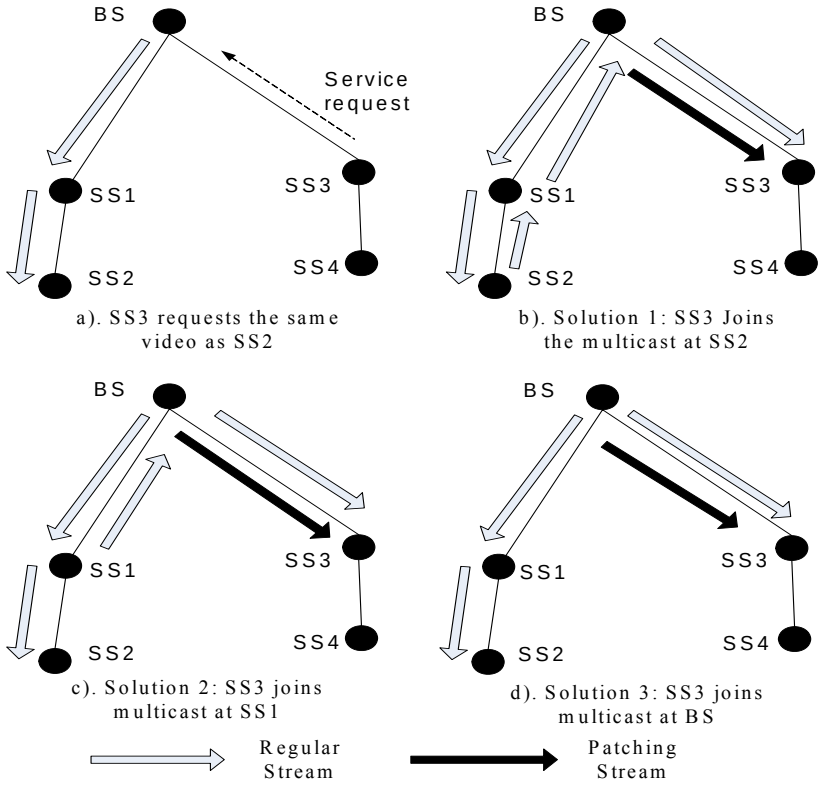


Figure 3. Different ways to join a multicast

Since end users access video servers through the mesh BS, we focus on centralized channel scheduling in BS. An open research problem of the centralized scheduling is that WiMax does not specify how to assign mini-slots to different stations. In this paper, we propose a technique that jointly solves the admission control problem and the channel scheduling problem. Our joint solution fully utilizes the bandwidth resource and can thus guarantee the required data rates of the admitted multimedia streams during the entire streaming period. The proposed algorithm is also compatible with the general centralized scheduling for other types of traffics (e.g., HTTP, FTP and VoIP) in the same network.

The number of simultaneously running video streams in the network is limited by the capacity of the network. We usually use multicast to extend the capacity of the video server. The idea is that different users share the same video stream if they request the same video.

In order for multicast to fully utilize the bandwidth resource, an underlying routing scheme is critical, as illustrated in Figure 3. In Figure 3(a), suppose two requests for a same video are received by SS2 and SS3, respectively. Figure 3(b), Figure 3(c), and Figure 3(d) depict three possible ways to build a multicast tree. Obviously, the third solution is more efficient in terms of overall bandwidth consumption. In this paper, we propose a technique to minimize the bandwidth consumption introduced by forming multicast trees.

Finally, to form a multicast group, the server has to put earlier arrived requests on hold to accommodate requests that arrive later. If users making the earlier request are kept waiting too long, they are likely to renege. Moreover, since many requests are likely to be forced to wait, only near VOD can be achieved. To address this dilemma, we introduced a technique called Patching [26] in our previous work. The basic idea of Patching is as follows. A new service request can exploit an existing multicast by buffering the on-going stream from the multicast while playing a new "catch-up" flow from the server. Once the catch-up flow has been played back to the skew point, it is terminated and the original multicast can be shared. We choose Patching as the application level multicast technique because of its centralized flavor, which can be easily employed on top of the centralized admission control and the channel allocation scheme. There are many existing Patching techniques (e.g., [26], [51], [52]), in this chapter, we only demonstrate how to employ the basic Patching scheme, and demonstrate the potential performance gain.

In summary, our contributions in this work are four parts:

- We propose a solution that jointly solves the admission control and channel allocation problem. The proposed solution guarantees the required data rate of admitted video streams.

- We propose a multicast routing scheme in WiMax mesh networks. The routing scheme builds efficient multicast tree and introduces negligible control overhead.
- We adopt the Patching technique to WiMax mesh networks by utilizing the proposed rate guarantee model and multicast routing scheme. We also use simulation to verify the efficiency of our proposed VOD solutions.
- We propose two bandwidth sharing approaches for patching stream and regular video stream and conduct sophisticated simulations to study their performance

3.2 System Model

We consider a WiMax mesh network that consists of one mesh BS node and N mesh SS nodes. Nodes are labeled as $j = 0, \ldots, N$. Specifically, the BS node is labeled as node 0. There is a link (i, j) between node i and j when they are within the transmission range of each other. The mesh topology can be represented by a graph $\mathcal{G} = \{\mathcal{N}, \mathcal{L}\}$, where $\mathcal{N} = \{0,1, \ldots, N\}$, and $\mathcal{L} = \{1,2, \ldots, L\}$ labels all the links. Assume link l has a bandwidth of w_l bps. We focus on the centralized scheduling of the IEEE 802.16 mesh mode, where a scheduling tree rooted at the mesh BS is constructed for the routing path between each SS and the mesh BS, and the BS acts as the centralized scheduler that determines the transmission and reception of every SS in each minislot. Denote a scheduling tree by $T = \{n_0(h_0, p_0), \ldots, n_N(h_N, p_N)\}$, where h_i denotes the number of hops from node n_i to the mesh BS n_0, and p_i is the parent node of n_i. The mesh BS is indexed by (0, 0). We denote the path from n_i to the mesh BS by P_i. Let $\mathcal{L}T$ be the set of links that belong to the scheduling tree T.

Unlike the wired link, the wireless link has special characteristics. For example, an active link may interfere or conflict with other links using the same channel. There are two types of constraints in wireless mesh networks:

Primary constraint: If each node has a single radio transceiver, due to the half duplex nature of the transceiver, the node cannot transmit and receive simultaneously.

Secondary constraint: Links do not share a common node will interfere with each other, if at least one of their corresponding transmitter or receiver nodes are within the range.

The *secondary constraint* depends on physical layer parameters and capabilities. Note that in wireless mesh networks, we can adopt techniques like *directional* (e.g., beam forming) *antennas* to minimize the interference caused to neighboring links. By carefully planning the locations of BS and SS nodes, interference among neighboring links can be greatly reduced. Another way to mitigate the secondary constraint is to use different channels with orthogonal frequency band or spread spectrum coding all nodes within the two hop neighborhood. In this paper, we only consider the primary constraint for link activation. This means each node can only activate either an incoming or an outgoing link at any time. Let $\mathcal{N}(i)$ denote the set of incoming and outgoing links of node n_i. The required data rate of link l is denoted by r_l, and we call $\boldsymbol{r} = \{r1, \ldots, r_L\}$ as the required *traffic pattern* of the network. The fraction of time that link l needs to be activated is denoted as $f_1 = r_l / w_l$. Let $\boldsymbol{f} = (f_1, \ldots, f_L)$. The following constraints are the necessary condition for a feasible $\boldsymbol{r}$,

$$\sum_{l:l\in\mathcal{N}(i)} f_l \le 1, \ \forall i \in \mathcal{N} \qquad (3.1)$$

It has been shown in [53] that (3.1) is also the sufficient condition for scheduling down link unicast streams from BS to each SS. We further point out that (3.1) is sufficient for any feasible traffic pattern $\boldsymbol{r}$ in a WiMax mesh network with tree topology.

Theorem 3.1 *Assuming that only primary constraints exist in WiMax mesh network,* (3.1) *is the necessary and sufficient condition for a feasible traffic pattern* $\mathbf{r}$.

Proof: The proof of necessary condition is from the definition of primary constraints.

Let $\mathcal{K}$ be the number of minislots in a channel scheduling period. Assume all minislots are dedicated for downlink centralized scheduling. We can choose a large enough $\mathcal{K}$ such that $a_l = \mathcal{K} f_l$ is an integer for every $l \in \mathcal{L}_\Gamma$. We prove the sufficient condition by constructing the multi-graph scheduling tree T_m corresponding to a scheduling tree T. T_m has the same node set as T, with a link $l \in \mathcal{L}_\Gamma$ represented by a_l edges in T_m between the same endpoint nodes. Figure 4 illustrates an example of a multi-graph scheduling tree.

We note that the number of minislots that we need to satisfy the traffic pattern $\boldsymbol{r}$ is the *chromatic index* Γ of T_m, which is the minimum number of colors needed to color the edge in T_m, such that no two edges incident on the same node are assigned the same color. Notice that T_m is a bipartite graph where node i with an odd h_i is in a set and node j with an even h_j belongs to another set. From the graph theory we know that for bipartite graph, $\Gamma = \Delta$, which is the maximum degree of a node. We have $\Delta = \max_{i \in \mathcal{K}} \sum_{l:l \in \mathcal{N}(i)} a_l$. Note that (3.1) implies that $\sum_{l:l \in \mathcal{N}(i)} a_l = \sum_{l:l \in \mathcal{N}(i)} f_l \mathcal{K} \le \mathcal{K}$, from which we have $\Gamma \le \mathcal{K}$. This means we can schedule the traffic pattern $\boldsymbol{r}$ in a scheduling period. Thus (3.1) is a sufficient condition for schedulability of $\boldsymbol{r}$. □

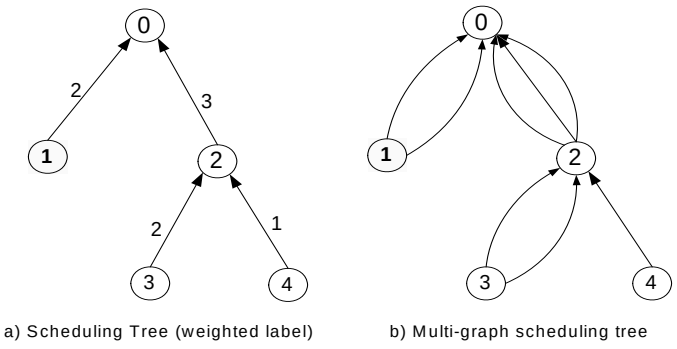


Figure 4. Illustration of the construction of the multi-graph scheduling tree

3.3 Admission Control for Unicast

We first consider the case that all the video streams are unicast streams. In this case, routing is done by the scheduling tree. In order to guarantee the data rate of all admitted streams, the admission control scheme should be jointly considered with the channel scheduling problem.

The mesh BS has to support the profiled data rate of each requested video. Otherwise the video quality will be distorted. The data rates of the video streams are usually characterized as variable bit rate (VBR) [54]. We can analyze the distribution of the data rate r for a particular video stream and request a data rate r^* such that $Prob(r^* \geq r) = 0.9$. This means that if the BS admits the request, it should guarantee the data rate of the stream at least 90% of time.

We define a data rate request of the kth video stream of node n_i as $b_i(k)$ bps, the number of data rate requests of n_i as K_i, and $b_i = \{b_i(k) \mid k = 1, \ldots, K_i\}$. The mesh BS then collects the set of data rate request (i.e., b_i) of each SS node. We define $\mathcal{B} = \{b_i \mid i = 1, \ldots, M\}$. The mesh BS decides to admit or postpone each individual $b_i(k)$. The mesh BS also calculates the transmission schedule and distributes them to all the SSs.

For SS n_i, define x_i as its *admission vector*, which is a binary vector with K_i elements, where the kth element is

$$x_i(k) = \begin{cases} 1, & \text{if stream } k \text{ of node } i \text{ is admitted,} \\ 0, & \text{otherwise.} \end{cases}$$

Let $\boldsymbol{x} = (x_1, \ldots, x_M)$ denote the *admission matrix* of all the SSs. Therefore, f_l (i.e., the fraction of time that link l needs to be activated during a scheduling period) can be calculated as:

$$f_l = (1/w_l) \cdot \sum_{i:l \in Pi} \sum_{k \in Ki} x_i(k) b_i(k) \qquad (3.2)$$

Using (3.2) to substitute the f_l in (3.1), yield,

$$\sum_{l:l\in\mathcal{N}(i)}(1/w_l)\cdot\sum_{i:l\in Pi}\sum_{k\in Ki}x_i(k)b_i(k)\le 1,\ \forall i\in\mathcal{N} \qquad (3.3)$$

Based on Theorem 3.1, the necessary and sufficient condition that $\boldsymbol{x}$ is schedulable is (3.3). Each constrain in (3.3) can be seen as being applied to a particular node in the mesh network. We design the admission control scheme as follows. We store all the waiting requests into a queue called *wqueue*. The request in *wqueue* is severed in the FIFO manner. All the running streams are stored in another queue called *rqueue*. Therefore, $\boldsymbol{x}$ is decided by managing the requests in *wqueue* and *rqueue*. The detail process is described in Algorithm 3.1.

Table 1 Algorithm 3.1

Algorithm 3.1 Joint Admission Control and Channel Scheduling
1. For i = 0 to end of *wqueue*
2. if (accepting *wqueue*[i] violates (3.3))
3. postpone *wqueue*[i]
4. else
5. *rqueue*.add(*wqueue*[i]) // admit request
6. *wqueue*.dequeue(i)
7. increase $\boldsymbol{f}$ //consume bandwidth
8. For i = 0 to end of *rqueue*
9. if(*rqueue[i]* is done)
10. *rqueue*.dequeue(i)
11. reduce $\boldsymbol{f}$ // release bandwidth
12. Generate the schedule based on *rqueue.*

Algorithm 3.1 guarantees that $\boldsymbol{x}$ satisfies all the constraints in (3.3). In line 2, we only need to check if adding a new stream violates the constraints associated to the nodes which are in the path of the corresponding stream. Thus line 2 has *O(logN)* comparisons. Line 7 and 11 increases and reduces the bandwidth by updating the corresponding f_l in $\boldsymbol{f}$, where l belongs to the path of the request stream. In line 12, we generate the schedule by greedy-coloring the edges in the multi-graph scheduling tree constructed based on Theorem 3.1. In practice, since the stream data rate is VBR, at each scheduling period, the instant

data rate of the video stream may be larger or smaller than the required data rate. If the instant data rate is below or equal to the required data rate, it will be fully satisfied. If the instant required data rate is larger than the required data rate, the amount of data rate exceeding the required data rate may not be allocated by the scheduler. We call this amount of data rate as residual data rate of a video stream. The aforementioned scheme is applied to all admitted streams.

It is worth mentioning that there may be other centralized channel scheduling algorithms running simultaneously in the WiMax network, supporting other types of traffics. Algorithm 3.1 is compatible with these algorithms. We can have certain number of dedicated minislots for the VOD application in each station. In this way, Algorithm 3.1 is independent of other types of traffics. Alternatively, several centralized algorithms could share certain amount of minislots in each station. They need to keep each other updated on the bandwidth consumption information. We omit the detail here since it is out of the scope of this work.

3.4 Multicast Routing Scheme

We propose an efficient and lightweight multicast routing scheme that serves as a generic solution for WiMax mesh networks. The scheme is based on top of the scheduling tree. The advantage is that we make use of the existing scheduling tree which is well maintained by the mesh BS. There is no extra maintenance overhead for the proposed multicast routing. Formally speaking, for any multicast tree T_{mu} and $\forall l \in T_{mu}$, we have $l \in \mathcal{L}_T$. Therefore, a multicast tree is a subtree of the scheduling tree. Since the multicast stream originates from the mesh BS, the corresponding multicast tree is rooted at the mesh BS.

Consider a request generated from node i. Our multicast join algorithm finds the node n_j that minimizes the cost of joining the multicast tree. The number of hops (i.e. wireless links) between two nodes i and j in the scheduling tree is denoted as $h(i,j)$. We note that a stream consumes identical bandwidth in each link. As a result, the actual bandwidth cost of joining the multicast at node j is proportional to $h(i,j)$. We refer to this bandwidth cost as the *join cost*. Thus, we can minimize the *join cost* by minimizing $h(i,j)$.

Given a request generated from a SS node i, Algorithm 3.2 outputs the node n_j at which the request should join the multicast.

Table 2 Algorithm 3.2

Algorithm 3.2 Join Multicast
1. if(node i belongs to T_{mu})
2. return i
3. $k = i$
4. while (k != 0) do
5. if(node k belongs to T_{mu})
6. return k
7. $k = p_k$ // trace to parent
8. return k // trace up to BS

The basic idea of the algorithm is to find the nearest ancestor in the multicast tree. We have the following result for Algorithm 3.2.

Theorem 3.2 *Algorithm 3.2 minimizes the join cost for a request generated from SS* n_i.

Proof: If n_i is in the multicast tree, then the *join cost* is zero, which is the first two lines of Algorithm 3.2.

If n_i is not in T_{mu}. Algorithm 3.2 finds n_i's nearest ancestor in the multicast tree. We prove it by contradictory. Assume the optimal n_j ($j \neq i$) is not the ancestor of n_i. Recall that P_j denotes the path from SS n_j to the BS. If n_i is in P_j, n_i must be in the multicast tree, which is a contradictory. If n_i is not in P_j, then there exists a node n_k in the scheduling tree such that n_k is the ancestor of both n_i and n_j (in the extreme case, $k = 0$) and n_k is in the multicast tree. Since the routing only considers links in the scheduling tree, we have $h(i,j) > h(i,k)$, which is contradictory to the assumption that n_j is the optimal node. Therefore, the optimal node must be n_i's nearest ancestor in the multicast tree. □

We now consider how to leave a multicast group. Generally, a user can choose to leave a multicast group at anytime. Algorithm 3.3 handles that a user of SS n_i leaves a multicast tree T_{mu}. The basic idea is if a link in the multicast tree only serves this user, we should remove the link from T_{mu}. The corresponding

bandwidth consumed in this link is also released. Otherwise, the user is removed without affecting the multicast tree.

Table 3 Algorithm 3.3

Algorithm 3.3 Leave Multicast
1. remove leaving user from the multicast group
2. $k = i$
3. while(k is a leaf of T_{mu} AND no user from k is in T_{mu})
4. remove link(k, p_k) from T_{mu}
5. $k = p_k$
6. return

3.5 Adopt Patching

In this section, we employ the idea of Patching to let the later users join an earlier multicast group. We first extend the joint admission control and channel scheduling algorithm for unicast to support Patching and then explore the physical layer multicast to further improve the physical channel utility. Our solutions still guarantees the data rate for all the admitted users.

3.5.1 Data Rate Guarantee for Patching

In Section 3.3, we present the admission control for unicast video stream. Similarly, the multicast admission decision should consult the lower layer bandwidth consumption situation in order to guarantee the data rate for the multicast steams. Even if there is a multicast group available for a later request, we do not admit it if the network does not have enough residual bandwidth for the multicast and Patching streams. We formulate the bandwidth capacity constraints for Patching as follows. For the ith multicast group, let $T_{mu}(i)$ denote the multicast tree, a_i denote the required data rate of the video streams of this multicast group (i.e., the scheduler should reserve a_i bandwidth for both the multicast stream and the patching streams in this multicast group) and $Pch(i)$ denotes the set of the paths of the ongoing Patching streams. We use $\mathcal{M}_i = \{T_{mu}(i), a_i, Pch(i)\}$ to represent a multicast group. For a mesh network with N simultaneous multicast

groups, denote the *multicast solution* $\mathcal{M} = \{\mathcal{M}_1, \ldots, \mathcal{M}_N\}$ as the set of all the ongoing multicast groups in the network. Notice that $\mathcal{M}$ is built and maintained by Algorithm 3.2 and Algorithm 3.3.

There are two basic ways for patching streams and regular video streams to share the bandwidth resources in the network. One solution is called *unified approach*. In the unified approach, the patching streams and regular video streams are equally treated, i.e., they are all treated as common data traffics and share the w_l bandwidth in link l. The fraction of time that link l needs to be activated can be characterized as

$$f_l = (1/w_l) \cdot (\sum_{i:l \in T_{mu}(i)} a_i + \sum_{i:l \in p, p \in Pch(i)} a_i) \qquad (3.4)$$

where $p \in Pch(i)$ is the path of Patching stream in $\mathcal{M}_i$. Substituting f_l in (3.1) by (3.4) yields,

$$\sum_{l:l \in \mathcal{N}(i)} (1/w_l) \cdot (\sum_{i:l \in T_{mu}(i)} a_i + \sum_{i:l \in p, p \in Pch(i)} a_i) \le 1, \forall i \in \mathcal{N} \qquad (3.5)$$

From Theorem 3.1, we can derive the following corollary:

Corollary 3.1 *If there is only the primary constraint in WiMax mesh networks,* (3.5) *is the necessary and sufficient condition for a feasible multicast solution* $\mathcal{M}$ *under the unified bandwidth sharing approach.*

In contrast to the unified approach, there is a *split approach* where each link l commits w_l^p ($w_l^p < w_l$) bandwidth for patching streams. The residual bandwidth is dedicated for regular video streams. In this case, the feasibility of scheduling the patching streams and regular video streams are validated separately. Similarly to corollary 3.1, we have:

Corollary 3.2 *If there is only the primary constraint in WiMax mesh networks, the necessary and sufficient condition for a feasible multicast solution* $\mathcal{M}$ *under the split bandwidth sharing approach is*

$$\begin{cases} \sum_{l:l\in\mathcal{N}(i)} \dfrac{\sum_{i:l\in T_{mu}(i)} a_i}{w_l - w_l^p} \le 1 \\ \sum_{l:l\in\mathcal{N}(i)} \dfrac{\sum_{i:l\in p,\, p\in Pch(i)} a_i}{w_l^p} \le 1 \end{cases} \quad \forall i \in \mathcal{N} \qquad (3.6)$$

We use simulation to compare the performance of these two bandwidth sharing approaches and present the result in Section 3.6.

Table 4 Algorithm 3.4

Algorithm 3.4 *WiPatching*
1. For i = 0 to end of *wqueue*
2. Use Algorithm 3.2 to update $\mathcal{M}$ for *wqueue*[i].
3. if (new $\mathcal{M}$ violates (3.5) or (3.6))
4. Undo the update of $\mathcal{M}$. //postpone request
5. else
6. *rqueue*.add(*wqueue*[i]) // admit request
7. *wqueue*.dequeue(i)
8. For i = 0 to end of *rqueue*
9. if(*rqueue*[i] is leaving)
10. *rqueue*.dequeue(i)
11. Update $\mathcal{M}$ by Algorithm 3.3
12. Generate the schedule based on $\mathcal{M}$.

We refer to the WiMax version of *Patching* as *WiPatching* and sketch it in Algorithm 3.4 which is an extension of Algorithm 3.1. Besides maintaining the request waiting queue *wqueue* and running queue *rqueue*, we also need to update the current multicast solution $\mathcal{M}$ in the network. If a request in the waiting queue can join an ongoing multicast group in $\mathcal{M}$, the corresponding multicast group is updated according to Algorithm 3.2. Otherwise, a new multicast group is added into $\mathcal{M}$ to a setup a video stream demanded by the waiting request. We shall further check if the corresponding new $\mathcal{M}$ after admitting this request violates the constraints in either (3.5) or (3.6), depending on which bandwidth sharing approach is adopted. After passing the validation, the request will be admitted, otherwise the request is postponed and $\mathcal{M}$ will be rolled back to the earlier version. If any ongoing user leaves, we use Algorithm 3.3 to update the multicast

solution for its multicast group. Similar to Algorithm 3.1, in line 12, we can generate the multi-graph based on $\mathcal{M}$ and assign the minislots accordingly.

3.5.2 **Physical Layer Multicast**

One more way to save bandwidth is taking advantage of the physical layer multicast. For example, node i, j, k are in the same multicast tree, node j, k are children of node i. If multicast streams from n_i to n_j and n_i to n_k share the same minislots, the bandwidth consumption of the multicast can be reduced by half. Generally speaking, if n_i is the nearest ancestor of both n_j and n_k, n_j and n_k are not ancestors of each other, then n_i can use physical layer multicast to save the bandwidth. Figure 5 illustrates a more complex example of using physical layer multicast. Assume SS5 joins a multicast group that involves SS4. In this case, even SS4 and SS5 are not children of SS1, because they get the stream from SS2 and SS3 respectively, SS1 can still multicast the video stream to its children, which are SS2 and SS3.

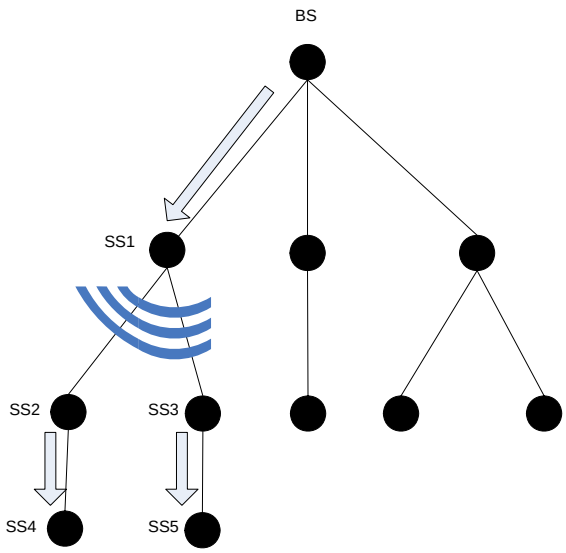


Figure 5. Illustration of using physical layer multicast

WiMax supports this kind of multicast. The WiMax standard [50] defines the multicast connection in PMP mode in one hop range, and the Connection ID (CID) used for the multicast service is the same for all SSs that participate in the multicast. We can adopt this physical layer multicast feature in the mesh mode. In practice, the antenna beam of a station should only cover all its children in the scheduling tree,

which facilitates the physical multicast but does not introduce the secondary constraint into the networks. We call this enhanced technique as *WiPatching*+.

3.6 Performance Study

We conduct comprehensive simulations to compare the performance of the *WiPatching* (WP), *WiPatching*+ (WP+) and the joint admission control and channel scheduling for unicast (AC). In the simulations, we considered a WiMax mesh network with one BS and 15 SSs. The simulation topology (i.e., the scheduling tree) is depicted in Figure 6.

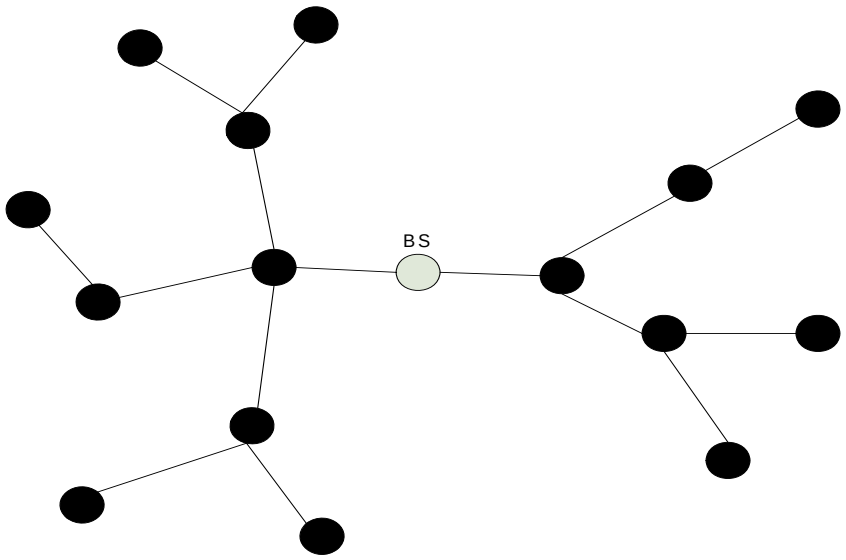


Figure 6. Simulation Topology

In WiMax mesh networks, the SSs typically are equipped with directional antenna fixed on top of buildings with LOS connections between each other. We thus assume the channel condition is static, with constant burst rate. The bandwidth capacity of links is set as $w = w_l$ = 59Mbps, $l \in \mathcal{L}_T$ according to [53]. The required streaming data rate is randomly distributed between 0.6 Mbps and 1 Mbps, which simulate streaming videos with different quality of resolutions.

We implemented the AC, WP and WP+ in our simulation code. The performance metrics used in this study are the acceptance ratio θ and average waiting time τ. The acceptance ratio θ is defined as:

$\theta = \frac{Number\ of\ admitted\ requests}{Number\ of\ submitted\ requests}$. This metric represents the capacity of the video server. The waiting time of a request is the time that a request spends in the waiting queue. The requests not admitted at the end of simulation are still counted. The average waiting time evaluates whether the technique offers true VOD service. The ideal waiting time for the VOD service is zero.

The time period after the admission of the first request in a multicast group, during which Patching is employed, is referred to as the Patching window. Therefore if the arrival time of a later request exceeds the Patching window of a multicast group, we do not assign the request to that multicast. A large Patching window favors the size of the multicast group; however, it increases both the number and duration of the Patching stream. The optimal Patching windows scheme is discussed in Chapter 4.

Assume the service requests from all the SSs follow the same Poisson arrival with arrival rate λ. For simplicity, we round the arrival time of the requests arriving between the *i*th and *i*+1th minute to be *i*+1 (*i* is an integer). Although this approximation loses some accuracy, we think it is adequate to qualitatively simulate the system. There are 20 different videos with the same length of 50 minutes. Each simulation runs 200 minutes. Each data point in the result figures is average value of 1000 simulation results.

3.6.1 **Compare Unified Approach and Split Approach**

We first study the performance of two different bandwidth sharing approaches, which are *unified approach* and *split approach*. These two approaches are implemented in WP. Let w_p denote the bandwidth reserved for patching streams in each link. We first vary w_p in the split approach from 3 Mbps to 45 Mbps. The average inter-arrival time of request is set to be 4, 5 or 6 minutes. Since there are 15 SSs in the network, the inter-arrival time of the request arrival process in the BS is 4/15, 5/15 or 6/15 minutes respectively.

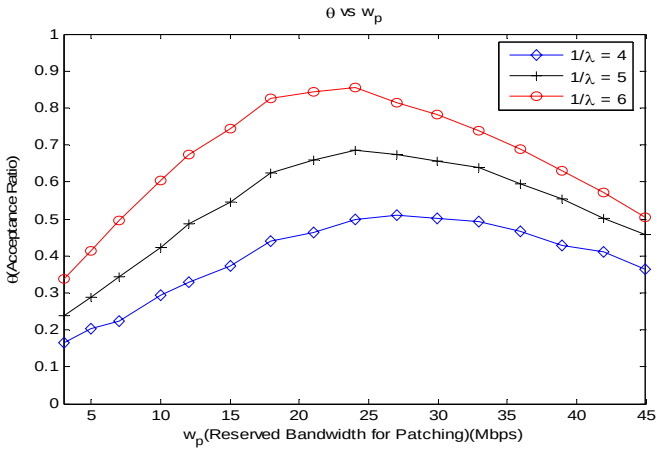


Figure 7. θ with different w_p

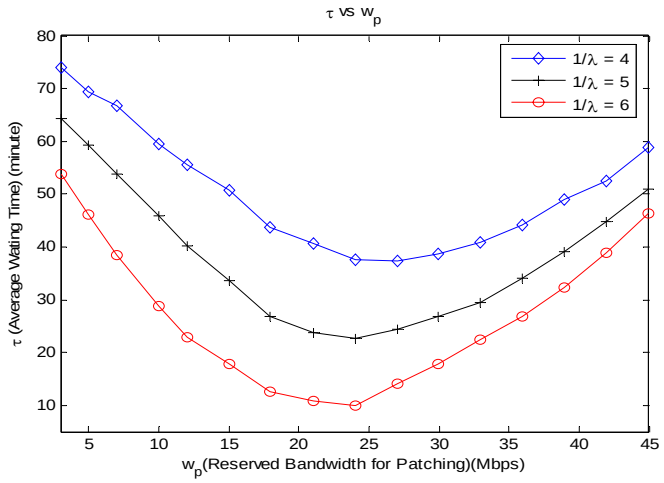


Figure 8. τ with different w_p

We plot the average acceptance ratio and average waiting time in Figure 7 and Figure 8. It is clearly shown in the plots that all the performance curves have inflexion points, when w_p is 24Mbps or 27 Mbps. This result indicates that we should have a balanced bandwidth allocation for patching stream and regular video stream. If w_p is too small, video streams have less chance to join a multicast group. In another hand, large w_p results in small bandwidth for regular video streams, which means a video request has less chance to join a multicast, if this join will consume extra bandwidth. Therefore, it is necessary to find the optimal w_p for split approach.

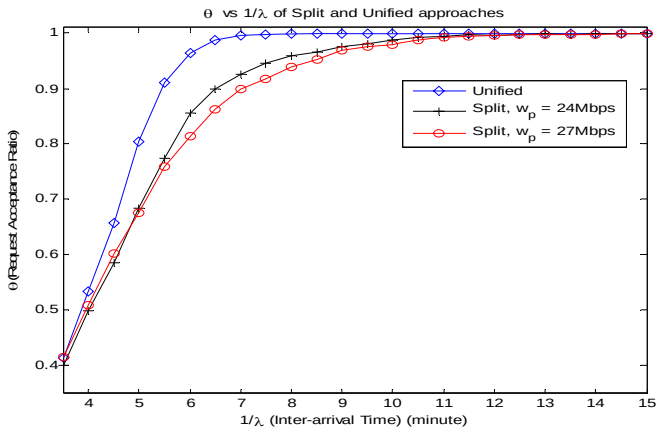


Figure 9. Compare θ of Unified and Split Approaches

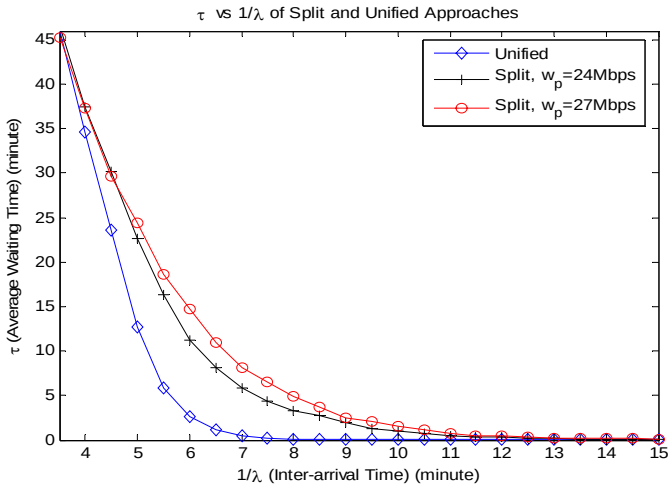


Figure 10. Compare τ of Unified and Split Approaches

We then compare the performance of the unified approach with the split approach with optimal w_p. We increase the arrival rate λ to challenge the system. Figure 9 and Figure 10 depict the average request acceptance ration and waiting time with different inter-arrival time respectively. We find that the unified approach outperforms the split approach with optimal w_p. This result shows that it is inefficient to fix the bandwidth reserved for patching streams. In the rest of the performance study, we adopt the unified approach in WP and WP+.

3.6.2 **Compare AC, WP and WP+**

The performances of AC, WP and WP+ are studied in this section. It is expected that WP and WP+ outperforms AC because of the using of multicast routing and patching scheme. The simulation parameters remain the same as the previous one. We vary the inter-arrival rate and record the average acceptance ratio and average waiting time of each technology.

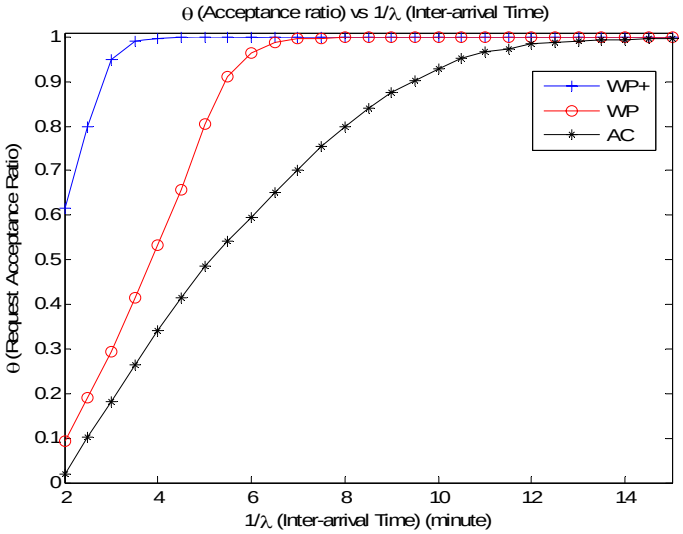


Figure 11. Request acceptance ratio

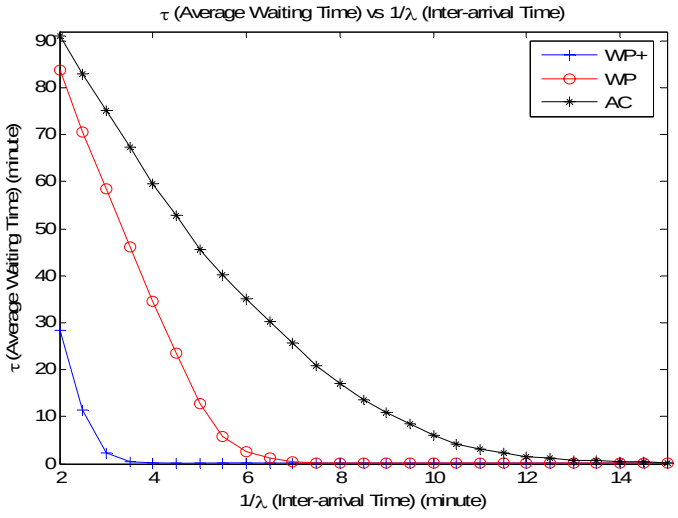


Figure 12. Average waiting time

We plot the result in Figure 11 and Figure 12. Figure 11 depicts the curves of θ vs $1/\lambda$, where $1/\lambda$ is the request inter-arrival time in each SS. The request arrival rate at the BS is the sum of the arrival rate of each SS. The result in Figure 11 shows that the performance of AC is worse than WP and WP+. We also see that WP+ outperforms WP when the inter-arrival time getting small. When the acceptance ratios of WP and AC decrease to 0.1, WP+ can keep the acceptance ratio above 0.6. This is because WP+ better saves the bandwidth and is able to accept more requests. Figure 12 depicts the average waiting time under different request arrival rate. We observe that the result in Figure 12 perfectly matches the results in Figure 11. As the acceptance ratio decreases, there are more waiting requests, thus the average waiting time becomes larger. Specifically, WP and WP+ provide an almost zero τ with smaller inter-arrival time. WP+ performs even better than WP. Our conclusion from this test is that WP and WP+ extend the capacity of AC and offer true VOD service under heavier request demand than AC does, while WP+ is the best technique among the three.

The multicast shares video streams among users requesting the same video, and can thus keep the workload of the server (i.e., the aggregate request arrival rate at the server) unchanged. If the requests for the same video appear less frequently, the server has less chance to use Patching. Assume the request is uniformly distributed among all the videos. It is expected that if the number of videos increases, the probability that two users request the same video becomes lower, and the performance gain of multicast will decrease. We conduct simulations to study this effect by changing the number of videos in the server. The number of videos is set to be 10, 20 and 30. Other settings remain the same as previous test.

Figure 13 and Figure 14 depict the acceptance ratio and average waiting time of WP and WP+ with different number of videos respectively. From both figures, we observe that the performance of WP and WP+ decline as the number of videos increases, which verifies the expectation. It is also interesting to see that the worst performance of WP+ (i.e., with 30 videos) performs better than the best performance of WP (i.e., with 10 videos). This further verifies the efficiency of WP+.

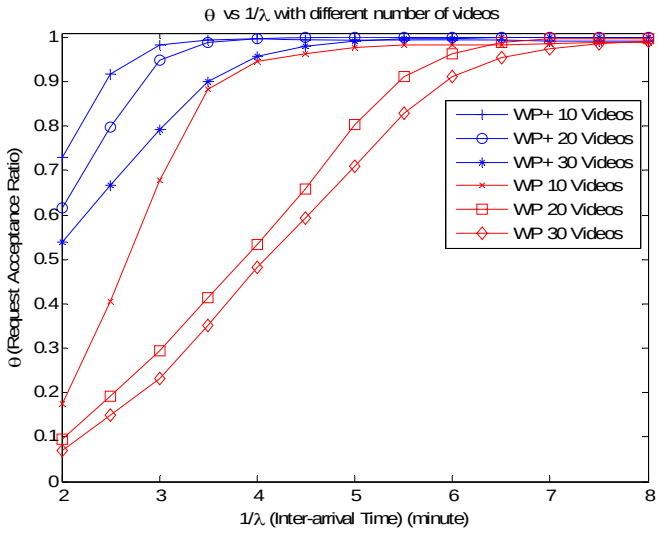


Figure 13. Acceptance ratio with different number of videos

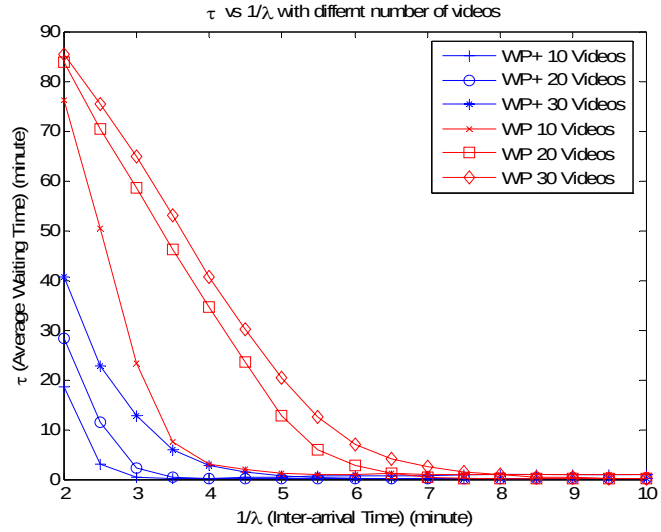


Figure 14. Average waiting time with different number of videos

3.7 Conclusion

In this chapter, a cross-layer framework for VOD service in WiMax-based WMN is proposed. We aim at supporting true VOD service in residential or business networks with a WiMax based wireless backhaul. We first propose a model that can jointly solve the admission control and channel scheduling problem in WiMax mesh networks. The joint solution provides data rate guarantee for video streams. We

first demonstrate the model for unicast video stream. We then apply multicast techniques to extend the capacity of the video server. We propose techniques to build multicast routing trees on top of scheduling trees. The proposed routing algorithm makes use of the well-maintained scheduling tree and thus introduces less maintenance cost. The algorithm also minimizes the cost of joining a multicast tree. Based on the multicast routing algorithm, we apply the application layer Patching technique which can offer true VOD service. We also extend the joint admission control and channel scheduling scheme to guarantee the data rate for Patching. Simulation study verifies the benefit of using Patching in Wimax mesh networks. Two bandwidth sharing approaches are proposed for patching streams and regular video streams. Sophisticated simulation study shows that the *unified approach* outperforms the *split approach*. Our simulation results also validate that using physical layer multicast can further improve the performance of Patching.

4. OPTIMIZING PATHCING IN WMA

In this chapter, we adopt a Patching-based multicast technique for video sharing in the general WMA environment. We optimize the Patching-based multicast by addressing two critical problems, namely, the Minimum Cost Multicast Tree (MCMT) problem and the Maximum Benefit Multicast Group (MBMG) problem. The MCMT problem is to find a minimum cost multicast tree in the network. We show that finding such a tree in the WMN can be formulated as a graph theory problem, which is to find the tree with minimum number of non-leaf nodes, and which spans all the nodes in the multicast group. We further prove the problem is NP-hard and propose a fast greedy algorithm to accommodate the real time feature of the VOD application. We solve the MBMG problem by minimizing the communication of a Patching group in the entire network. A Markov model is proposed to capture the growth of the multicast group in the WMN. Simulation study results validate the proposed solutions of the two problems.

4.1 Introduction

Multicast is an effective technique to reduce the demand on server bandwidth. Standard multicast, however, is not suited for VOD applications since the majority of the users would be forced to wait for more requests for the same video before the service can start. This is particularly unacceptable for less popular videos. To allow the users of a multicast group to start their own video playback at their arrival time instead of the multicast time, the Patching technique [26] is proposed. The basic idea of Patching is as follows. A new service request can exploit an existing multicast by buffering the on-going stream from the multicast while playing a new "catch-up" flow from the server. Once the catch-up flow, which is called Patching stream, has been played back to the skew point, it is terminated and the original multicast can be shared.

Existing Patching techniques ([26], [51], [52]) rely on IP multicast, which is not available for WMN. Moreover, when they try to optimize the data sharing benefit of multicast, only bandwidth consumption in the server end is considered. This is inaccurate in WMN, where bandwidth consumed in

each link should be considered as the "cost" of the multicast group. Due to the aforementioned reasons, a naive adaptation of the Patching technique for WMN would achieve limited performance gain.

In order to optimize the performance of Patching over WMN, two critical problems need to be taken into consideration. They are the Maximum Benefit Multicast Group (MBMG) problem and the Minimum Cost Multicast Tree (MCMT) problem. We briefly discuss these two problems and the related works in the rest of this section.

From the networking point of view, Patching is to group multiple unicast sessions into a multicast session while introducing a unicast overhead (i.e., Patching stream) for each grouping operation. A later request requires a longer Patching stream. As the length of the Patching stream increases, the benefit of the grouping decreases. It is important to know when is the right time to start a new multicast instead of grouping the new request into an existing multicast. Is it possible to have a grouping strategy that maximizes the benefit of the Patching scheme? This question motivates the MBMG problem. Our solution of MBMG problem in this work is different from the prior optimizing Patching techniques ([51], [52]), since we take the network topology into consideration and minimize the communication cost in the entire network.

Having an optimal grouping strategy is not good enough to convey Patching over WMN, because finding an efficient multicast tree is non-trivial. Finding a multicast tree with the minimum sum of edge weights in a random weighted graph is the famous Steiner Tree problem [37]. However, the authors of [38] have recently pointed out that the optimal multicast tree in wireless mesh networks is not a Steiner tree. The actual number of transmissions to multicast a packet is smaller than the number of links of the tree due to the broadcast nature of wireless transmissions. However, it is still interesting to study the complexity of general multicast in WMN.

We will show that the required number of transmissions equals to the number of non-leaf nodes in the multicast tree. In this work, the tree with minimum number of non-leaf nodes and spanning the nodes in the multicast group is called a *Minimum Cost Multicast Tree*. Finding such a tree is the proposed MCMT

problem. We will demonstrate that the MCMT problem is NP-hard in a unit disk graph. Since the grouping is in real time, when the group membership changes, we may invoke the tree construction algorithm to build a new tree. We propose a fast greedy algorithm to accommodate the real time feature of the VOD application.

4.2 MCMT Problem

We model the wireless mesh network as a connected graph $G = \{V, E\}$, where V is the set of mesh nodes, and E is the set of edges. Each mesh node has the same communication range. There is an edge between two nodes, if they are in the communication range of each other. Let (u,v) and (v,u) denote the same edge between node u and v. Unlike those papers analyzing the capacity of the wireless mesh network, we do not employ any network model in this work, since we focus on the tree construction problem.

We assume the mesh node equipped with omni-directional antenna, therefore the signal transmitted by a node can be received by all its neighbors. Given a multicast tree T, the root and intermediate node of T just needs to transmit a packet once, and all its children can receive the packet. Since the leaf nodes do not forward packet, we have:

Lemma 4.1 *The number of transmissions to multicast a packet to all the nodes in a multicast tree T is the number of non-leaf nodes in T.*

Based on Lemma 4.1, we define the cost of a multicast tree T in a WMN as $C(T)$, which is also the number of non-leaf nodes of T. The MCMT problem of a graph can be formally defined as follows.

Definition of MCMT problem: Given a connected graph $G = \{V, E\}$, we have a set $S \subseteq V$, while S consists of the source and the receivers of a multicast. Let S_T denote the set of tree that spanning all the nodes in S. Find a tree $T^* \in S_T$ such that $T^* = \arg\min_{T \in S_T} C(T)$. T^* is called the Minimum Cost Multicast Tree (MCMT) of G.

Since the source of the multicast always transmits the packet, the problem can be interpreted as finding a multicast tree with minimum number of intermediate nodes and spanning all the nodes in *S*. When $S = V$, we broadcast the packet in the network. In this case, we show that the MCMT is highly related to the *Minimum Connected Dominating Set* (MCDS) of this graph. The MCDS is a connected dominating Set with minimum cardinality. The MCDS problem has been thoroughly studied when people try to construct a virtual backbone for routing in the wireless ad hoc networks (e.g., [55], [56], [57]). Denote U as the MCDS of G and $\overline{U} = V - U$. According to the definition of MCDS, there exists a function $f : \overline{U} \to U$ such that $\forall v \in \overline{U}$, there is an edge $(f(v), v)$. We can generate a tree T from U by Algorithm 4.1, which outputs a set of edges E_T that represents T.

Table 5 Algorithm 4.1

Algorithm 4.1. Generate a tree from a MCDS	
Input: G, U, f.	
Output: E_T	
1.	G_U = G's subgraph induced by U
2.	T_U = spanning tree of $G_{U=}$
3.	E_T = all edges in T_U
4.	For every $v \in \overline{U}$ Do
5.	$E_T = E_T \cup \{(f(v), v)\}$

In order to be the MCMT, T has to be the spanning tree of *G*. This is proved in the following lemma.

Lemma 4.2 *If the input U is a connected dominating set (CDS) of G, Algorithm* 4.1 *generates a spanning tree of G*.

Proof: Since the original T is T_U which is the spanning tree of G_U, and every node in $\overline{U}$ is added to T by introducing one extra edge, T is the spanning tree of G. □

The cost of the tree is investigated in the Lemma 4.3.

Lemma 4.3 *If T is generated by Algorithm* 4.1 *and U is the input MCDS, we have* $|U|+1 \geq C(T) \geq |U|$.

Proof: Since every node in $\overline{U}$ is added to T by attaching it to a node in U, every node in T with degree larger than one is also in U. A degree one node n in T can not be in U, otherwise $U-\{n\}$ is a smaller connected dominating set of G. Consequently, the number of nodes in T with degree larger than one equals to $|U|$. If the source of the multicast (i.e., the root of T) is in U, $C(T)=|U|$; otherwise $C(T)=|U|+1$. Thus we prove $|U|+1 \geq C(T) \geq |U|$. □

Now we are able to proof that T is the MCMT when the input U of Algorithm 4.1 includes the multicast source or the source is not in any MCDS of G.

Lemma 4.4 *Let* s *denote the source of the multicast. When S = V, Algorithm* 4.1 *generates the MCMT for G if and only if either of the following two cases is true*.

CASE 1: $s \in U$,

CASE 2: *s is not in any MCDS of G*,

Where U is the input MCDS of Algorithm 4.1.

Proof: Assume T is the tree generated by Algorithm 4.1. According to Lemma 4.2, we have $T \in S_T$. Assume there is another $T_1 \in S_T$, such that $C(T) > C(T_1)$. Since T_1 is also a spanning tree of G, the set of node in T_1 with degree larger than one, denoted by U_1, is a connected dominating set of G. Notice that s may not in U_1, we have $C(T_1) \geq |U_1|$ Let us analysis the two cases separately.

In CASE 1, Since $s \in U$, according to Lemma 4.3 we have

$|U| = C(T) > C(T_1) \geq |U_1|$.

Since $|U| > |U_1|$, it is controversy that U is the MCDS of G. Therefore T has the minimum cost and is the MCMT of G.

In CASE 2, we have $|U| = C(T) - 1$ and $|U_1| = C(T_1) - 1$. Since $C(T) > C(T_1)$, we still get $|U| > |U_1|$. Similar to CASE 1, we proof *T* is the MCMT of *G*.

If we are not in either CASE 1 or CASE 2, then $s \notin U$ and there is another MCDS U' such that $s \in U'$. In this case, we can use U' as input and generate a tree T'. According to Lemma 4.3, $C(T') = |U'|$ and $C(T) = |U| + 1$. Since $|U'| = |U|$ and $C(T') > C(T)$, *T* is not the MCMT of *G*. □

Finally, we are able to show that the MCMT problem is NP-hard by showing that its special case is NP-hard. Since WMN is usually modeled as a unit disk graph (e.g., [38], [55]), we think the result is more interesting for a unit disk graph.

Theorem 4.1 *The MCMT problem is NP-hard in a unit disk graph*.

Proof: Lemma 4.4 shows that if we can solve the MCDS problem for a unit disk graph, there is a polynomial time algorithm (i.e., Algorithm 4.1) to solve a special case of the MCMT problem (i.e., when *S* = *V*). Since the MCDS problem in a unit disk graph is NP-hard [58], the MCMT problem in a unit disk graph is also NP-hard. □

Consider the VOD application in wireless mesh networks. If a request is generated by the user attaching to a particular mesh backbone node, we say this is the node of the user request. Similarly, when a new request is merged into a multicast group, we say its node joins the multicast. Since the requests are generated at random, in terms of its node id and arrival time, the multicast tree may be reconstructed when a new node joins the multicast. Consequently, we need a fast tree construction algorithm to handle the real time feature of our application.

It is interesting to study the fast approximation algorithm for the MCMT problem. To the authors' best knowledge, there is no such study in the literature. However, since the focus of this paper is to show the benefit of Patching-based multicast for VOD application over wireless mesh network, we propose a fast greedy algorithm for us to use in the simulation study. In Algorithm 4.2, we denote *d(v)* as the degree of node *v*. The function *f* has been defined previously. The insight of Algorithm 4.2 is to find the non-leaf

nodes from the MCDS. All the irrelevant nodes are removed in order to construct a tree spanning *S*. We proof the correctness of Algorithm 4.2 in the following theorem.

Table 6 Algorithm 4.2

Algorithm 4.2 Greedy algorithm for MCMT problem

Input: $G(V, E)$, S

Output: E_T

1. U = MCDS of G
2. $X = \{v \mid v \in S, v \notin U\}$
3. $Y = \{v \mid v \notin S, v \notin U\}$
4. While $Y \neq \varnothing$ Do
5. $V = V - Y$
6. For $v \in X$ Do
7. $f(v) = \arg\max_{u \in U, (u,v) \in E} d(u)$
8. $E = E - \{(v,u) \mid u \neq f(v)\}$ //end for loop
9. $Y = \{u \mid u \in U, u \notin S, d(u) = 1\}$
10. $X = X \cup \{u \mid u \in U, u \in S, d(u) = 1\}$
11. $U = U - Y - X$ //end while loop
12. Generate E_T from Algorithm 4.1 with $G(V, E)$, U, f as input.

Theorem 4.2 *Algorithm* 4.2 *constructs a tree that spans S.*

Proof: Since the major part of the algorithm is a while loop, we discuss the result of each iteration of the loop. Since we only remove nodes in *Y* from *V* in line 5, no node in *S* is deleted. It is obvious that both *G* and *U* are connected after the operation between line 6 and line 8. Since we only remove degree one nodes from *U*, *U* is still connected and dominates all the nodes in *G*. Therefore in line 12, the input *U* is a connected dominating set of *G*. According to Lemma 4.2, Algorithm 4.1 produces a spanning tree of *G*. Since *Y* is empty, all the leaf nodes of the spanning tree are in *X*. Thus we prove Algorithm 4.2 returns tree spanning *S*. □

Algorithm 4.2 requires for an algorithm to produce the MCDS of the graph in the beginning. There are many existing algorithms for MCDS, we choose a fast and efficient algorithm proposed in [59]. For convenience, we call Algorithm 4.2 as *Minimum Cost Tree* (MCT) algorithm in the rest of the paper.

4.3 MBMG Problem

Define a *multicast group* as a group of user sharing the video stream in the network. Our goal is to guarantee the requested data rate of every VOD user while minimizing the traffic introduced into the network. In order to solve the optimal patching problem, we have to define some metrics to capture the gain as well as the cost of grouping several users into a multicast group. In this work, we employ per user workload introduced by the multicast group into the network as the metrics. It is the average amount of data that the VOD application requires the network to transmit in order to meet the requested data rate of each member in the group. This metrics is denoted as *D*. The goal of this work is to minimize *D*. The formal definition of *D* is presented in the following section.

4.3.1 Definition of *D*

Denote the first user of the multicast group as the regular user and the later user merged into this group as patching user. The *ith* user arrives at time t_i second ($i = 0,1,...,N$). For simplicity, assume the video stream is constant bit rate (CBR) traffic and the rate is *R* bits/second. We also assume ideal admission control and scheduling schemes such that every non-leaf node in the multicast tree can transmit at *R* bps for an admitted video streaming session.

We use the same graph model for WMN as the model used in the MCMT problem. Given a multicast group of users, define a *multicast node set* as the set of nodes that the users in a *multicast group* attach to. We assume the system adopt a tree-based multicast routing algorithm *A*. Given *G*, a *multicast node set* θ, and a source node *s* as input, $A(G, s, \theta)$ returns a multicast tree rooted at *s* and spanning all the nodes in θ. The source *s* is the gateway node connecting to the video server. Note that since θ gradually grows as more users join the group, we assume that *A* is called when a new member joins into θ and a new

multicast tree is generated accordingly. Without loss of generality, we assume there is no user request generated at the source *s*. In this work, we optimize grouping for a single server, therefore we only consider one source in the Graph. When there are multiple multicast sessions in the same WMN, the result of this work remains the same for each session, as long as the overall bandwidth consumption of these multicast sessions does not exceed the capacity of the network. Please refer to the admission control algorithm discussed in Chapter 3. We label the nodes in G as $j = \{0, 1, \ldots, M\}$, where 0 is the label of source *s*. Table 7 lists the major notations used in the analysis.

Unlike the prior works ([51], [52]), we consider the workload in each mesh node in the calculation of D. Denote θ_i as the multicast node set after *ith* user arrives, where θ_0 includes the source and the mesh node of the regular user. Denote T_i as the multicast tree corresponding to θ_i, $T_i = A(G, s, \theta_i)$. Denote $C(\theta_i)$ as the cost of a *multicast node set*. $C(\theta_i)$ equals to the number of transmissions required to multicast a packet in T_i.

We denote D_m^i as the required data transmission in the multicast tree between the arrival of the *ith* and (*i*+1)*th* user ($i < N$). We further denote D_m^N as the required data transmission in the multicast tree after the arrival of the *Nth* user and $t_{N+1} = t_v + t_0$, where t_v is the length of the video. D_m^i is calculated as:

$$D_m^i = C(\theta_i) \cdot (t_{i+1} - t_i) \cdot R \qquad (4.1)$$

Therefore, we have $D_m = \sum_{i=0}^{N} D_m^i / (N+1)$. Similarly, the required data transmission in the *ith* Patching stream ($i > 0$) is calculated as:

$$D_p^i = L_i \cdot (t_i - t_0) \cdot R \qquad (4.2)$$

Since the regular user does not need Patching stream, we define $D_p^0 = 0$. The required data transmission for each user in the Patching streams is calculated as $D_p = \sum_{i=1}^{N} D_p^i / (N+1)$. Finally we define:

$$D = D_m + D_p = \frac{\sum_{i=0}^{N} C(\theta_i) \cdot (t_{i+1} - t_i) + \sum_{i=1}^{N} L_i \cdot (t_i - t_0)}{N+1} \cdot R \qquad (4.3)$$

Table 7 Notations used in Section 4.3

D	Per user data transmitted
D_m	Per user data transmitted in multicast session
D_p	Per user data transmitted in Patching streams
D_m^i	Data transmitted in the multicast tree between the arrival of the *ith* and (*i*+1)*th* user.
D_p^i	Data transmitted in *ith* Patching stream
L_i	Path length of the *ith* Patching stream
M	Number of non-source mesh nodes
N	Number of patching user of a multicast group
t_v	Length of the video
t_i	Arrival time of the *ith* user
τ	Patching window size
R	Video transmission rate
θ_i	Multicast node set after *ith* user arrives
T_i	Multicast tree spanning θ_i
$C(\theta_i)$	Cost of θ_i, i.e., the number of data transmission in T_i
λ_j	Rate of the Poisson user arrival at mesh node *j*
λ	Rate of the Poisson user arrival at server end
P_N	Probability that *N* user requests arrive at *s* in τ seconds
p_j	Probability that a request is generated at mesh node *j*

Equation (4.3) is the closed formula of *D*. Clearly, *D* includes all of the data required to be successfully transmitted in the WMN by a gradually generated VOD multicast group. *D* could be

interpreted as the per user workload given by a particular VOD multicast group to the WMN. We understand that D does not equal to the actual data transmitted in the WMN in order to fulfill the requirement from the application, since lots of retransmissions are involved during the streaming due to the lossy wireless channel and contention in the WMN. Let us assume that if the over all workload of the multicast group is within the capacity of the WMN, then $D \cdot (N+1)$ is the lower bound of the actual amount data transmitted for a multicast group. Therefore we think D is a reasonable optimization goal to guide us making the grouping decision.

If we know the value of all the variables in (4.3) during the run time, we can make the grouping decision as follows. After a new user request arrives, we calculate the D_{new} using (4.3) and compare it with the D_{old} before this request. If $D_{new} \le D_{old}$, this user request is merged into the multicast, because the merging favors the multicast group; otherwise, we start a new multicast group and set this user as the regular user of a new group.

4.3.2 **Optimal Patching Window**

In real world implementation, the grouping usually takes place on the server side. Therefore, it is hard for the server to get all the information to use equation (4.3), unless we design the mesh network and video server into a cross-layer framework. We use a metric called *Patching Window* (PW) to help the server decide when to start a new multicast. We denote PW as τ. If a user request arriving at time t is τ seconds later than the beginning of a multicast group (i.e., $t - t_0 > \tau$), it can not be merged into this multicast group. By using PW, the server can convey the grouping without any information of the variables in the right side of (4.3). The optimal value of PW is calculated offline.

Assume the user request is randomly generated at each mesh node of WMN, consequently D is a random variable. We derive the optimal size of PW denoted as τ^* that minimizes expected value of D (i.e., $E[D]$) . From equation (3), $E[D] = E[D_m] + E[D_p]$, we derive $E[D_m]$ and $E[D_p]$ respectively.

Let the user arrival at mesh node j (j = 1,…, M) follows a Poisson process of rate λ_i. Therefore the user arrival at the server is a Poisson process of rate $\lambda = \sum_{j=1}^{M} \lambda_j$. Denote the inter-arrival time of two use requests as a random variable t, which has PDF $f(t) = \lambda e^{-\lambda t}, (t \geq 0)$. Consequently, we have $t = t_i - t_{i-1}$. The distribution of two important events of this analysis can be calculated as follows:

$$P_N = \Pr(N \text{ requests arrive at } s \text{ in } \tau \text{ seconds}) = \frac{(\lambda\tau)^N e^{-\lambda\tau}}{N!} \qquad (4.4)$$

$$p_j = \Pr(\text{A request is generated at node } j) = \frac{\lambda_j}{\lambda} \qquad (4.5)$$

These two events are independent. They indicate *when* and *where* a request is generated

4.3.3 **Derivation of** $E[D_m]$

In this section, we study the expected size of data transmitted by the multicast session of the group. Recall that the user request has a Poisson arrival and the inter-arrival time of two use request as a random variable t. Consequently, we have $t = t_i - t_{i-1}, (i > 1)$ and $N \cdot t = t_N - t_0$. Based on our definition, $E[D_m]$ can be presented as:

$$E[D_m] = E[\frac{t \cdot \sum_{i=0}^{N-1} C(\theta_i)}{N+1}] \cdot R + E[\frac{C(\theta_N) \cdot (t_v - N \cdot t)}{N+1}] \cdot R \qquad (4.6)$$

Let us define:

$$C_N = \begin{cases} 0, \text{ if } N = 0, \\ \dfrac{\sum_{i=0}^{N-1} C(\theta_i)}{N}, \text{ if } N > 0. \end{cases}$$

Since $\tau = N \cdot t$, (4.6) can be rewritten as:

$$E[D_m] = E[\frac{C_N}{N+1}] \cdot \tau \cdot R + E[\frac{C(\theta_N)}{N+1}] \cdot (t_v - \tau) \cdot R \quad (4.7)$$

Given the value of N, both C_N and $C(\theta_N)$ are random variables related to the random user generation in a graph and the multicast tree construction algorithm. On the other hand, N itself is a random variable independent to $C(\theta_N)$. Consequently we have:

$$E[\frac{C_N}{N+1}] = \sum_{N=0}^{\infty} \frac{E[C_N]}{N+1} \cdot P_N \quad (4.8)$$

$$E[\frac{C(\theta_N)}{N+1}] = \sum_{N=0}^{\infty} \frac{E[C(\theta_N)]}{N+1} \cdot P_N \quad (4.9)$$

Since $E[C_N] = \sum_{i=0}^{N-1} E[C(\theta_i)]/N, (N > 0)$, Equation (4.8) and (4.9) indicate that $E[D_m]$ can be represented as a function with two variables, N and τ, if we are able to calculate $E[C(\theta_N)]$ for each N. To derive $E[C(\theta_N)]$, we use a homogeneous Markov model to capture the dynamics of the multicast node set as follows.

We define the state as subset of $V - \{0\}$ (recall that s is labeled by 0). We use a binary sequence $\{b1, b2, \ldots, b_M\}$ to index the state. We have:

$$b_i = \begin{cases} 1, \text{node } i \text{ is in the state,} \\ 0, \text{node } i \text{ is NOT in the state.} \end{cases}$$

The corresponding decimal number of the binary sequence is the index of the state. For example, $S_1 = \{1\}$, $S_3 = \{1,2\}$. We define the special case $S_0 = \{0\}$, and the probability that a user request generates at node 0 is $\lambda_0 = 0$. There are 2^M states in total. Each state is a possible *multicast node set*. Denote the transition probability from S_j to S_k as $p_{j,k}$, which means S_k could grow from S_j with probability $p_{j,k}$. The transition probability is calculated by three rules:

Rule 1: If $k = j$, $p_{k,k} = \sum_{i \in S_k} p_i$,

Rule 2: If $S_k - S_j = \{i\}, i \in V, p_{j,k} = p_i$,

Rule 3: Otherwise, $p_{j,k} = 0$.

Rule 1 corresponds to the case that a user request generates from a mesh node in the current multicast node set. **Rule 2** indicates the case that a user request generates from a mesh node outside the multicast node set. **Rule 3** means there is no transition between any other two states after applying **Rule** 1 and **Rule** 2. The transition graph depicted in Figure 15 illustrates the growth of a multicast node set. The growth starts at S_0. According to the rules, when the first request arrives, the state transits from S_0 to all the states with cardinality of 2. Similar transitions happen when the *N*th request joins the multicast group.

The transition matrix of this model is $\mathrm{P} := (p_{j,k})_{2^M \times 2^M}$. Since θ_N is the multicast node set after the arrival of one regular user and N patching users, the distribution of θ_N can be obtained from $\mathrm{P}^{N+1} := (p_{j,k}^{N+1})_{2^M \times 2^M}$. More specifically, we have $\Pr(\theta_N = S_k) = p_{0,k}^{N+1}$. Therefore, $E[C(\theta_N)]$ can be calculated as:

$$E[C(\theta_N)] = \sum_{k=1}^{2^M} \Pr(\theta_N = S_k) \cdot C(S_k) = \sum_{k=1}^{2^M} p_{0,k}^{N+1} \cdot C(S_k) \qquad (4.10)$$

Since the number of states in the model is exponential to the size of the graph, we can reduce the size of the model by merging all the states with the same cost i into one state K_i such that $K_i = \{S_j \mid C(S_j) = i\}$. The transition probability from K_i to K_j, denoted by $q_{i,j}$, is calculated as:

$$q_{i,j} = \sum_{\forall S_m \in K_i, S_n \in K_j} p_{m,n}$$

Similar to the old model, a transition in the new model also represents the arrival of a user request. Assume the maximum cost of a multicast node set is W. The number of states of the new model is $W + 1$. W is the number of transmissions to broadcast a packet in the networks. Since broadcast can be done by a spanning tree, we have $W \le M$.

Denote the transition matrix as $Q := (q_{j,k})_{(W+1)\times(W+1)}$. Similar to equation (4.10), we can calculate $E[C(\theta_N)]$ as:

$$E[C(\theta_N)] = \sum_{i=0}^{W} q_{0,i}^{N+1} \cdot i$$

Although we are able to calculate the value of $E[C(\theta_N)]$ for each N, equation (4.7) has infinite component which includes both N and τ. Since N and τ are dependent, it is hard to derive a closed formula of $E[D_m]$ with single variable τ. We discuss this issue after the derivation of $E[D_p]$.

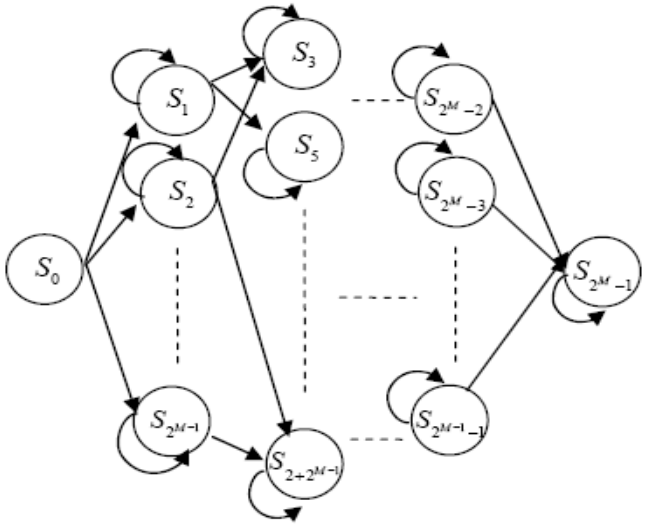


Figure 15. Illustration of the transition graph

4.3.4 **Derivation of** $E[D_p]$

In this section, we study the expected size of data transmitted by the Patching streams of a multicast group. From equation (4.3), $E[D_p]$ can be represented as:

$$E[D_p] = E[\frac{\sum_{i=1}^{N} L_i \cdot (t_i - t_0)}{N+1} \cdot R]$$

Consider a multicast group, without loss of generality, let $t_0 = 0$. Assume the routing protocol guarantee that the path lengths of patching streams are *i.i.d*. Thus we let $E[L_i] = E[L], (i > 0)$, where L is the path length of a unicast session between the source and an arbitrary node. Since L_i is independent regarding to N and t_i, we have:

$$E[D_p] = E[L] \cdot E[\frac{\sum_{i=1}^{N} t_i}{N+1}] \cdot R \qquad (4.11).$$

Recall that the user request has a Poisson arrival and the inter-arrival time of two use request as a random variable t. Consequently, we have $t_i = t_i - 1 + t$, ($i > 1$) and

$$\sum_{i=1}^{N} t_i = \frac{(N+1) \cdot N \cdot t}{2}.$$

Equation (4.11) can be rewritten as:

$$E[D_p] = E[L] \cdot E[\frac{N \cdot t}{2}] \cdot R \qquad (4.12).$$

Since $\tau = N \cdot t$, equation (4.12) can be simplified as:

$$E[D_p] = \frac{E[L] \cdot \tau \cdot R}{2} \qquad (4.13).$$

Let H_j denote the shortest path between node j and the source ($j = 1, 2, ...M$). If the networks adopt a short path routing for unicast patching stream, we have:

$$E[L] = \sum_{j=1}^{M} H_j \cdot p_j .$$

The value of $E[L]$ can be at least approximated with other routing strategy, we omit the detail discussion here. Therefore the value of $E[D_p]$ can be calculated by equation (4.13).

4.3.5 **Calculate τ^***

Based on our derivations of $E[D_m]$ and $E[D_p]$, $E[D]$ can be calculated by the following equation.

$$E[D] = \sum_{N=0}^{\infty} \frac{E[C_N]}{N+1} \cdot P_N \cdot \tau \cdot R + \sum_{N=0}^{\infty} \frac{E[C(\theta_N)]}{N+1} \cdot P_N \cdot (t_v - \tau) \cdot R + \frac{E[L] \cdot \tau \cdot R}{2} \qquad (4.14)$$

Due to the infinite summations in the upper equation, $E[D]$ does not have a closed formula. We propose to derive the optimal τ^* by a numerical approach.

To study how the value of $E[D]$ varies with respective to τ, we conduct numerical study and plot the curves of $E[D]$ with different values of τ. To eliminate the infinite summation in the computation, we only calculate the values of N which appear with high probability in the summations (i.e., in equation (4.8) and equation (4.9)). Consider the *Chebyshev's inequality* as follows:

If a random variable x has a finite mean μ and finite variance σ^2, then for all real number k > 0,

$$\Pr(|x - \mu| \geq k) \leq \frac{\sigma^2}{k^2}.$$

Since the mean and variance of N are both $\lambda \cdot \tau$, according to the *Chebyshev's inequality*, for any real number $k > 0$, we have:

$$\Pr(|N - \lambda \cdot \tau| \geq k) \leq \frac{\lambda \cdot \tau}{k^2}.$$

For numerical study, we choose k such that $\frac{\lambda \cdot \tau}{k^2} \leq 0.01$. This means we have no less than 99% probability such that $\max(0, \lambda \cdot \tau - k) \leq N \leq \lambda \cdot \tau + k$.

We consider a start topology with 6 nodes, where the source is the hub node. The source can broadcast data to each node by one transmission. We set R = 300 kbps and t_v = 50 minutes. We set 5 mesh nodes have the same user arrival rate and set λ to four different values, i.e., 2.5, 5, 7.5 and 10 per minute.

We vary PW from 1 minute to 30 minute the plot $E[D]$ using equation (4.14) in Figure 16. The curves show that the benefit of patching decreases when PW is small. As the size of PW increases, the additional cost of patching stream start to dominate the communication cost of the group. Therefore, we see that $E[D]$ keeps increasing after it reaches a minimum value. However, τ^* varies with different settings. Therefore, we need to conduct numerical study for each setting of the network.

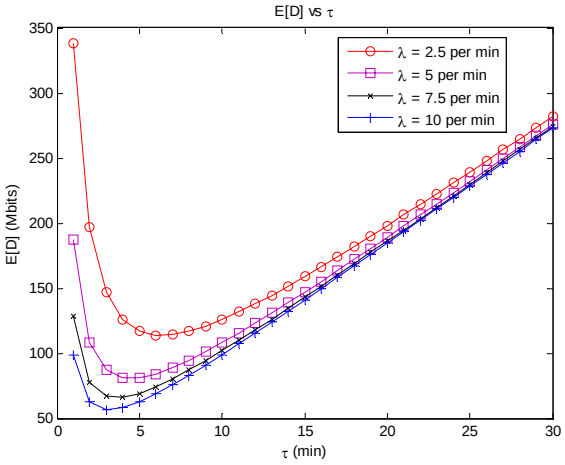


Figure 16. Result of a simple numerial study of E[D]

4.4 Performance Study

In this section, we use simulation to evaluate our solutions of the MBMG problem and MCMT problem. Specifically, we first validate our proposed analytical model for $E[D]$ by comparison between the results of the simulation study and numerical study. We then compare the proposed MCT with a fast algorithm for the Steiner Tree problem in terms of their corresponding minimum $E[D]$.

4.4.1 Model Validation

In this section, we compare the result of simulation study and the numerical study to validate the proposed analytical models of $E[D]$. The simulation code is written in C++. We simulate the growth of a multicast group as follows. User requests are randomly generated from each mesh node staring at time zero. The first request is the regular user request of the multicast group. We then group the later user requests as

many as possible until the arrival time of the request exceeds the patching window. These requests are patching user requests. A multicast routing algorithm is called to build multicast tree for the node set of each new multicast group. The shortest path routing is used for unicast patching stream. The data transmitted in the network is calculated accordingly. We generate several connected random graphs and also vary the user arrival rate. The average number of D from large number of simulations is compared with the $E[D]$ calculated by the model. Our study shows that the simulation results perfectly match the result derived from the model.

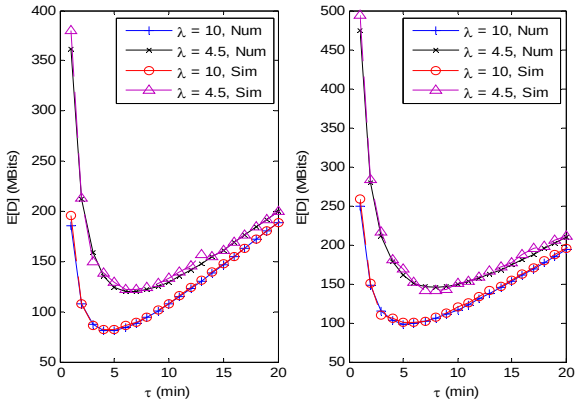
We simulate the Poisson arrival by letting the inter-arrival time of the requests follow a negative exponential distribution. The proposed MCT algorithm is adopted as the multicast routing algorithm. To validate our model under different network topologies, we generate the connected random graph as follows. Given the number of nodes of the graph as input, we first randomly generate a spanning tree connecting all the nodes. We then randomly decide how many extra edges to add into this spanning tree. Both end nodes of the extra edge are randomly decided. We generate several random graphs with number of nodes equaling to 10, 30 and 50. We think this numbers reflect the size of a typical WMN for residential areas.

The other settings of the simulation as well as the numerical approach are as follows. For the sack of the simplicity of the simulation, each node has the same user arrival rate. The aggregated rate on the server is set to be 10 or 4.5 per minute. The video data rate R is set to be 300 kbps and the video length is 50 minutes. For each random graph, we plot four curves for $E[D]$. Two curves are the average number of D calculated from 500 simulation results (denoted as Sim in the plot) with user arrival rate λ = 10 and 4.5 respectively. Similarly, other two curves are the actual $E[D]$ generated from the numerical approach (denoted as Num in the plot).

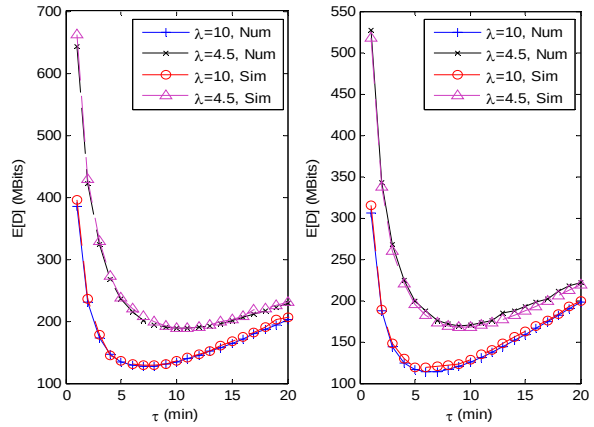
During the simulation, we observe that two graphs with different topologies may have similar plots. This is because the routing algorithm generates multicast trees with similar cost under those topologies. We only report the results of graphs with different plots.

We plot the results of 4 random graphs with 10, 30 and 50 nodes in Figure 17, Figure 18 and Figure 19 respectively. All the figures reveal three important facts:

- The optimal patching window sizes under different topologies and same other settings are different. This indicates that topology is relevant to the optimization goal. This observation also validates that the optimization framework in this work is different from the framework of prior works. Prior work will output the same value of the optimal patching window size with different topologies, since they do not include the data transmission requirement in the network in the formula of the optimization goal.
- The simulation result perfectly matches the result of the numerical approaches. This fact validates the correctness of our proposed model.
- The curves of $E[D]$ with different values of patching window have similar pattern. The curves decline when patching window is small. After $E[D]$ reaches its minimum value, it increase monotonically. This fact shows that $E[D]$ has a minimum value.

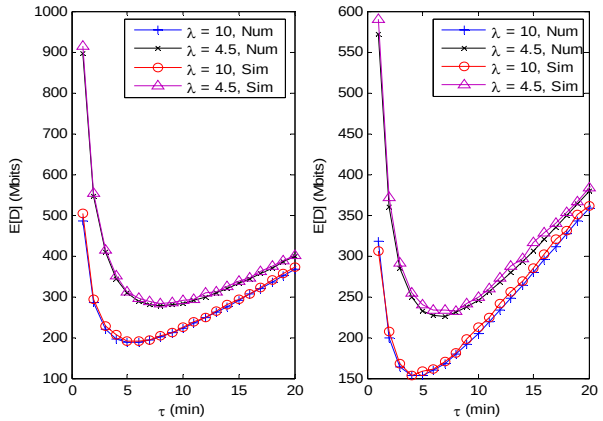


(a)

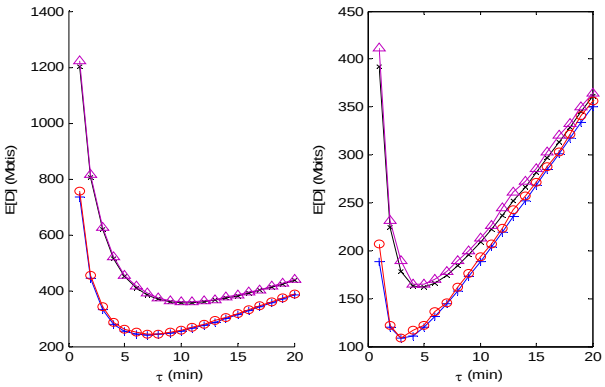


(b)

Figure 17. Results from 4 random graph with 10 nodes

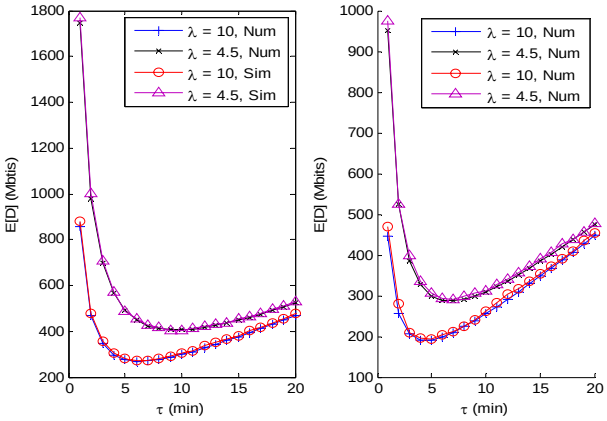


(a)

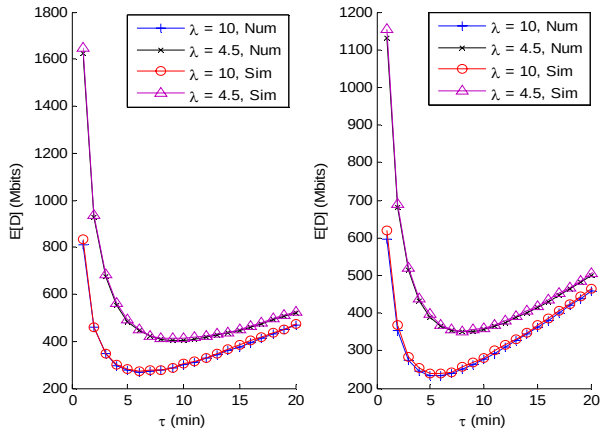


(b)

Figure 18. Results from 4 random graph with 30 nodes



(a)



(b)

Figure 19. Results form 4 random graph with 50 nodes

4.4.2 **Performance of MCT Algorithm**

In this section, we study the performance of the MCT algorithm. A fast Steiner Tree algorithm, namely MST algorithm [60], is used as the benchmark. The MST algorithm is a $2(1-1/l)$ approximation algorithm, where l is the number of leaves in the optimal tree. We assume the mesh network take advantage of the broadcast features of the wireless transmission, therefore the cost of the multicast tree equals to the non-leaf nodes in the tree. Since the interest of the paper is the data sharing for VOD application, both algorithms are used in the proposed numerical approach to derive the minimum $E[D]$, which is the

expected per user data transmission under the optimal Patching Window size. Let us denote the minimum $E[D]$ derived from the MCT algorithm and MST algorithm as MD_{MCT} and MD_{MST} respectively.

Table 8 Average on the minimum E[D]

λ (per min)	10 Nodes			30 Nodes			50 Nodes		
	MD_{MST}	MD_{MCT}	Save (%)	MD_{MST}	MD_{MCT}	Save (%)	MD_{MST}	MD_{MCT}	Save (%)
1	321.02	273.86	14.69	549.195	461.282	16.01	716.3325	612.882	14.44
2	233.19	197.39	15.35	385.301	319.219	17.15	520.9497	440.8674	15.37
3	192.19	162.32	15.54	317.68	265.544	16.41	429.7284	362.2545	15.7
4	168.32	141.06	16.19	282.055	238.013	15.61	368.4165	314.7123	14.58
5	149.62	126.49	15.45	257.488	217.36	15.58	335.1417	282.2196	15.79
6	137.14	115.59	15.72	234.156	197.866	15.5	306.1092	257.7375	15.81
7	127.68	107.24	16.01	221.692	184.661	16.71	288.3579	239.1093	17.08
8	119.50	100.23	16.12	208.05	174.154	16.29	266.8146	223.5384	16.22
9	111.74	94.68	15.27	198.056	167.333	15.51	248.5419	211.0722	15.07
10	107.79	90.03	16.48	187.834	159.03	15.33	239.3226	200.739	16.12
11	100.41	85.64	14.72	175.522	148.295	15.51	226.5246	190.8798	15.74
12	97.63	81.96	16.05	172.767	146.167	15.39	214.6035	182.6322	14.9
13	92.41	78.80	14.73	163.229	137.199	15.94	209.5317	175.617	16.18
14	89.75	75.49	15.88	160.645	134.577	16.22	200.7864	169.5024	15.57
15	85.21	71.31	16.32	153.349	128.421	16.27	194.3637	164.1699	15.54
16	82.64	69.23	16.23	145.616	122.512	15.87	189.3393	159.2166	15.91
17	79.83	67.35	15.64	139.555	116.698	16.37	184.7415	154.524	16.36
18	78.13	66.57	14.80	134.805	113.202	16.02	180.9495	149.8314	17.2
19	76.12	63.56	16.49	131.043	110.048	16.02	174.1239	145.6128	16.37
20	72.57	60.78	16.24	124.963	105.298	15.73	167.7249	141.8208	15.44

We set the aggregated arrival rate at the server end from 1 to 20 per minute. The other settings are the same as those in the Section 4.4.1. Given the number of nodes in the graph, we randomly generate 100 connected graphs and derive the MD_{MCT} and MD_{MST} under each graph topology. The average values of MD_{MCT} and MD_{MST} with these 100 graphs under different settings are reported in Table 8 . We conduct numerical studies for graphs with 10, 30 and 50 nodes. The result in Table 8 shows that MCT algorithm transmits about 15% less data than the MST algorithm under the various settings and topologies. This means MCT algorithm can save about 15% per user data transmission while providing the same quality of

VOD application as the MST algorithm. The reason that MCT algorithm can outperform MST algorithm under the same settings and topology is that MCT algorithm can construct less expensive multicast tree for most of the multicast node sets than the MST algorithm does.

Overall, the performance study reveals that an algorithm specified for the MCMT problem could outperforms a good approximation algorithm for the Steiner Tree problem. Moreover, the study also shows that about 15% performance gain could be achieved for VOD application.

4.5 Conclusion

In this chapter, we study two crucial problems for adopting a Patching-based multicast in WMN, namely the *minimum cost multicast tree problem* and *maximum benefit multicast group problem*.

We model the WMN as a connected graph and show that finding the minimum cost multicast tree in WMN is not only different from the similar problem in wired network, but also NP-hard. To adapt the real time feature of the VOD application, we propose a fast multicast routing algorithm, namely *Minimum Cost Tree* (MCT) algorithm, based on the existing *minimum connected dominating set* algorithms. We also show that the optimal grouping in the Patching technique is also different from the prior works, which only minimize the bandwidth usage in the server end. We propose to minimize the communication cost of a Patching group in the entire network. A novel Markov model is proposed to capture the dynamics of the multicast session in the network. By using the numerical approach, we derive the optimal patching window that minimizes the per user workload introduced by the multicast group.

Simulation results under different random graph topologies validate our proposed model. Moreover, we show that the MCT algorithm can save about 15% data transmission over the MST algorithm in our VOD scenario.

5. DYNAMIC STREAM MERGING

In recent years, there has been a dramatic increase in the number of users who access online videos from wireless access networks. It is highly desirable that such wireless networks are robust in handling sudden spurts in demand for various videos due to special events. Such abrupt increase in the network usage should not significantly impact other normal access to regular videos. A promising solution is to share the video streams in the wireless access network. However, conventional video sharing techniques assume cooperation with the video server. On the other hand, it is generally difficult, if at all possible, for wireless access networks to cooperate with online video sites. In this chapter, we tackle this problem in wireless mesh access networks by proposing a distributed video sharing technique called Dynamic Stream Merging (DSM). DSM improves the robustness of the access network without the need to involve the online video sites. We provide analytical analysis to show that per-link sharing performance can be optimized with little time and message complexity. Simulation study using NS-2 simulator is conduced to study the performance of DSM with a large system setting. We also present a wireless mesh network prototype based on DSM. This testbed allows different streaming sessions to share video data from the Mobile YouTube website. To the best of our knowledge, this is the first work that shares video data from a commercial online video site in a wireless mesh network. The results from the simulation and experiment validate the correctness of DSM and demonstrate the effectiveness of our prototype.

5.1 Introduction

In this chapter, we are interested in exploiting the possibility of video sharing in the WMA environment without the cooperation from the online video site. As discussed in Chapter 4, the video multicast technique consists of the multicast grouping process and the multicast routing process. These processes typically require that video provider collaborates with the network where the multicast takes place. This is generally not feasible in an online video access environment, since the video resource providers in the Internet do not own nor have control over the WMN. To address the aforementioned issue

associated with the loosely coupling between the Internet and WMA networks, we introduce a novel video sharing technique called *Dynamic Stream Merging* (DSM). It does not require the cooperation of the VOD servers in the Internet. Basically, DSM is a light-weight distributed solution that improves the robustness of the WMA environment without imposing much overhead on the network. To better evaluate the proposed technique, we built an 802.11g-based wireless mesh access network in our department building, using netbook computers as the mesh nodes. Without loss of generality, we choose the mobile YouTube site as the Internet VOD resource, and successfully demonstrate the sharing of YouTube video streams in our mesh network. An illustration of our WMA environment is given in Figure 20.

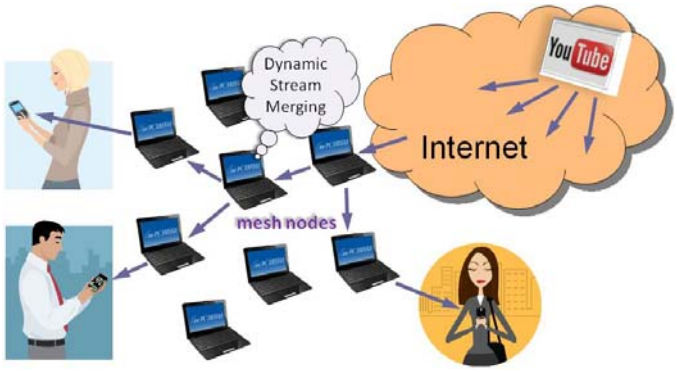


Figure 20. Illustration of the testing WMA environment

5.2 Dynamic Stream Merging

5.2.1 Stream Merging

A mesh network can be modeled as a directed graph G(V, E) where V is the set of mesh nodes and E is the set of links. A link is a pair of two mesh nodes who are in the communication range of each other. The nodes are labeled with i, where i = 1, 2, …, N, and N = | V |. Let M be the total number of video streams in the network. We also label the video streams as k, where k = 1, 2, …. M. If node i and node j are neighbors and there is a video stream k passing from i to j, we say stream k passes through link <i, j>. Nodes i and j are referred to as *upstream* and *downstream nodes*, respectively. We model a video stream as a sequence of non-equal-sized segments. The segment could be the video frame, *group of pictures* (GOP),

or user-defined chunk in the video. The granularity of the segment is flexible as long as it preserves the temporal order of the video. Without loss of generality, we assume that the segment ID of a video stream starts from 1 and grows in increasing order. We denote as t_i^x the highest of the segment IDs of all the video segments that have arrived at node *i* from stream *x*. Typically, t_i^x is the latest segment arriving at node *i* from stream *x*.

We give the sketch of our *Merging Algorithm* (i.e., Algorithm 5.1) in Table 9 . The input of the algorithm consists of a link <*i*, *j*>, and a set of streams *S* passing through link <*i*, *j*>. All the streams in *S* are for the same video. The algorithm has two parts. The first part, a loop, is for the upstream node *i*. In each iteration of the loop, Step 3 identifies the stream *x* with the property that $t_i^x \le t_i^z$ for any stream $z \in S$, where $z \neq x$. Similarly, the stream *y* with the second smallest such value is determined in Step 4. That is, we have $t_i^x \le t_i^y \le t_i^z$ for any $z \in S$, where $z \neq y$ and $z \neq x$. In Step 5, τ_i^x is given the value of t_i^y . τ_i^x is referred to as the τ *value* of stream *x* in this paper. In Step 6, node *i* informs node *j* *i*'s intension to merge stream *x* with stream *y*. The stream merging notation "$y \to x$" signifies that stream *x* is merged by stream *y*, and node *j* should now use the data received from stream *y* for both streams *x* and *y* in the downstream. We refer to stream *y* as the *acquiring stream* and stream *x* as the *merged stream* or *mergee* in this paper. After Step 6, we say that the status of stream *x* at node *i* is in "*merged*" mode. In Step 7, node *i* stops the transmission of the merged stream *x* after sending out segment τ_i^x although stream *x* continues to arrive at node *i*. We say that node *i* blocks stream *x* at link <*i*, *j*>. In Step 8, the merged stream *x* is removed from the set *S* to prepare for the next iteration of the merge process. Thus, we have one less stream after each round of merging. This process is repeated until there is only one stream left in *S*. The second part of Algorithm 5.1 is for the downstream node *j*. After *j* receives the merging notification of $y \to x$ from node *i* (Step 10), node *j* understands that node *i* will soon block stream *x* (after sending segment τ_i^x). In response, node *j* first forwards the remaining data segments arriving from stream *x* (Step 11). It then continues to forward the subsequent segments for this stream by reusing the same segments node *j* received earlier for stream *y*

(Step 12). Since stream x is behind stream y in the data streaming order, reuse of older data segments from stream y for stream x is possible. Let n be the size of the input set S. The message complexity of Algorithm 5.1 is O(n). The time complexity of the algorithm is also O(n) if the streams in S are sorted on their t_i^x value; otherwise we need to sort S first and the time complexity becomes O($n$log$n$).

Table 9 Algorithm 5.1

Algorithm 5.1 Input: set S, link $<i, j>$
1. **Algorithm at node *i***
2. While $\lvert S \rvert > 1$
3. $x = \arg\min_{k \in S} t_i^k$
4. $y = \arg\min_{k \in S-\{x\}} t_i^k$
5. $\tau_i^x = t_i^y$
6. i notifies j about $y \rightarrow x$
7. i stops forwarding x after sending out segment τ_i^x on x
8. $S = S - \{x\}$
9. **Algorithm at node *j***
10. Receives notification about $y \rightarrow x$ from i
11. Forward remaining data coming from stream x
12. Reuse data from stream y for stream x

We explain the effect of Algorithm 5.1 with a simple merging example as depicted in Figure 21. It shows that two streams, s_1 and s_2, with the same video ID, are passing through link <1, 2>. Since the stream merging $s_1 \rightarrow s_2$ occurring at <1, 2> does not affect the behavior of stream s_1, we focus our discussion on the impact on stream s_2 during the merging operation. In Figure 21(a), after node 1 receives segment 5 from stream s_1 (i.e., $t_1^{s_1} = 5$) and segment 2 from stream s_2 (i.e., $t_1^{s_2} = 2$), node 1 notices that both streams s_1 and s_2 have the same video ID (we will discuss video ID in Section 5.4) as well as the same node ID for the next hop (i.e., node 2). Node 1 then notifies node 2 of node 1's intension to merge these two

streams (i.e., $s_1 \rightarrow s_2$), and soon block stream s_2, at link <1, 2>, after segment 5 ($\tau_2^x = t_1^{s_1} = 5$). In response, node 2 treats data packets arriving from s_1 as data for both streams s_1 and s_2 in the downstream. That is, after node 2 finishes forwarding segments 2, 3, 4, and 5 from stream s_2 to the next hop, node 2 continues to forward the subsequent segments (i.e., 6, 7, 8, etc.) by reusing these segments received earlier for stream s_1. This can be achieved by spoofing the packet header with the header information of s_2, (i.e., IP, Port, etc.). The desirable effect of this merger is that we have one less stream transmission between nodes 1 and 2 as illustrated in Figure 21(b).

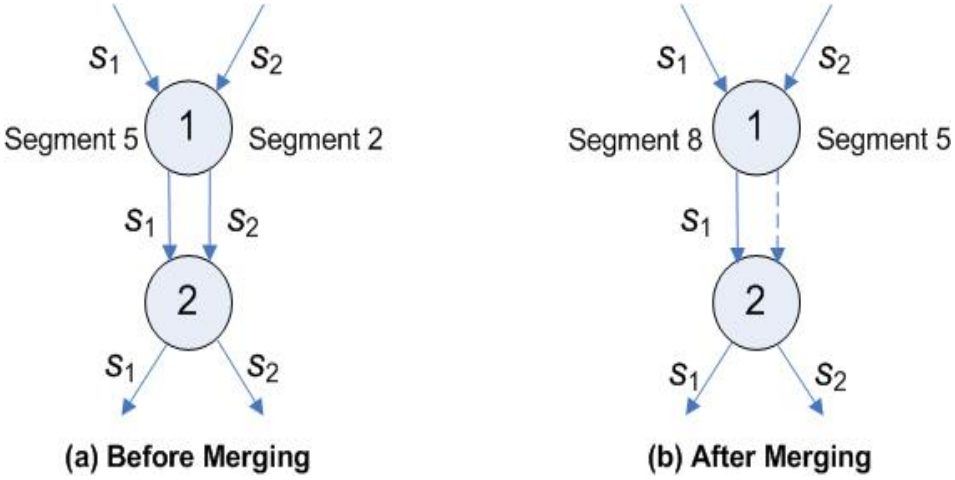


Figure 21. A simple merging example

We further prove some desirable properties of Algorithm 5.1 as follows.

Lemma 5.1 *Algorithm* 5.1 *constructs an ordered set of merging relationships*

$$\{ k_2 \rightarrow k_1, k_3 \rightarrow k_2, ..., k_{n-1} \rightarrow k_{n-2}, k_n \rightarrow k_{n-1} \}$$

such that $t_i^{k_1} \le t_i^{k_2} \le t_i^{k_3} .... t_i^{k_{n-1}} \le t_i^{k_n}$, $n = |S|$; *S and i are the input of Algorithm* 5.1.

Proof: According to Algorithm 5.1, k_i and k_{i+1} are the stream *x* and stream *y* chosen in the *i*th iteration of the while loop (*i* = 1, …, n-1). □

Based on Lemma 5.1, Algorithm 5.1 eventually merges all the streams in the set *S* into a single stream. We represent this series of stream merging as $k_n \rightarrow k_{n-1} \rightarrow ... \rightarrow k_2 \rightarrow k_1$, where

$t_i^{k_n} \geq t_i^{k_{n-1}} \geq t_i^{k_{n-2}} .... t_i^{k_2} \geq t_i^{k_1}$. $k_n \rightarrow k_{n-1} \rightarrow ... \rightarrow k_2 \rightarrow k_1$ is referred to as a *merging chain* in this paper. The head of the chain, k_n, is the only stream that remains after the sequence of mergers. That is, the length of this merging chain is increased by one stream after each iteration of Algorithm 5.1 as illustrated below:

Initial streams:	$k_n, k_{n-1}, ..., k_2, k_1.$
After one iteration:	$k_n, k_{n-1}, ..., k_2 ; \quad k_2 \rightarrow k_1$
After two iterations:	$k_n, k_{n-1}, ..., k_3 ; \quad k_3 \rightarrow k_2 \rightarrow k_1$
After *n*-1 iterations:	$k_n ; \quad k_n \rightarrow k_{n-1} \rightarrow ... \rightarrow k_2 \rightarrow k_1$

There are two factors that can affect the performance of video streaming, namely data loss and data delay. If the video data is lost in the network or cannot arrive at its destination in time, the video player is not able to decode the corresponding video frame. This affects the quality of the video playback. Let us define the data integrity property of a stream merging technique as follows. When stream merging is performed at any node *i* to reduce traffic on link <*i* , *j*> , the merging technique is said to ensure the *data integrity* if the upstream node *i* does not cause data loss or incur delay in preparing the data for forwarding to the downstream node *j*. We note that the upstream node *i* may experience data loss and delay in receiving its data from the last hop, due to various conditions such as congestion in the wireless environment. This impact on node *i* also affects the downstream node *j*. The scope of our definition for data integrity is limited to the stream merging actions internal to the upstream node *i*. In other words, we are primarily concerned with the correctness and efficiency of the proposed merging strategy.

Lemma 5.2 *Algorithm* 5.1 *ensures data integrity*.

Proof: Let us consider two nodes *i* and *j*, and the link <*i*, *j*> between them. According to the definition of data integrity, we need to prove that the sequence of mergers at node *i* does not incur data loss or delay for the downstream node *j*.

According to Lemma 5.1, Algorithm 5.1 constructs the *n*-1 mergers indicated in the merging chain $k_n \rightarrow k_{n-1} \rightarrow ... \rightarrow k_2 \rightarrow k_1$, where $t_i^{k_n} \geq t_i^{k_{n-1}} \geq .... t_i^{k_2} \geq t_i^{k_1}$. We prove Lemma 5.2 using mathematical induction

on *n*, the number of streams being merged. Since no merging occurs when *n*=1, we start the proof with n = 2.

Basis: If *n* = 2. We have $k_2 \rightarrow k_1$. In this case, k_1's data, up to and including segment $\tau_i^{k_1}$, are forwarded from node *i* to node *j* in the normal fashion without any delay. k_1's subsequent segments (after segment $\tau_i^{k_1}$) are always available ahead of time at node *j* for stream k_2. Thus, there is no data loss or delay due to $k_2 \rightarrow k_1$.

Inductive Step: Assume Lemma 5.2 holds when *n* = *N* (*N* > 2). Consider the case when *n* = *N* + 1. Algorithm 5.1 needs to do $k_{N+1} \rightarrow k_N \rightarrow ... \rightarrow k_2 \rightarrow k_1$, where $t_i^{k_{N+1}} \geq t_i^{k_N} \geq .... t_i^{k_2} \geq t_i^{k_1}$. Since Algorithm 5.1 merges the *N*+1 streams in the merging chain from right to left, the *N* rightmost streams in the chain are merged first. That is, Algorithm 5.1 first constructs $k_N \rightarrow k_{N-1} \rightarrow ... \rightarrow k_2 \rightarrow k_1$ over the first *N*-1 iterations, and then performs $k_{N+1} \rightarrow k_N$ in the final iteration. According to the induction hypothesis, $k_N \rightarrow k_{N-1} \rightarrow ... \rightarrow k_2 \rightarrow k_1$ can be done with the assurance of data integrity. Furthermore, we already proved in the base case that any two streams can be merged without incurring data loss or delay. Thus, $k_{N+1} \rightarrow k_N$ can also be done without data loss or delay. It follows that there is no data loss or delay due to $k_{N+1} \rightarrow k_N \rightarrow ... \rightarrow k_2 \rightarrow k_1$.

Since both the basis and the inductive steps have been proved, this lemma holds for any number of streams merged by Algorithm 5.1. □

Lemma 5.2 shows that Algorithm 5.1 provides the data integrity of the video stream at a single link. If we apply Algorithm 5.1 to all the links in the network, it can be proved that the data integrity property is also true over the entire network.

Theorem 5.1 *Applying Algorithm* 5.1 *at every link ensures data integrity across the entire network.*

Proof: According to Lemma 5.2, the data integrity property is ensured for stream merging at any link <*i*, *j*> in the network. Since merging of streams at the upstream node *i* has no effect on the ability of the

downstream node *j* in forwarding the data to the next hops in the streams, none of the nodes in the next hop can be affected by the mergers in <*i*, *j*>. Therefore, Algorithm 5.1 can be applied independently at each link, and the data integrity property holds at all the links in the network. □

The property stated in Theorem 5.1 is desirable. It says that the proposed optimization technique for wireless video dissemination is a distributed solution. It employs a simple local algorithm with little overhead.

Given a merger $y \rightarrow x$ at link <*i*, *j*>, we define $\alpha_i^x = L_x - \tau_i^x$, where L_x is the total number of data segments in stream *x*. In other words, α_i^x is the amount of data transmission saved at link <*i*, *j*> due to the fact that the upstream node *i* does not need to continue to send the data segments on stream *x,* after segment τ_i^x.

Theorem 5.2 *Algorithm* 5.1 *maximizes the value of* α_i^x *for any merged stream x in a given merging chain.*

Proof: Given a video, the difference $D_i(y,x) = t_i^y - t_i^x$ is called the *data delivery distance* between streams *y* and x at node *i*. If $D_i(y,x) > 0$, stream *y* is said to be ahead of stream *x* at node *i*. Stream *y* is behind stream *x* if $D_i(y,x) < 0$. From Lemma 5.1, Algorithm 5.1 performs a sequence of merging operations $k_2 \rightarrow k_1$, $k_3 \rightarrow k_2$, …, $k_{n-1} \rightarrow k_{n-2}$, and $k_n \rightarrow k_{n-1}$ in that order, such that $t_i^{k_1} \le t_i^{k_2} \le t_i^{k_3} .... t_i^{k_{n-1}} \le t_i^{k_n}$. This merging order ensures that any of the mergers, say $y \rightarrow x$, the merged stream *x* is merged with the acquiring stream *y* that is least ahead of *x* among all the streams being merged. τ_i^x, therefore, is minimized (Steps 4 and 5 in Algorithm 5.1). This follows that $\alpha_i^x = L_x - \tau_i^x$ is maximized. □

Theorem 5.2 shows that Algorithm 5.1 maximizes the performance of merging at each wireless link in the network.

5.2.2 **Buffering Scheme**

To facilitate reuse of the data segments for another stream some time later, we need to cache the current segments to a buffer. Let us recapitulate the merging operation. Given a merger $y \rightarrow x$ at link <*i*, *j*>, the upstream node *i* blocks stream *x* at <*i*, *j*> after it forwards segment τ_i^x, where $\tau_i^x = t_i^y$. Before, stream *x* is blocked by node *i*, node *j* experiences data arriving for stream *x* from both incoming streams *x* and *y* simultaneously (i.e., segments t_i^x to τ_i^x from stream *x*, and concurrently subsequent segments from stream *y*). To minimize out-of-order forwarding of stream *x* to the next hop, node *j* can buffer the early data from stream *y* while it finishes forwarding the remaining data (segments t_i^x to τ_i^x) from stream *x*. We note that severe out-of-order delivery of data segments may complicate the reassembly of the video frames at the player. DSM ensures that data packets are relayed in the same order they arrive at a mesh node.

The buffering can be done as follows. Node *j* caches *x*'s segments after segment τ_i^x (i.e., these segments are from stream *y*) into a buffer while it is relaying the segments between t_i^x and τ_i^x from stream *x* to the next hop in the downstream. When node *j* does not see any more data arriving in stream *x* from node *i* (i.e., *i* has blocked the stream), node *j* fixes the size of the buffer and starts to use it as a FIFO queue as follows. Each subsequent data segment from stream *y* is appended to the end of the queue while node *j* removes the segment at the head of the queue and forwards it to the next hop. By using this buffering scheme, node *j* maintains the order of stream *x* while it also reuses the data from *y* for *x*. The buffer is released at the end of the video session.

Since the buffering scheme consumes memory space in downstream node, an admission control scheme could be employed to avoid the potential buffer overflow when the required buffer size is not affordable. It is challenging to estimate the exact size the buffer given a merging, since both the size of each segment and the segment rate of the stream vary. Given merging $y \rightarrow x$, to approximate the upper bound of the buffer size, one can use the maximal segment size seen by the downstream node multiply $t_i^y - t_i^x$ since if the If the segment rate of the stream is constant, we will have to buffer $t_i^y - t_i^x$ segment

before *x* is terminated. The segment rate of stream could be constant. For example, if the unit of the segment is frame, the segment rate is constant since a video stream typically has a constant frame rate.

We give an example in Figure 22 to illustrate the buffer management strategy. In this figure, the dark arrows to the left represent stream S1 and the light arrows to the right depict stream S2. The initial state of this example is shown in Figure 22 (a). It shows that two streams with a temporal distance of 3 video segments are passing through node N2. To merge these two streams, N2 dynamically allocate a FIFO buffer with a capacity of three chunks as seen in Figure 22(b). In this figure, N2 continues to receive and forward the video segments for stream S1 and S2. However, N2 also saves the incoming segment 7 arriving on S1 to the FIFO buffer. The scenario in Figure 22 (c) illustrates the final step of the merging when the FIFO is full. From this time on, N2 can forward video segments to S2 in the downstream, using data from the FIFO buffer.

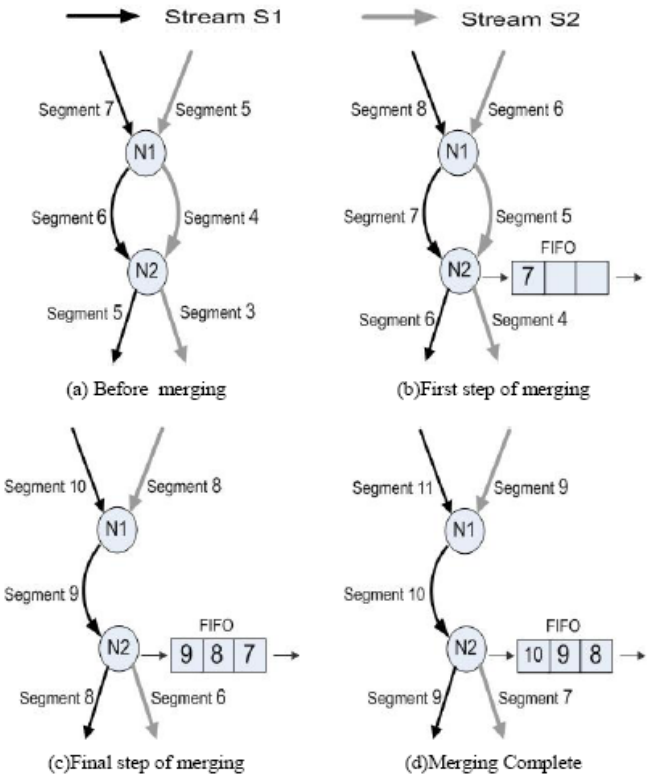


Figure 22. Example of the buffering scheme

5.2.3 **Cancel the Merging**

Due to the dynamics of the network, we may have to cancel the merging in a link. For example, consider the stream merging $s_1 \rightarrow s_2$ occurring at link <1, 2> as illustrated in Figure 21. According to Algorithm 5.1, node 2 reuses data from stream s_1 and relays them to the next hop in stream s_2. If stream s_2 is later terminated by the end user or the routing protocol diverts s_2 from node 1, node 1 will experience the discontinuation of the stream s_2. In a different scenario, if the routing protocol changes the next hop of s_2 at node 1, node 1 can recognize this change from the routing table. In any of these cases, node 1 needs to inform node 2 to stop caching data for stream s_2 and release the FIFO buffer.

To support the canceling operation described above, each mesh node must be able to differentiate between the actual termination of an incoming stream and the temporary pause of the stream due to the link condition such as congestion and interference. This can be achieved as follows:

- Each mesh node monitors the incoming data stream to estimate its data rate.
- Each mesh node periodically broadcasts a beacon or hello message. A node can detect the congestion and interference in a wireless link by measuring the RSSI (Received Signal Stream Indicator) and FLR (Frame Loss Rate) of the beacons from the mesh nodes in the proximity [77].

An incoming stream at a mesh node is considered as a *dead stream* if its estimated data rate is zero, but the link condition is good. We note that a merged stream (e.g., stream s_2 in link <1, 2> in Figure 21) is not considered as dead because the downstream node is able to reuse the data from the acquiring stream for the merged stream (e.g., stream s_1 in link <1, 2> in Figure 21).

The proposed merger cancellation technique is presented in Table 10 . Each mesh node periodically checks the data rate, link condition, as well as the local routing information of each stream. Given a stream x on link $<i, j>$, if stream x is dead or its next hop information changes at node i, node i and node j use Algorithm 5.2 (in Table 10) to cancel all the mergers relevant to stream x.

Table 10 Algorithm 5.2

Algorithm 5.2 Input: stream x on link $<i, j>$ // x is dead or its next hop changes at node i
1. **Algorithm at node *i***
2. If x is merged by y at link $<i, j>$ then
3. Remove $y \rightarrow x$
4. For any stream merged by x // affected by the cancellation of x
5. Set the stream as unmerged
6. Notify j to cancel the mergers relevant to x
7. Wait for the acknowledgement from node j
8. Run Algorithm 5.1 to attempt to remerge the unmerged streams
9. Unblock and resume forwarding of remaining unmerged streams
10. **Algorithm at node *j***
11. Receive the notification from i about the cancellation of stream x
12. If x is merged by y at link $<i, j>$ then
13. Remove $y \rightarrow x$
14. Stop caching data from y for x
15. Release the corresponding FIFO buffer for data from y
16. For any stream merged by x
17. Set the stream as unmerged
18. Release the corresponding FIFO buffer for data from x
19. Send an acknowledgement to node i about the cancellation
20. Wait for node i to activate the merging of the unmerged streams

Similar to Algorithm 5.1, Algorithm 5.2 also has two parts. The first part runs on node i, and the second part on node j. In the first part of the algorithm, node i first updates all the mergers relevant to x. If x is merged by y (i.e., $y \rightarrow x$), the merger $y \rightarrow x$ is removed at node i. All the streams merged by x are set as unmerged. Node i then notifies node j to cancel the mergers relevant to x and waits for node j to acknowledge the cancellation. When node i receives the acknowledgement, it calls Algorithm 5.1, in Step 8, to find new acquiring streams to remerge the unmerged streams. There may be some streams remain unmerged after the execution of Algorithm 5.1. In this case (Step 9), node i needs to unblock and resume the forwarding of these streams (e.g., stream s_2 are blocked by node 1 in Figure 21).

Let us define a stream k as a dead stream at node i if the sampling data rate of k at node i is below certain threshold. The sampling duration and the threshold are tunable parameters in the system. A mesh node periodically monitors the data rate of the ongoing streams. Note that the data copied from other streams is also counted on in the sampling, therefore a stream merged on link $\{i, j\}$ will not be treated as dead in downstream node j. However, this may cause node j not being able to detect the death of a merged stream.

We propose to use Algorithm 5.2 to handle the dynamics of the networks. We call this algorithm as the *Cancel Algorithm*. The input of the algorithm is a stream k on link $\{i, j\}$ which is found dead at node i. Similar to Algorithm 5.1, this algorithm has two parts. The first part runs on node i. When node i senses the death of stream k, it sets all the streams previously merged by k as unmerged. These unmerged streams are illegible to be merged in Algorithm 5.1 again. If k is merged on link $\{i, j\}$, node j will not be able to detect k's death. Therefore node i also needs to notify node j about the death of stream k. In the second part of the algorithm, when node j receives this notification from node i, it stops copying video data on stream k and sets all streams merged by k as unmerged. Consequently, stream k will not be forwarded by node j and all the streams merged by k are illegible to be merged by Algorithm 5.1 again. Finally both nodes will call Algorithm 5.1 to merge those unmerged streams. Node i will also resume the streams which were previously merged by k and remain unmerged after the call of Algorithm 5.1.

In the second part of the algorithm, when node j receives the notification from node i about x, it cancels all the mergers relevant to x. If x is merged by y (i.e., $y \rightarrow x$) at link $<i, j>$, node j removes the merger $y \rightarrow x$, stops caching data from y for x, and releases the FIFO buffer for data from y. For all streams merged by x, node j sets them as unmerged and releases the corresponding FIFO buffers. Node j then sends an acknowledgement to node i about the cancellation and waits for node i to invoke Algorithm 5.1 to merge the unmerged streams at link $<i, j>$. The original version of Algorithm 5.1 uses unmerged stream as input, which means that only the unmerged stream can be the acquiring stream in the algorithm. Actually, the proof of Lemma 5.2 shows that any ongoing stream starts before stream x can be the acquiring stream

of x, no matter it is merged or not. Therefore, we add a set U as the input of Algorithm 5.1. U is the set of all ongoing streams in link <i, j>. We change step 4 in Algorithm 5.1 as $y = \arg\min_{k \in U-\{x\}, t_i^k \geq t_i^x} t_i^k$. Since each stream is still merged to an old stream with smallest temporal difference, the modified Algorithm 5.1 still has the data integrity and maximizes the merging performance for each merged stream. In the rest of this chapter, we still refer to the new version of Algorithm 5.1 as Algorithm 5.1.

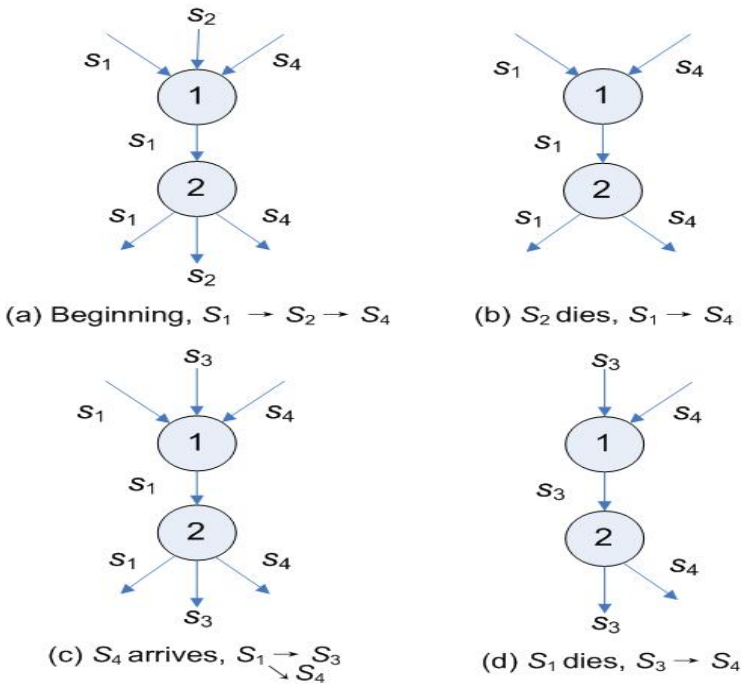


Figure 23. Examples of handling network dynamics

In Figure 23, we illustrate four examples of using Algorithm 5.1 and Algorithm 5.2 to handle the network dynamics. In this example, we consider 4 streams s_1, s_2, s_3, and s_4. We label the subscripts of the streams according to the order of their start time, in which s_1 starts first. At the beginning, we assume there are three streams s_1, s_2, and s_4 on link <1, 2> and s_3 is not passing through this link. Algorithm 5.1 produces the merging chain $s_1 \rightarrow s_2 \rightarrow s_4$. As illustrated in Figure 23(a), node 1 only sends s_1 to node 2 while node 2 reuses the data from s_1 for s_2 and s_4 based on the merging relationships. Sometime later, s_2 is dead at node 1 (Figure 23(b)). According to Algorithm 5.2, the affected mergers $s_1 \rightarrow s_2$ and $s_2 \rightarrow s_4$ are cancelled at link <1, 2>. Since s_4 is now set as unmerged, it is re-merged by s_1 (i.e., $s_1 \rightarrow s_4$) as illustrated in Figure 23(b).

Suppose at this moment, s_3 arrives at link <1, 2> as seen in Figure 23(c). Algorithm 5.1 lets s_1 merge s_3. Since s_4 is already merged by s_1, there is no need to remerge s_4 to s_3, although s_4 is closer to s_3. As a result, we have both s_3 and s_4 merged to s_1 in Figure 23(c), i.e., merging relationships may have a tree topology. In the last example (Figure 23(d)), we assume s_1 dies at link <1, 2> after s_3 arrives. In this case, the nodes use Algorithm 5.2 to cancel the affected mergers $s_1 \rightarrow s_3$ and $s_1 \rightarrow s_4$. Algorithm 5.2 sets both s_4 and s_3 as unmerged and invokes Algorithm 5.1 which returns the merger $s_3 \rightarrow s_4$. Since s_3 is currently blocked at node 1 (Figure 23(c)), node 1 needs to unblock the flow by resuming the forwarding of s_3 at link <1, 2> as shown in Figure 23(d). Finally, s_4 reuses the data from s_3 at node 2.

Theorem 3. Algorithm 5.2 maintains the data integrity in the network.

Proof: According to Theorem 5.1, Algorithm 5.1 ensures the data integrity in the network. Since Algorithm 5.2 uses Algorithm 5.1 to remerge the unmerged streams after the cancellation, Algorithm 5.2 maintains the data integrity in the network. □

5.2.4 **Merging Tree**

As shown in Figure 23(c), DSM constructs merging relationships as a tree topology. In general, a stream can only be merged by one stream at any time while it can be the acquiring stream in many mergers simultaneously. Motivated by this fact, we propose a *Merge Tree* (M-tree) structure to represent the merging relationship and facilitate the stream merging at each mesh node. Each mesh node uses an incoming M-tree (*i*M-tree) and an outgoing M-*tree* (*o*M-tree) to record information about each incoming stream and each outgoing steam, respectively. The root node of an M-tree represents the actual stream being received or transmitted. In other words, only the root node of an M-tree is unmerged. An edge (including a parent node and a child node) in the M-tree represents a merger, where the acquiring stream and the mergee stream are the parent node and child node respectively. The tree structure informs the mesh node that the streams corresponding to the non-root nodes have been merged. These merged streams need

to reuse the video data from the stream corresponding to the root node. That is, they share a "multicast" stream.

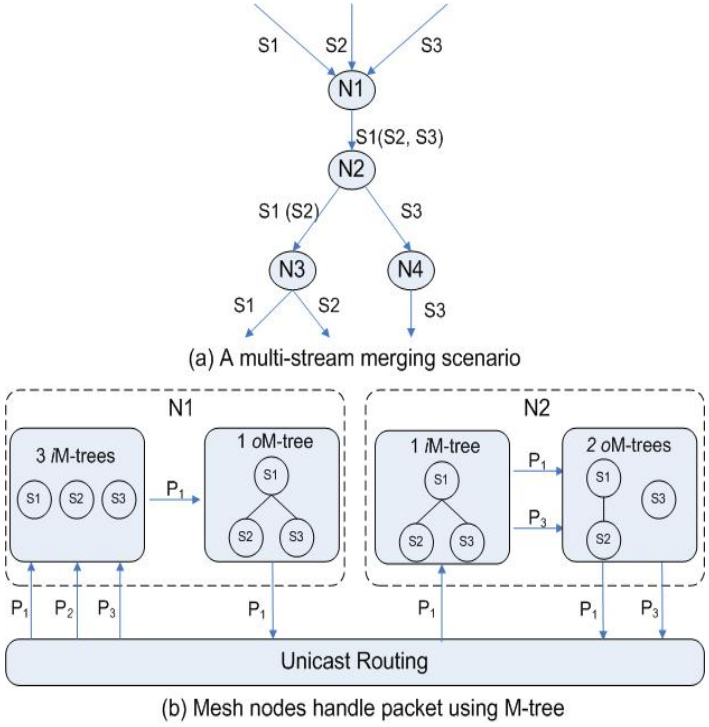


Figure 24. Example of packet forwarding using M-tree

Data forwarding at each mesh node is performed according to its *o*M-trees as follows. Only the streams indicated in the root nodes of the *o*M-trees need to be forwarded. An example is illustrated in Figure 24. As shown in Figure 24(a), N1 has three incoming streams S1, S2 and S3 from three different upstream nodes. These three streams share the same next hop, N2. After passing through N2, S1 and S2 go to N3, and S3 goes to N4. In this scenario, the three video streams are merged at the link between N1 and N2. S1 and S2 continue to be merged at the link between N2 and N3, while stream S3 is diverted to N4. These activities are accomplished with the help of the M-trees as follows. As illustrated in Figure 24 (b), both N1 and N2 have the DSM component running atop certain unicast routing protocol. P_i (i =1, 2, 3) denotes the packets of the video segment being forwarded for stream Si (i = 1, 2, 3). The three *i*M-trees in N1 indicate that it has three incoming streams. However, it has only one *o*M-tree, N1 needs to forward data only for S1 as indicated in the root note of this *o*M-tree. We note that the *i*M-tree of N2 is the same as

the *o*M-tree of N1. This allows N2 to interpret the incoming stream as a multicast stream for merger S1 and also its mergees S2 and S3. Since there are two *o*M-trees at N2, it needs to forward data for the two streams S1 and S3 indicated in the roots of these two oM-trees. Packet P_1 is forwarded on stream S1. The *o*M-tree indicates that this packet is also intended for the mergee S2. N2 also forwards a copy of this packet, P_3, on stream S3. We do not show P_2 as an outgoing packet in Figure 24 to make the distinction between a packet for an actual merger stream and a packet for a mergee stream.

5.3 Design of DSM

In this section, we investigate the design of DSM. We present the data structures, protocols, and algorithms that help realize the ideas discussed in Section 5.3.

5.3.1 General Requirement and Data Structure

DSM can be implemented either as a software module on top of the unicast routing protocol or as an extension (e.g., a software patch) of the unicast routing protocol in WMNs. DSM requires the next hop id for each video stream from the routing layer. This information can be provided by most unicast routing protocols used in WMNs, no matter the protocol maintains a complete route in mesh nodes or it forwards packets in a hop by hop manner. Since DSM works in the network layer, it should be able to recognize the segment ID, stream ID, and video ID associated with each video packet in the network layer. However, in the current IP networks, it is even challenging to recognize the video packet in the network layer, not to mention to identify the IDs associated with this video packet. In this section, we assume these requirements can be satisfied. Please refer to Section 5.4 for our solutions to fulfill these requirements in a real system implementation.

The core data structure of DSM is a *Session Table*. Each entry of the session table represents an stream passing through the mesh node. Table 11 gives the important fields of the Session Table. Given $y \rightarrow x$ on link <*i*, *j*>, after node *i* sends the merge notification "$y \rightarrow x$" to node *j*, the status of stream x at node *i* is set to "*merged*." The other possible values for the status of a stream are "*unmerged*" and

"*dead*" as we have discussed in Section 5.2. Using the same example, the *a_stream* of stream *x* at node *i* is set to stream *y,* and the *mergee_list* records the list of streams merged by stream *x*. Since we have $y \rightarrow x$, *x* is added to the *mergee_list* of stream *y* at node *j*. In other words, the *a_stream* field and the *mergee_list* field are used to represent the *i*M-tree and *o*M-tree at the upstream node *i* and the downstream node *j* of each link <*i*, *j*>, respectively. The *stream_id* field contains the stream ID which uniquely identifies a stream in the WMNs. The descriptions of the other fields are given in Table 11 . They were already discussed in details previously.

Table 11 Important Fields in the Session Table

vid	video ID
src	source address and source port number of the session
dest	destination address and destination port number
stream_id	the stream ID
cur_sid	current (highest) segment ID observed in the mesh node
tau_sid	tau value used is merging the streams
status	status of this session (merged, unmerged, or dead)
next_hop	address of node in the next hop of the stream
a_stream	the acquiring stream of this stream
mergee_list	list of streams merged by this stream
seg_rate	data rate estimated for this stream
buffer	FIFO queue used for caching data for this stream

5.3.2 **Control Operations**

We propose two operations, namely the *Merge Operation* and the *Cancel Operation*, to facilitate the control operations discussed in the merge algorithm (Algorithm 5.1) and cancel algorithm (Algorithm 5.2).

In the merge operation, the upstream node notifies the downstream nodes about the intension of merging two streams on the link. Similarly, the upstream node notifies the downstream node to cancel the

merging in a cancel operation. In both operations, the downstream node acknowledges the upstream node to confirm the success of the operation. The operation will timeout if the upstream node does not receive the acknowledgement before certain deadline. Several merge operations and cancel operations can be batched into one control message to reduce the communication overhead.

Consider a merge operation for a merger $y \to x$ at link <*i*, *j*>. When the merge operation succeeds, *i* and *j* update relevant fields in the session table as follows. Node *i* sets the *a_stream* field of stream *x* as *y*, changes x's status as merged, and set the *ter_sid* of *x* as y's *cur_sid*. The *o*M-tree rooted as *x* (*x* was not merged before the operation) is merged to the *o*M-tree that contains stream *y* (the status of *y* can be merged or unmerged). The downstream node *j* adds *x* into *y*'s *mergee_list*. This means the *i*M-tree rooted as *x* is merged into the *i*M-tree containing y. Similarly, consider a cancel operation applied to the same merger $y \to x$ at link <*i*, *j*>. When the cancel operation succeeds, *i* and *j* update the relevant fields as follow. Node *i* sets the *a_stream* field of *x* to itself and the status of *x* as unmerged. Stream *x* becomes the root of an *o*M-tree and is ready to be merged or resumed (depending on the result or Algorithm 5.1). Node *j* removes *x* from *y*'s *mergee_list*. This action updates the *i*M-tree containing stream *j* by removing the sub-tree rooted at stream *x*.

5.3.3 **Algorithm for Data Forwarding and Sharing**

We recall that DSM blocks the data forwarding of a merged stream at the upstream node of a link and enables data sharing through a FIFO buffer at the downstream node. The Recursive_Forward function presented in Table 12 is the sketch of algorithm for data sharing and data forwarding in DSM. The input of this function is a stream y, a packet p of stream y, and the recursive level l. For each packet p arriving at the mesh node for a stream y, this mesh node calls this function with the recursive level l set to 0. For each stream x in y's mergee_list, the function makes a copy of p (i.e., p_copy) for x and recursively calls itself with x and p_copy as the input. The level argument l of the recursive call equals to the current level plus one. The recursive calls in the Recursive_Forward function ensure that the buffering and merging will be

carried out at all the streams involved in the M-tree. For each stream y examined by this function, it checks, in Step 4, whether p is an incoming packet from the upstream node (i.e., l = 0) or a copied packet passed from the acquiring stream (i.e,. l > 0) through a recursive call. If p is a copied packet for stream y, the function performs Steps 5 to 9. It appends p to the end of y's FIFO buffer (i.e., y.buffer) in Step 5. In Step 6, if the upstream mesh node has stopped forwarding stream y, the function assigns the packet at the head of the FIFO buffer to the packet p in Step 7; otherwise, the stream y is still arriving from the upstream node and the computation exits the function in Step 9. This function now proceeds to Step 10 to check the status of stream y. If its status is "merged," the function stops forwarding the packet p to the next hop of stream y in Step 11 if p belongs to a segment with an ID greater than the τ value; otherwise, the mesh node should forward p to the next hop in Step 13. We note that the packet forwarded is either the packet just dequeued from the FIFO buffer in Step 7 if l > 0, or the packet received from the upstream if l = 0.

Table 12 Pseudo code of the Recursive_Forward function

Recursive_Forward Input: stream *y*, packet *p* of stream *y*, recursive level *l*
1. For each stream *x* in *y.mergee_list* Do
2. Make a copy of *p* as *p_copy* for *x*
3. Recursive_Forward(*p_copy*, x, *l* + 1)
4. IF (*l* > 0) // *p* is a copy
5. Enqueue_FIFO(*y.buffer*, *p*)
6. IF (*y* stops in upstream node)
7. *p* = Dequeue_FIFO(*y.buffer*) // merging in effect
8. ELSE // y has not stopped, no yet ready to reuse data from buffer
9. RETURN
10. IF (*y.status* == merged && *p.segment_id* > *y.tau_sid*)
11. Drop *p* // merging in effect in next hop, no need to forward *p*
12. ELSE
13. Forward *p*
14. RETURN

5.4 System Design and Implementation

We implemented a wireless mesh access network prototype based on the proposed DSM framework. This system consists of a wireless mesh testbed, an online video site, and video players at the user end. The mobile YouTube site is chosen as the online video site. We use the Linux version of RealPlayer [61] as the video player. The videos at the mobile YouTube site are H.263 encoded. The video streams have an average data rate of about 60 kilobyte per second. Three protocols are used by Mobile YouTube to stream data to RealPlayer, namely *Real Time Stream Protocol* (RTSP) [62], *Real-time Transport Protocol* (RTP) [63], and *RTP Control Protocol* (RTCP) [63]. Specifically, the end user and the video server utilize RTSP to set up the RTP streaming session. RTSP is an application-level protocol designed to work with the RTP protocol. It provides means for choosing delivery channels (e.g., UDP, TCP) and delivery mechanism based upon RTP. For an accepted video request, two RTP streams are sent from the video server to the user. One of the streams contains the video data and the other stream contains audio data. DSM supports data sharing for both video RTP streams and audio RTP streams. For the rest of this section, we refer to both video RTP streams and audio RTP streams simply as RTP streams. During a video session, the end user periodically sends feedback on the quality of the RTP stream to the video server using RTCP.

Since our wireless mesh network supports various types of communication applications besides video streaming, the network must be able to recognize video packets (i.e., RTP packets) and apply the DSM technique only to these packets. This can be achieved as follows. It is easy to intercept an RTSP packet during the initialization of a video session because RTSP packets typically use TCP port 554. Such RTSP control messages, exchanged between the video server and the end users, contain the IP address and the port number of the upcoming RTP stream. Thus, any future packet with this IP address and port number in the header can be recognized as an RTP packet of the video.

In order to share the RTP packet among different video sessions (due to stream merging), the mesh node also needs to know the Segment ID, the Stream ID, and the video ID of an RTP packet. The RTP

header contains fields such as sequence number, frame marker, and timestamp that can identify the payload of an RTP packet. In this work, we choose to use the 16-bit sequence number as the segment ID. However, the sequence number and the timestamp each begin with a randomly initialized value set by the RTP protocol at the video server. This makes it difficult to compare and recognize that two RTP packets from different streams are actually the same (i.e., they carry the same video/audio content). We solve this problem by modifying the sequence number and the timestamp as they pass through the gateway of the mesh network so that both start from zero. There is a 32-bit field in the RTP header called *Synchronization Source Identifier* (SSRC) which uniquely identifies the source of this RTP stream from an online video server. Since streams from different servers may have the same SSRC, we use the source IP address together with the SSRC as the *stream ID*. Notice that the video and audio streams of a video session have different SSRC. Therefore, they have different stream ID's. In particular, the RTP streams of the different video sessions of a given video also have different stream ID's. We need to label these streams with the same video ID (*vid*) in order to facilitate stream merging. This is done as follows. We maintain a URL Table in the gateway to store the URL of each ongoing video session in the network. We note that a URL uniquely identifies a video. In the URL Table, each URL is associated with a unique 32-bit *vid*. The fixed length makes it convenient for including the *vid* in the header to label each RTP packet. When a video is no longer present in the network, its *vid* can be recycled and used for another video in the future. Given a new RTP stream, we discover its *vid* as follows. The corresponding RTSP control messages contain the correlation between the URL of the video and the stream ID (i.e, source IP address and SSRC) of this RTP stream. More specifically, we can find such information by parsing the RTSP "SETUP" packet and the RTSP "SETUP reply" packet. Thus, we can determine the URL for a given RTP stream. Since we can map this URL to the *vid* using the URL Table, we can also determine the *vid* of the RTP stream. Once the *vid* has been identified for the RTP stream, its stream ID (i.e., source IP address and SSRC), source port number, and the *vid* is stored in another table called *Stream Table*.

The proposed idea is implemented in Linux kernel version 2.6.27. The implementation consists of a *DSM gateway* (DSMGW) module and a DSM module. The DSMGW module is a kernel module running on the gateway. The DSM module is installed at all the mesh nodes (including the gateway node).

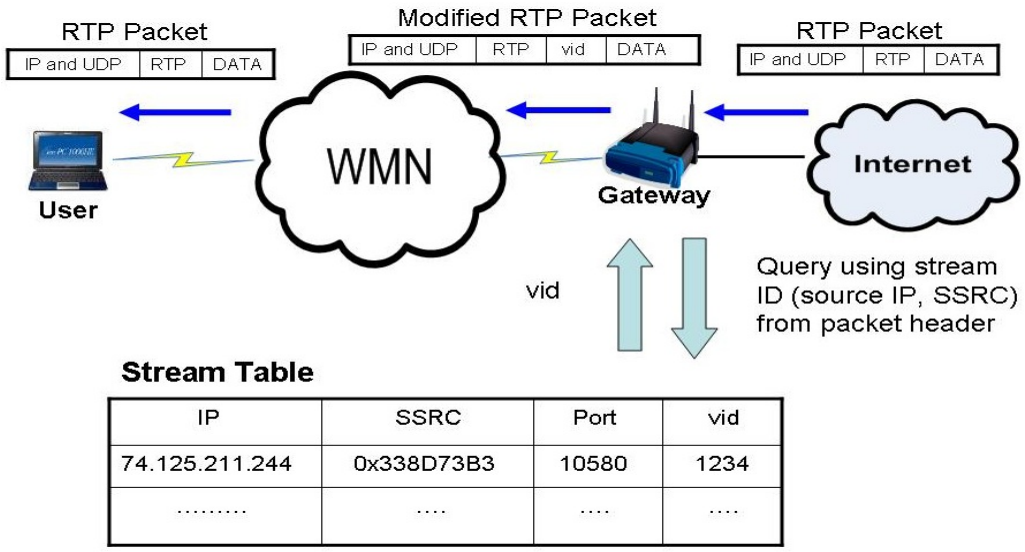


Figure 25. Example of DSMGW

The operation of the DSMGW module is illustrated in Figure 25. It manages the connections between the outside network (i.e., the Internet) and the interior network (i.e., the mesh testbed). In particular, the DSMGW manages the Stream Table as shown in Figure 25. During the initialization of a video session, the DSMGW intercepts and parses the RTSP messages to determine the IP address and the source port number of the source as we have already discussed. Such information is broadcast to all the mesh nodes in the network to help them recognize the upcoming RTP packets. It is also the responsibility of the DSMGW to create a new entry in the Stream Table for this new RTP stream, plus a new entry in the URL Table as necessary. When any packet of this stream arrives at the mesh gateway sometime later, the DSMGW uses the source IP address and the source port number, extracted from the header, to look up the Stream Table. If a matching entry is found in this table, the packet is recognized as an RTP packet. The DSMGW, then looks up the Stream Table using the source IP address and the SSRC (i.e., stream ID), also available from the header of the RTP packet, to find out the *vid* for this packet. This *vid* is appended to the

end of the header before this RTP packet is relayed to the mesh node in the next hop according to the routing protocol. We use AODV routing in our implementation. Other responsibilities of the DSMGW include modifying the sequence number and timestamp of each RTP packet to facilitate video sharing as we have discussed.

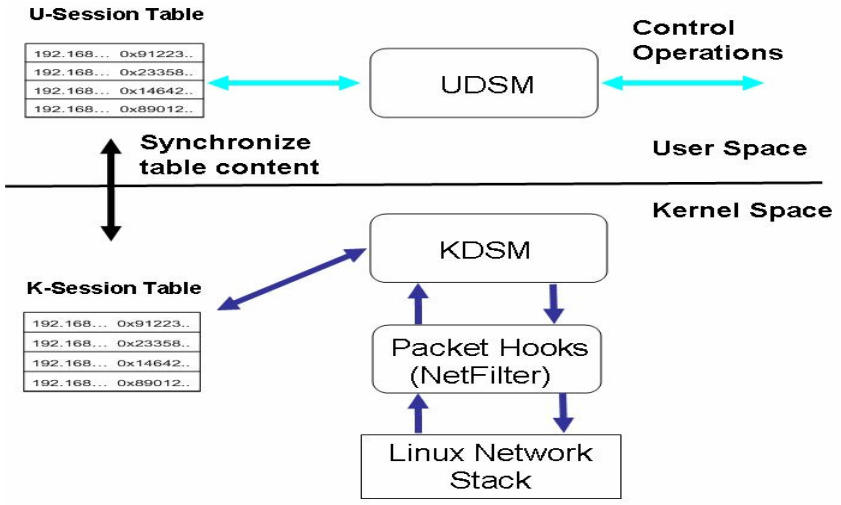


Figure 26. Architecture of the DSM module

The DSM module operates atop of the routing protocol. It consists of two sub-modules, namely the KDSM module and the UDSM module. The KDSM module works in the kernel space and the UDSM module resides in the user space. As depicted in Figure 26, each module maintains a copy of the session table. We denote the session table in the kernel space and user space as *K-Session Table* and *U-Session Table*, respectively.

The KDSM module uses the Netfilter framework [64] to intercept the RTP packet from the Linux network stack. KDSM discovers a new stream if the SSRC and the source IP address (i.e., stream ID) of the RTP packet are not found in the K-Session Table. The KDSM module utilizes the Recursive_Forward function to process the packet. The UDSM module monitors the streams in the U-Session Table. It is responsible for triggering stream merging and cancellation. The UDSM module also queries the underlying routing protocol for the next hop of each stream and stores the information into the U-Session

Table. We periodically synchronize the content of the U-Session Table and the K-Session Table. Therefore, both the user space and kernel space have the latest version the session table.

5.5 Simulation Study

We conducted intensive performance study on DSM by simulation using NS-2.31 [66]. The purpose of the simulation study is to show that DSM can achieve the effect of multicast without any multicast grouping and routing support with a complicated network topology and random settings. All the mesh nodes in the simulation were configured with an omni-directional antenna. The carrier sensing range and transmission range were both set to be 250 meters. The MAC parameters of the wireless nodes were set according to the specifications of in IEEE 802.11b protocol, where the data rate was set to be 11 Mbps. We chose AODV as the unicast routing protocol. The CIF format (352x288 pixels) *akiyo* video clip was used in the simulation. The video was encoded using the MPEG4 encoder provided by ffmpeg [67]. The encoding parameters were the same as those used to encode the videos in the experimental study. Since the video sequence lasts only 10 seconds, we combined several identical video sequences to create a longer video. The video length in the simulation is 100 seconds. DSM was set to allow the merging of two streams with temporal difference less than 10 seconds. Since we repeated simulation for each setting for hundreds of times, the simulation uses shorter video than the experiment does for the sake of a reasonable simulation time.

Figure 27 illustrates the network topology of the simulation. We use a 700x700 meters square to simulate a residential area. The mesh network deployed in this residential area has 13 nodes. Each node is represented by a black circle in the figure. There is a line between two nodes if they are in the transmission range of each other. A 7x7 grid is used to present the square area and label the coordinators of the mesh nodes. N0 is the gateway and is the only source of the video traffics in the simulation. Since we are only interested in the video streaming over the mesh backbone, we let a video request generate at any one of the 12 non-gateway mesh nodes with equal probability. The arriving time of the video request follows the

Poisson arrival. In the simulation, we varied the average inter-arrival time of the Poisson arrival to investigate the different degrees of stress on the network.

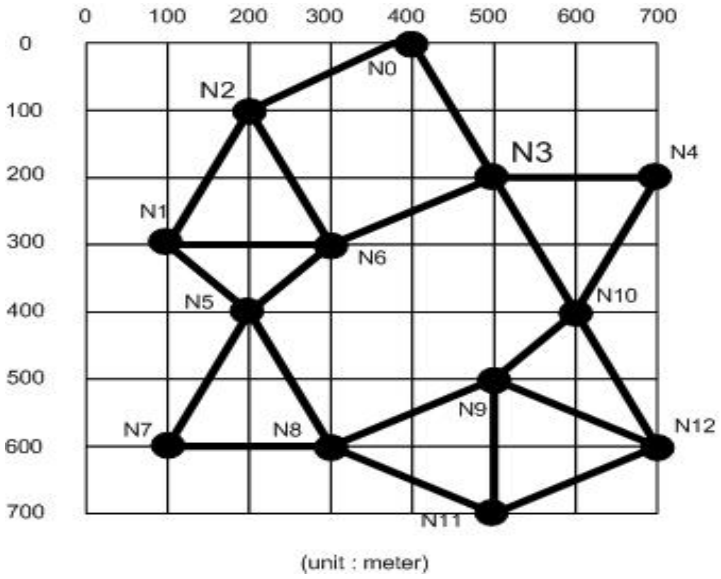
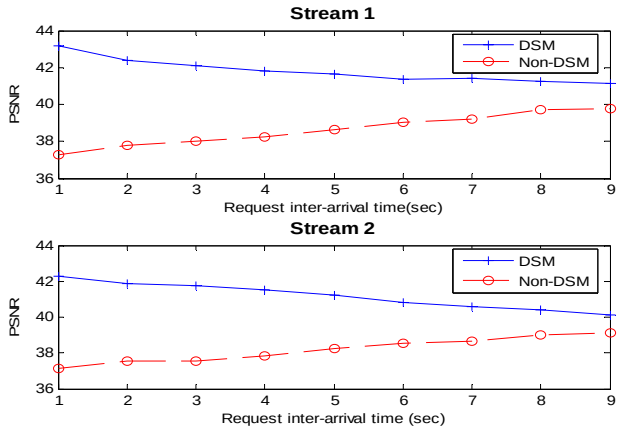


Figure 27. Simulation Topology

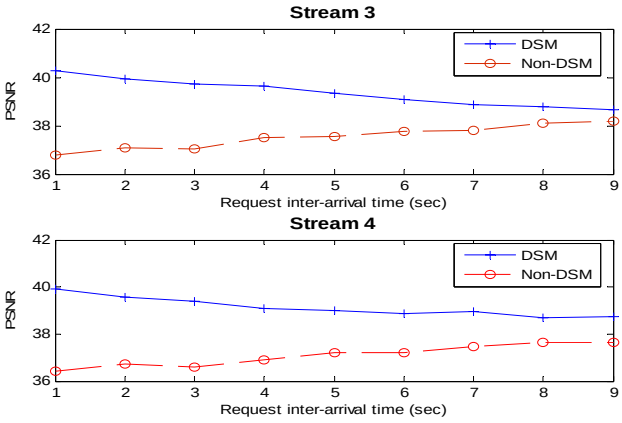
In each simulation run, six video streams were initiated at the gateway node N_0. We varied the average inter-arrival time of the video stream requests (denoted by τ) from 1 second to 9 second. The larger the inter-arrival time, the larger the temporal difference between streams is. We report the simulation results under the scenario with DSM loaded as well as the scenario without DSM (denoted as Non-DSM). The performance data is the average over 500 simulation runs under the same setting. As discussed before, the starting time and destination of each stream are randomly decided for each simulation run.

Our performance metrics are the average PSNR value of the video, the work load and the throughput. The PSNR is used to measure the quality of the video streams. We denote the work load of node Ni (i = 1,..,12) in a simulation run as α_i. α_i is defined as the amount of video data which is supposed to be transmitted from other mesh nodes to Ni. We denote the throughput of node Ni in a simulation run as β_i. β_i is defined as the amount of video data which is successfully transmitted to Ni. We have $\alpha_i \geq \beta_i$. The values of α_i and β_i are calculated from the routing level trace generated by the NS2 simulator. We use

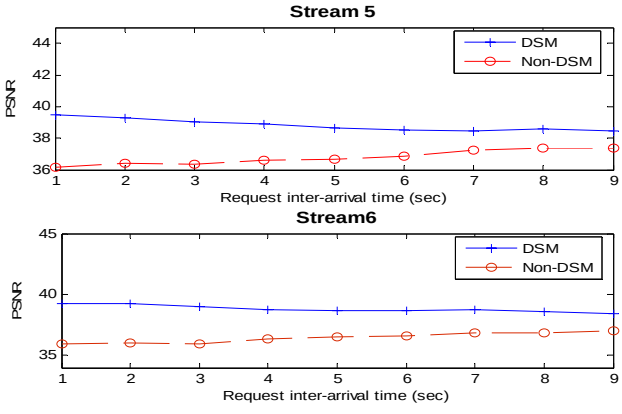
these three metrics to show that DSM can achieve better performance (i.e., higher video quality) and introduce fewer burdens (i.e., smaller work load) to the network than Non-DSM.



(a)



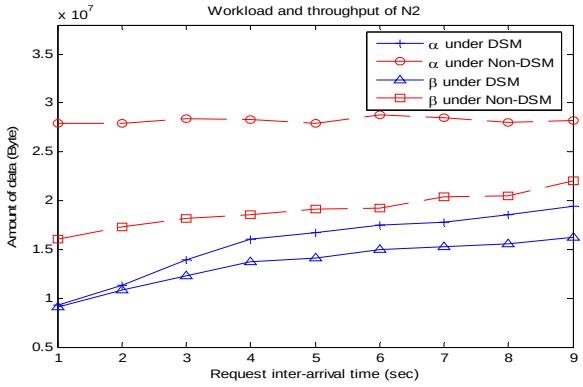
(b)



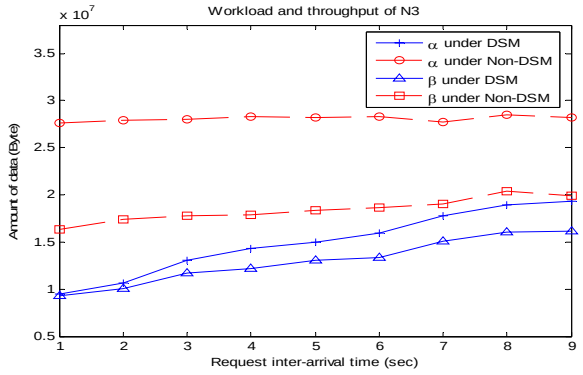
(c)

Figure 28. Average PSNR of the sex streams

Figure 28 depicts the average PSNR values of the video streams under different values of τ. Stream *i* in the figures denotes the stream of the *i*th video request. DSM outperforms Non-DSM in terms of the video qualities of all the streams in the simulation. As τ increases, the PSNR values of DSM and Non-DSM have different trends. The PSNR value of DSM decreases as τ increases. This is because the streams have larger temporal difference under larger τ. There is less chance for DSM to merge the streams with large temporal difference due to the buffer limit. Hence the network becomes more congested and the quality of the video decreases. On the other hand, The PSNR value of Non-DSM increases as τ increases. This is due to a less congested network under larger τ.



(a)



(b)

Figure 29. Work load and throughput at N_2 and N_3

We then measure how much data the network has to send in order to achieve the reported video quality under DSM and Non-DSM. Since all the streams have to go through either N2 or N3, we report the workloads and the throughputs of these two nodes to show the stress on the network. We plot the data of

N2 and N3 in Figure 29(a) and Figure 29(b) respectively. The plots show that DSM has smaller work load and throughput than Non-DSM. The curves in Figure 29 show that the work load and the throughput under Non-DSM does not change too much with different values of τ since there is no stream merging. The work load and throughput under DSM soars as τ increases. This shows that the network becomes more congested when there is less change for merging with larger τ. We define the loss ratio at node Ni as $(\alpha_i - \beta_i)/\alpha_i$. Table 13 lists the loss ratio of N2 under the scenarios of DSM and Non-DSM. We also report the percentage of savings on the work load and the throughout by choosing DSM over Non-DSM in Table 13 . DSM has lower loss ratio than Non-DSM in all the simulations. The data also shows that DSM can save more than 31% of work load and more than 25.95% of throughput in the bottleneck node N2 while achieving better video quality for all the video streams.

Table 13 Comparison of data in bottleneck node N2

τ (sec)	Loss ratio (Non-DSM)	Loss ratio (DSM)	DSM Saves (α_2)	DSM Saves (β_2)
1	42.55%	2.51%	66.72%	43.52%
2	38.15%	4.28%	59.33%	37.06%
3	36.07%	12.22%	50.81%	32.46%
4	34.42%	14.11%	43.35%	25.82%
5	31.48%	15.41%	40.06%	26.01%
6	33.30%	14.21%	39.27%	21.88%
7	28.38%	14.51%	37.51%	25.40%
8	26.92%	15.93%	33.86%	23.91%
9	22.08%	16.08%	31.23%	25.94%

5.6 Experimental Study

We implemented our mesh router using the Asus Eee PC 1005HA netbook computer which consists of an Atom 1.6GHz processor, 2GB RAM, and the Atheros 802.11n wireless card (AR928X chipset). We installed the Fedora 10 operating system (Linux kernel version 2.6.27) and our DSM module in each netbook. One of the netbooks, installed with the DSMGW module, was used as the gateway node.

In the experiment, the gateway was connected to the campus networks through Ethernet cable. The mesh nodes were assigned with static IP addresses and utilized the *domain name system* (DNS) server in the campus network to resolve the URL. The gateway node had the *network address translation* (NAT) service enabled. The AODV-UU (version 0.9.5) [68] was used as the routing software in our experiment. For video clients, we installed RealPlayer video player in the netbooks. The network traffic was recorded using the WireShark [69] packet analyzer.

Figure 30. The line topology in the validation test

We performed experiments, under various scenarios, in order to validate the correctness of the DSM technique and to test the correct operation of the system prototype. We discuss two representative scenarios in this paper. They are based on a line topology as illustrated in Figure 30, which consists of four mesh nodes, where node n_0 is the gateway. The Mobile YouTube is used as the online video source, with a unique URL for each video. We started the playback of the same video at nodes n_1, n_2, and n_3 at different times. Since the video RTP stream carries majority of the data for each video session, we report the amount of *accumulative data transmitted* (ADT) of the video RTP stream of each video session over time to show the effect of stream merging.

We use s_i to denote the video stream played back at node n_i (i = 1, 2, 3). In the first scenario, the three video streams start and stop in the same order, i.e., both starting and ending orders are s_1, s_2, and s_3. The ADT's of the three video streams at links $<n_0, n_1>$ and $<n_1, n_2>$ are plotted in Figure 31 and Figure 32, respectively. In this scenario, s_1 starts first at 81st second, followed by s_2 25 seconds later, and finally s_3 after another 25 seconds. Since we have $s_1 \rightarrow s_2 \rightarrow s_3$ at link $<n_0, n_1>$, n_0 blocks s_2 and s_3 after they reach their τ values. As shown in Figure 31, the ADT values of both s_2 and s_3 stop growing after the streams start for 25 seconds. In contrast, the ADT of s_1 continue to increase. When we terminate the playback of s_1 at

166th second, n_0 senses the death of s_1, cancels the merger $s_1 \rightarrow s_2$, and unblock s_2 at link <n_0, n_1>. The ADT of s_2 now grows again. Similarly, when we terminate the playback of s_2 at 192nd second, the ADT of s_3 also starts to increase again. The ADT's at link <n_1, n_2> for this scenario are plotted in Figure 32. It shows that the ADT of s_3 stops growing 25 seconds after it started. This is due to the merger $s_2 \rightarrow s_3$ at link <n_1, n_2>. Figure 32 also shows that the ADT of s_3 starts growing again when the playback of s_2 is terminated and n_1 unblocks s_3 at link <n_1, n_2>. We note that we do not report the ADT's at link <n_2, n_3> because there is no stream merging at this link.

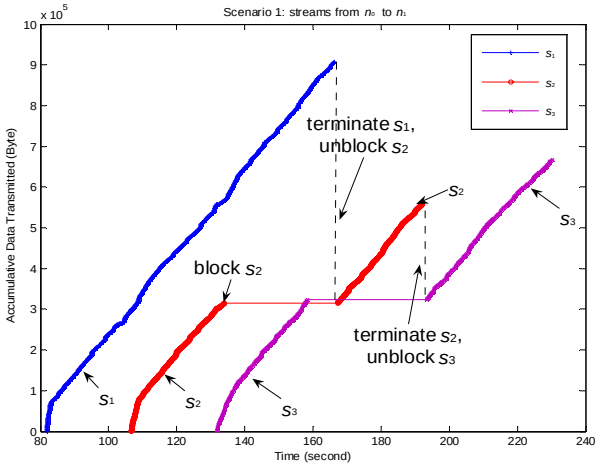


Figure 31. Streams from n_0 to n_1 in scenario 1

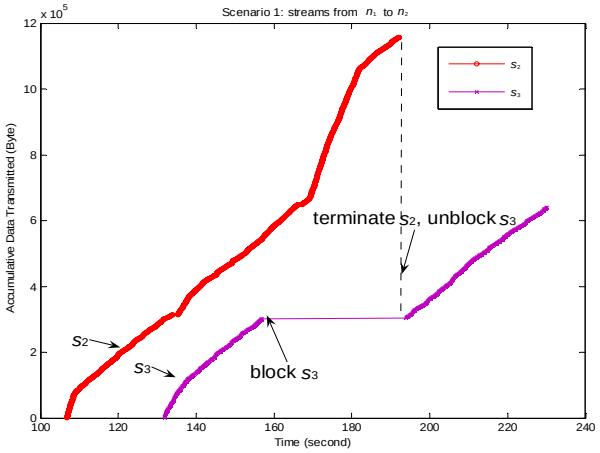


Figure 32. Streams from n_1 to n_2 in scenario 1

The second scenario is more complicated. In this case, the stream s_2 starts first, followed by s_1, and then s_3. After both s_1 and s_3 are merged with s_2 (i.e., $s_2 \rightarrow s_1 \rightarrow s_3$), we first pause the playback of s_2 and then resume its playback sometime later. Finally, the playback of s_1 is terminated, followed by s_3, and then s_2. We note that the starting order and ending order are different. The ADT's of the streams at links $<n_0, n_1>$ and $<n_1, n_2>$ are plotted in Figure 33 and Figure 34, respectively. In Figure 33, the ADT's of s_1 and s_3 stop increasing at 68th second and 93rd second, respectively. This is due to the two mergers, $s_2 \rightarrow s_1$ at 68th second and $s_1 \rightarrow s_3$ at 93rd second. These two mergers result in the merging chain $s_2 \rightarrow s_1 \rightarrow s_3$ constructed at link $<n_0, n_1>$. At 115th second, the playback of s_2 is paused and the video server stops sending data on s_2. When n_0 senses the death of s_2, it cancels the merger $s_2 \rightarrow s_1$ and unblocks s_1. As a result, the ADT of s_1 starts growing again. At 186th second, we resume the playback of s_2, and it falls behind s_1 and s_3 in the playback of the video. Since s_2 is closer to s_3 then s_1, s_2 is merged by s_3. The merging relationships at link $<n_0, n_1>$ become $s_1 \rightarrow s_3 \rightarrow s_2$. In Figure 33, the ADT of s_2 continues to increase for a small duration (from 186th second to 192nd second) and then stops growing. When the playback of s_1 is terminated at 276th second; the ADT of s_3 starts to increase again due to the cancellation of $s_1 \rightarrow s_3$. Similarly, when the playback of s_3 is terminated at 304th second; the ADT of s_2 starts growing again due to the cancellation of $s_3 \rightarrow s_2$. The ADT's associated with link $<n_1, n_2>$ are presented in Figure 34. Similar to Figure 33, the ADT of s_2 stops growing when we pause this stream. When the playback of s_2 is resumed at 186th second, its ADT starts to grow again. Within several seconds, s_2 is blocked again by node n_1 because it is merged by s_3, and it remains blocked until s_3 is terminated near the end of this experiment, as seen in Figure 34. The ADT of s_3 stops increasing at time 93rd second, in Figure 34, because it is merged to s_2 (i.e., $s_2 \rightarrow s_3$). After s_2 is paused, the ADT of s_3 starts growing again which indicates that s_3 is unblocked at link $<n_1, n_2>$. When the playback of s_3 is terminated near the end of this experiment, s_2 is unblocked and the ADT of s_2 grows again.

In summary, we observed in both scenarios the correctness of the merging algorithm and the cancel algorithm. The result also shows that stream merging takes place in a distributed manner using only local information. All the video sessions in our tests showed excellent video playback quality. This is discussed next in terms of frame loss ratio.

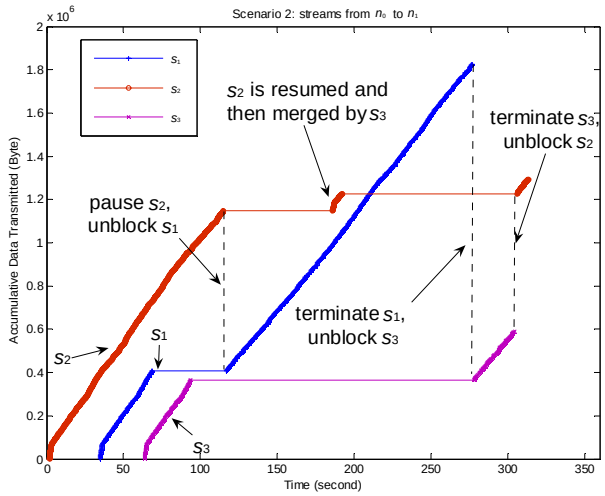


Figure 33. Streams from n_0 to n_1 in scenario 2

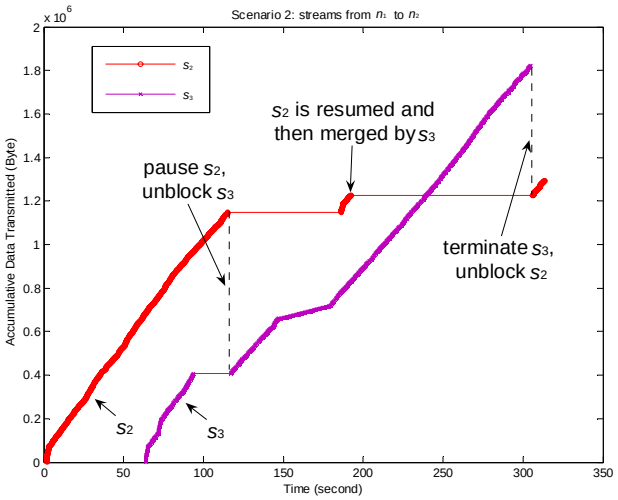


Figure 34. Streams from n_1 to n_2 in scenario 2

Besides the validation test, we also performed stress test to show the efficiency of DSM in saving wireless bandwidth. The network for this test is deployed in the second floor of our department building as

depicted in Figure 35. The black node is the gateway node. The other five white nodes are non-gateway mesh nodes. The performance metric is the *average frame loss ratio* of the video streams (again, we only report the results for the video RTP streams). This ratio represents the quality of the video playback at the user end. RealPlayer reports the frame loss ratio during streaming. If the network is heavily congested, the RealPlayer may assume the network is down and terminates the playback. In this case, we consider the remaining frames in the video as lost and calculate the frame loss ratio accordingly. To avoid other WiFi traffic in the building that can vary for the different test cases, we conducted our experiments after midnight.

In each test setting, a request for the same video, at the Mobile YouTube Website, is initiated every one minute at any one of the five mesh nodes with equal probability. This random process is repeated until the total number of video streams in the wireless network reaches a set number. In our study, we varied this number between 1 and 25, as seen in Figure 36, to investigate the different degrees of stress on the network. For a given number of concurrent video streams, a more robust design should place less stress on the network. We compare the proposed DSM network with a non-DSM environment in this study. Each data point, shown in Figure 36, is the average over ten runs of the corresponding experiment.

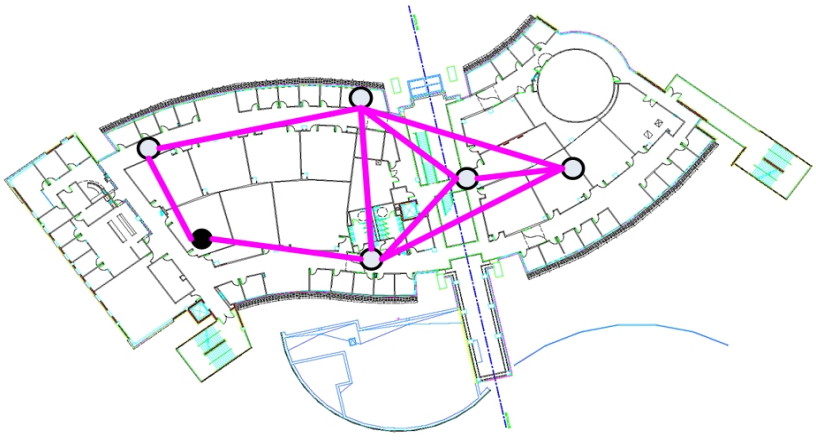


Figure 35. Network topology in the stress test

As depicted in Figure 36, the frame loss ratios for both DSM and Non-DSM techniques are less than 0.1 when the number of streams in the network is below 6. As this number increases, the frame loss

ratio soars all the way up to almost 1 for Non-DSM. This is mainly due to severe congestion in the network. We also observed increases in the number of video sessions terminated by RealPlayer as the number of streams increases. These limitations motivated us to investigate a more robust wireless mesh access network. In contrast, we observe in Figure 36 that the stress on the DSM network is much less (i.e., the frame loss ratio is much lower). In fact, the performance curve of DSM is almost flat. This can be explained as follows. When the number of concurrent streams for a given video is higher than a certain number (8 in Figure 36), there are many opportunities for stream merging, and the additional demand for the same video does not impose more stress on the network. This experimental result validates the robustness of DSM in handling sudden spurts of interest for certain videos due to special events. Under this circumstance, there are still plenty of bandwidth remained for the normal access to regular videos.

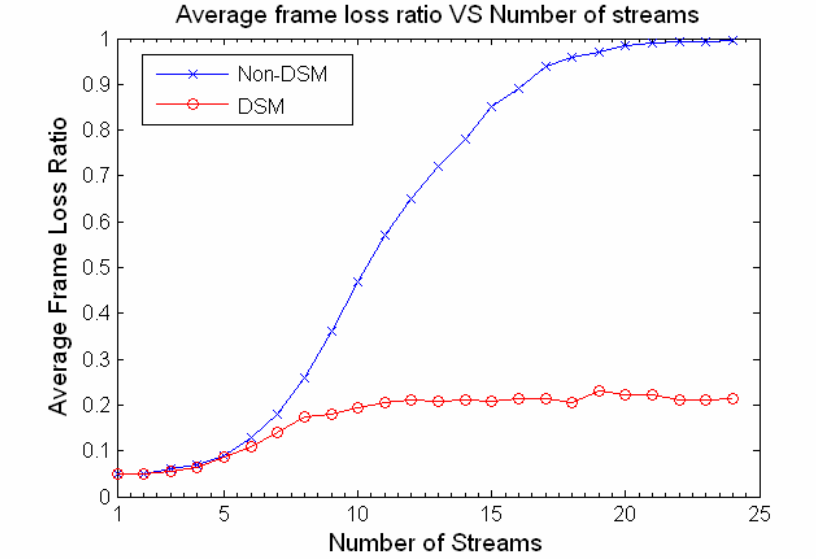


Figure 36. Result of the stress test

5.7 Conclusion

Video streaming has been the subject of intensive research for many years due to its wide range of applications ranging from social networks, electronic commerce, distance learning, to news and entertainment. For a wired environment, reducing the demand on server bandwidth is important to ensure the scalability of the applications. In recent years, we have witnessed a growing demand on online video

resources from wireless access networks. It is highly desirable that such wireless networks are robust in handling sudden spurts in demand for specific videos due to various reasons such as current events. Such situations should not have a significant impact on the normal access to regular videos. Furthermore, a robust wireless access network should be able to sustain applications, such as movies on demand, known to follow the so called 80:20 rule. That is 80 percent of the accesses are for 20% of the videos. The aforementioned requirements can be achieved through video-stream sharing. However, existing video sharing techniques require the cooperation from the video server. On the other hand, it is generally not possible for the online video site to cooperate with the local access networks. In this chapter, we tackle this problem in wireless mesh access networks by proposing a distributed technique called *Dynamic Stream Merging* (DSM). We provided theoretical analysis to show that DSM can achieve optimality locally at each link in terms of video sharing performance, and have low computational and message complexity. The simulation study based on NS2 is conducted to study the performance of DSM under large network settings. We also implemented the proposed technique in a wireless mesh access network prototype, and successfully demonstrated the effectiveness of DSM in supporting wireless resource sharing in accessing videos at Mobile YouTube. To the best of our knowledge, this is the first work that shares video data from a commercial online video site in a wireless mesh network. Our experimental results validate the correctness of DSM and indicate the efficiency of our prototype.

6. HANDOFF FOR VIDEO STREAMING

In this chapter, we study the handoff problem for video streaming in the WMA environment. Traditional handoff approaches focus on maintaining connectivity for mobile users. The connectivity remains a problem for mobile users in the WMA environment. Since the mesh router has limited coverage, user has to update the mesh router it attaches to when immigrating to the coverage of a new mesh router. A good connectivity is usually sufficient for the requirement of non-real-time best effort traffic. However, it is not adequate for data intensive real-time application such as video streaming. Video streaming requires not only the connectivity guarantee but also the quality of service (QoS) guarantee. Therefore, the design goal of a robust handoff technique for video streaming is to not only provide connectivity to the user, but also preserve the quality of the video during the handoff. Triggers relevant to the QoS should be utilized in the handoff decision making process. Conventional handoff techniques in the WMA environment only consider localized information, i.e, the link quality between the mobile user and the mesh router in the vicinity. The localized information may lead to the selection of a bad mesh router which is not able to provide the QoS guarantee to the mobile user due to the congestion in routes associated with the mesh router. Therefore, when the mobile user evaluates the mesh router, it should take the value of the routing metrics of the route associated with the mesh router into account. Last but not the least, the mesh router should guarantee that there is no video data loss during the handoff. After the mobile user disconnects to its previous mesh router, this mesh router should not drop the data for the mobile user. A redirection scheme is needed to forward the data arriving at the previous mesh router to the new mesh router which the mobile user connects to.

6.1 Introduction

Handoff is one of the critical issues in mobile communication. The most common scenario of handoff happens in the cellular communication system where subscriber station moves from the cell of one base station to the cell of another base station. This scenario is called the inter-cell handoff. Another

handoff scenario in cellular communication system is called intra-cell handoff where the subscriber station changes it channel in a cell. There are two classes of handoff, namely the hard hand-off and the soft hand-off. In hard hand-off, the connectivity to the original base station is broken and only then the connection to the target base station is established. This kind of strategy is called *break-before-make*. The intention of the hard handoff is to switch the cells instantaneously in order to minimize the disruption to the communication sessions of the users (i.e., phone call or data session). On the other hand, during the soft hand-off, the connection to the original cell is retained for a while after the connection to the target cell is made. This strategy is called *make-before-break*. In this case both connections are maintained in the networks during the handoff. The subscriber station chooses to use the one with best signal quality for communication. The advantages of hard handoff are the short processing time and easy hardware implementation. However this approach is not as reliable as soft handoff since if the handoff to the target cell fails, some interruption will occur even if the subscriber station is able to reconnect to the original cell. The soft handoff trades the usage of multiple connections and the complexity of the hardware design for a more reliable handoff experience.

The researches ([70], [71]) on fast handoff in 802.11-based WMN have been conducted in recent years. The handoff is supported in many commercialized mesh products offered by companies such as [72], [73], [74]. Existing works tackle the handoff problem in two different layers, namely the link layer handoff and the network layer handoff. When a mobile user is moving out of the communication range of a mesh router, a link layer handoff is triggered that tries to associate the user to another mesh router, if there is any. Authors in [70] propose a fast link layer handoff scheme in their SMesh framework. The whole network acts as a virtual WLAN for the users. The mesh router monitors the DHCP (Dynamic Host Configuration Protocl) request message periodically broadcasted by the user to measure the link quality to the user. The users in SMesh have the same IP address for default gateway. When handoff takes place, the target mesh router uses the gratuitous ARP (Address Resolution Protocol) to force the user to access its default gateway through the target mesh router. Since SMesh utilizes standard protocols, existing 802.11-enabled portable

devices (e.g., notebook) can use the network without installing any new software and hardware. The network layer handoff reroutes the packet for the mobile user after it associates to a new mesh router. Authors in [71] compare two network layer handoff schemes using TMIP [75] and OLSR [76] respectively. In TMIP, a simple version of mobile IP in the WMN, user has a "home" mesh router. When the user immigrates to other mesh routers (called "foreign" mesh routers), the traffic is redirected from the "home" mesh router to the "foreign" mesh router. OLSR is a "flat" link state routing protocol. In the OLSR-based scheme, the routing table of each mesh router is updated when handoff takes place. Experiment shows that the OLSR-based scheme outperforms the TMIP-based scheme since it avoids the inefficient "triangular routing".

Different methods and metrics are used to evaluate the quality of the link in 802.11 networks. For example, routers in [70] monitor the periodic DHCP request from mobile users for link quality measurement. This approach requires all the routers use the same channel. In [71], since the routers work in the infrastructure mode, the traditional channel scanning approach for WLAN is used. This approach usually utilizes metrics such as *Signal to Noise Ratio* (SNR) and *Received Signal Strength Indicator* (RSSI) to trigger the handoff. Authors in [77] propose that the user continuously monitors the beacons from all the access points using the same channel or the overlapping channels in the neighborhood. They also suggest the handoff should be triggered proactively. The orthogonal channels are only scanned when there is no opportunity to find a good candidate in the same or overlapping channel. Authors in [78] propose to take the qualities of both uplink and downlink into account in order to avoid an asymmetric link.

The existing handoff techniques in WMN focus on connectivity between the mobile users and the mesh routers. The triggers for scanning and the handoff are based on the quality of the connectivity. However, the lower layer triggers such as RSSI and FLR (MAC layer frame loss ratio) can not directly imply the quality of the ongoing video stream at the users end. For example, the video playback may not be affected by a sudden burst of packet loss due to the video buffered at the video player. Moreover since the streaming traffic has variable bit rate, the variance of the data rate may not reflect the variance of the

streaming quality. Therefore, it is desirable to have QoS-related handoff triggers that directly measure the quality of video streaming for mobile user.

The handoff in the 802.11 WLAN networks only takes the link quality between the mobile user and the neighboring access points into account. Since the access points are typically connected to the high-speed wired networks, the handoff techniques assume that the bottleneck is the wireless link. Therefore, as long as an access point provides good connectivity to the mobile user, it is a good candidate to switch to. However this assumption does not hold in WMN. By switching to a new mesh router, the user also chooses to utilize the routes associated with this mesh router. We argue that the handoff in the WMN should utilize the routing level information to evaluate the mesh router.

Existing link state routing based network layer handoff is triggered when the mobile node associates to a new mesh router. The new mesh router then starts the link-state update process which propagates the update to the entire network and triggers router recalculation at each mesh router. The old mesh router also sends out the link-state update which indicates the broken link between itself and the mobile user. Since the link-state update is not instantaneous, video data might be dropped at certain mesh router where the route to the mobile user is not available. For example, a mesh node closer to the old mesh node than the new mesh node may get only the link-state update initiated from the old mesh node at certain point of time. This mesh node is not able to forward the video data to its destination. We propose a novel redirection scheme to avoid the data loss during the handoff.

Our contribution in this chapter is threefold. First, we propose to utilize the quality of video at the user as one of the triggers in the handoff. Second, the routing metrics at the mesh routers are considered in the handoff decision making. Third, a redirection scheme is proposed to enhance the network layer handoff for video streaming. The design of the propose handoff technique is presented in Section 6.2.

6.2 QoS Oriented Handoff

Let us model a video stream as a sequence of frames labeled as i = 1, 2, …, L, where L is the total number of frames in the video. Each frame has a sampling timestamp which is calculated during the encoding of the video. Denote S_i as the sampling timestamp of the ith frame in the video stream. When frame i arrives at the video user, its arrival time is denoted as R_i. If frame i has not arrived at the video user, we set $R_i = \infty$. In order to avoid the jittering due to the packet delay, a video player usually buffers certain amount of frames before playing back the video. Without loss of generality, assume frame 1 arrives at the end user in time and the video starts at time T ($T > R_1$), the playback deadline of the ith frame is denoted as $D_i = T + (S_i - S_1)$. The values of T and D_i are reset if there is VCR-like operation which changes the playback time of the video.

Frame i is said a valid frame to the end user if $D_i \geq R_i$. Advanced frame encoding techniques use inter-frame information to facilitate the video frame compression. The frames in the same group of pictures have dependency relation. For example, the decoding of a P or a B frame relies on the data in the I frame of the same group of pictures. Therefore, if frame i is invalid, all the frames depends on frame i is also invalid. Given a time window between T_x and T_y, the video frame loss ratio (VFLR) is defined as:

$$\alpha_{T_x,T_y} = \frac{\left|\{i \mid T_x \leq D_i \leq T_y, i \text{ is invalid}\}\right|}{\left|\{i \mid T_x \leq D_i \leq T_y\}\right|} \quad (6.1)$$

Equation (6.1) shows that the value of VFLR in a time window is the total number of frames whose deadline are within the time window divided by the number of invalid frames whose deadline are within the time widow. VFLR is a metric that represents the quality of the video stream during a given time window. The larger the VFLR, the worse the video quality is. We propose to measure the value of VFLR with a small time window and smooth this value using a time sliding window moving average (TSWMA) filter. The sliding window used in the TSWMA is larger then the time window used to calculate VFLR.

Assume the size of the TSWMA window is *w*, i.e., it keeps *w* recent smoothed values as $\alpha_1, \alpha_2, ..., \alpha_w$, where α_1 is the latest sample . The slope *k* of the smoothed VFLR is

$$k = \sum_{i=1}^{w/2} (\alpha_i - \alpha_{i+w/2}) / \alpha_1 \qquad (6.2)$$

The scanning is triggered if the slope is negative than a predefined threshold. During the scanning phase, the mobile users get the RSSI value to the neighboring mesh nodes as well as the routing metrics in each mesh node. The mesh router includes its routing metrics into the periodic beacon message. The handoff is triggered when the latest smoothed VFLR value is below a predefined threshold. The mobile node selects a candidate set of neighboring mesh routers with good connectivity based on the value of RSSI. It then chooses the mesh router with the best routing metric to the video source as the target mesh router. For example, if the video source is outside the networks, the mobile user chooses the router with best routing metric to its gateway. The link layer handoff and the network layer handoff are taken place after the mobile user makes the handoff decision.

We choose to use the link state routing protocol in the mesh networks. As mentioned in [71], this class of protocol has better performance than mobile IP approach during the handoff. Assume the network layer uses IP-based routing protocol. Consider the case that a mobile user is switching from mesh router A to mesh router B. As mentioned in the in Section 6.1, in a link state routing protocol, it takes time to flood the link-state update from mesh router B to the entire networks. The data for the mobile user might be dropped at mesh router who has not received the new link-state update regarding to the handoff. To tackle this problem, we propose a novel redirection scheme at mesh router A as follows. Before the mobile user associates to the mesh router B, it notifies mesh router A the IP address of the target mesh router (i.e, the IP address of mesh router B). The network layer handoff is triggered after the mobile user switches to mesh router B. During the handoff, mesh router B updates the routing entry to the mobile user and sends the link-state update. Assume the larger the link state value is, the worse the link is. Define the link state between mesh router A and the mobile user before the handoff as l , the value of the shortest path from mesh router

A to mesh router B as p. Instead of reporting its broken link with the mobile user, mesh router A sets the link state to the mobile user as $\alpha \cdot (l + p), \alpha > 1$ and sends the faked link-state update to the networks. The purpose of this mechanism is to solicit video data which will be dropped at the mesh routers without a route to the mobile router. Since we intentionally make the faked link between mesh router A and the mobile user as a bad link, this faked link-state update does not affect the mesh routers who have already received the link-state update from the router B (i.e., the real link state). To forward these solicited data to the mobile user, mesh router A also configures an IP tunnel that redirects all the data for the mobile user to mesh router B. Mesh router B forwards the data received from the tunnel to the mobile user. Mesh router A stops sending the faked link-state update and cancel the tunnel when it receives the link-state update about the mobile user and mesh router B. This is because when the link-state update from mesh router B arrives at mesh router A, a route from mesh router A to the mobile user has been established in the network. Therefore, the duration of redirection scheme at mesh router A depends on the propagation delay of the link-state update from mesh router B to mesh router A.

An example of the proposed redirection scheme is illustrated in Figure 37. In Figure 37 (a), a mobile user associates to mesh router n_3. The video stream is from the gateway and traverses mesh router n_1, n_2, and n_3 before it reaches the mobile user. The topology in the figure indicates that n_3 and n_4 need to communicate with each other through n_1, n_2. Suppose the mobile user switches from n_1 to n_4, and the link state routing is adopted in the networks. When the network layer handoff starts, both n_1 and n_4 send out the link-state update. Suppose at a certain moment of time, the link-state update from n_4 reaches n_1 and the link-state update from n_3 reaches n_2. Figure 37 (b) illustrates the scenario when the proposed redirection scheme is not employed. Since n_1 receives the update from n_4, it starts forwarding the data from the gateway to n_4. However, certain data has been incorrectly routed to n_2 by n_1 due to the delay occurred by the link layer handoff and network layer handoff. Since both n_3 and n_2 have not received the new update from n_4, the incorrectly routed data will be dropped at these two nodes. In Figure 37 (c), the redirection scheme is adopted. In this case, n_3 sends a faked link-state update to n_2. Since n_2 has not received the new

update from n_4, it forwards all the incorrectly routed data to n_3. n_3 sets up a IP tunnel to n_4 and redirects the incorrectly routed data to n_4 through the tunnel. When n_4 receives the data from the tunnel, it forwards the data to the mobile user.

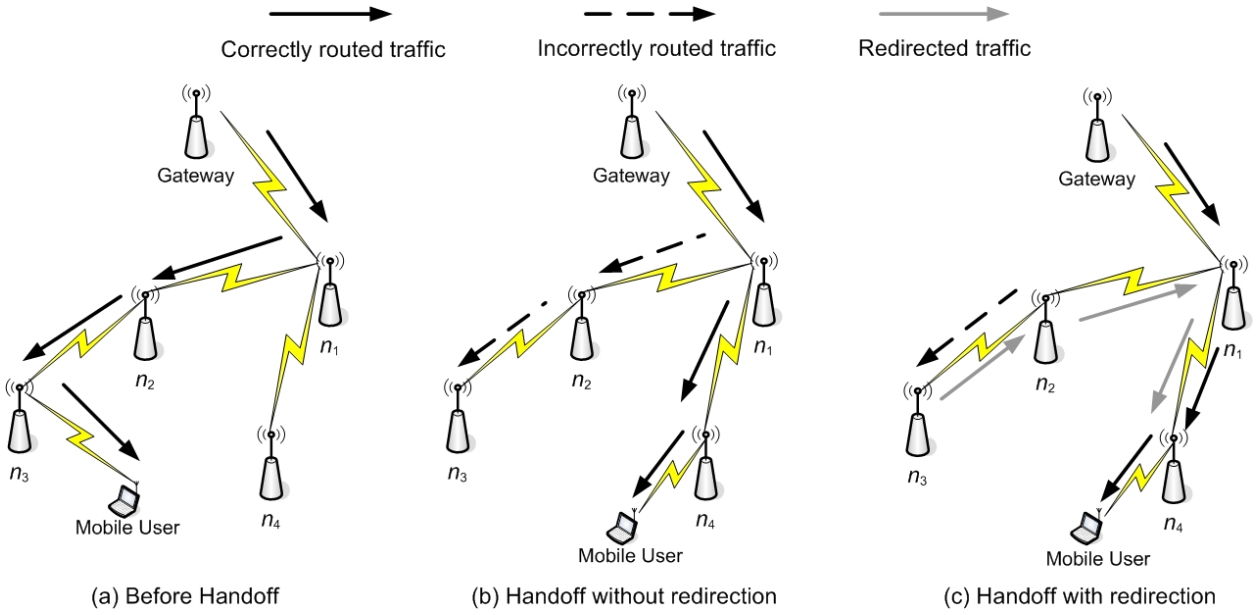


Figure 37. Example of traffic redirection during handoff

7. CONCLUDING REMARKS

In this dissertation, we propose protocols and theoretical insights that lead to a robust wireless mesh access (WMA) environment for mobile video users. The video sharing methodology has been employed throughout this dissertation to help the access network sustain a sudden spurt of requests on popular video in peak time. By considering together channel scheduling, admission control, as well as multicast routing in the WiMax-based WMA environment, we have shown that the cross layer framework can be leveraged to achieve reliable and scalable video-on-demand services in the access networks. Theoretical models have been proposed to formulate the video multicast problem in a general wireless mesh networks. Based on these models, we have proposed algorithms towards to the optimal performance of the Patching-based video multicast in WMA environment. Practical issues are also examined in this work. We propose a Dynamic Stream Merging (DSM) technique to enable sharing of wireless resources without the cooperation from the online video server. As a light-weighted distributed technique, DSM maximizes the per link video sharing performance with small time complexity and message complexity. We implement DSM under Linux system and validate its efficiency in a wireless mesh testbed. We also study the handoff issue for mobile video user and propose a novel QoS oriented handoff technique. We point out that the quality of the video as well as the routing information associate with the mesh router are overlooked in existing handoff technique. Our cross-layer handoff takes the video frame loss ratio, the routing metrics of the mesh routerss, and the link quality to each mesh router into account. A novel redirection scheme is also employed to avoid the data loss in continuous video stream during the handoff.

LIST OF REFERENCES

[1] YouTube, http://www.youtube.com

[2] Hulu, http://www.hulu.com

[3] I. F. Akyildiz, X. Wang, W. Wang, "Wireless Mesh Networks: a survey", Computer Networks, Vol 47, issue 5, pp 445 – 487, March 2005.

[4] Third Generation Partnership Project "RLC Protocol Specification (3G TS 25.332:)", 1999

[5] TIA/EIA/IS-707-A-2.10, "Data Service Options for Spread Spectrum Systems: Radio Link Protocol Type 3", January 2000.

[6] Mobile YouTube, http://m.youtube.com

[7] ITU-T and ISO/IEC JTC 1, "Advanced video coding for generic audiovisual services," ITU-T Recommendation H.264 and ISO/IEC 14496-10 (MPEG-4 AVC), Version 1: May 2003, Version 2: May 2004, Version 3: Mar. 2005, Version 4: Sep. 2005, Version 5 and Version 6: June 2006, Version 7: Apr. 2007, Version 8 (including SVC extension): Consented in July 2007.

[8] Advanced Video Coding for Generic Audiovisual Services, ITU-T Rec. H.264 & ISO/IEC 14496-10 AVC, v3: 2005, Amendment 3: Scalable Video Coding.

[9] K. A. Hua and S. Sheu, "Skycsraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-on-Demand Systems," in Proceedings of ACM SIGCOMM'97, Cannes, France, September, 1997, pp. 89-99.

[10] L. Juhn and L. Tseng, "Harmonic Broadcasting for Video on Demand Services," IEEE Transaction on Broadcasting, Vol, 43, pp. 268-271, September, 1997.

[11] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller,"Construction of an efficient overlay multicast infrastructure for realtime applications," in Proceedings IEEE Infocom, June 2003.

[12] Yi Cui, Baochun Li, Klara Nahrstedt. "oStream: Asynchronous Streaming Multicast in Application-Layer Overlay Networks," in IEEE Journal on Selected Areas in Communications, Special Issue on Recent Advances in Service Overlay Networks, vol. 22, no. 1, pp. 91-106, January 2004.

[13] K. A. Hua, and D. A. Tran, “Range Multicast for Video on Demand,” Multimedia Tools and Applications (2005), 27(3):367–391.

[14] D. A. Tran, K. A. Hua, and T. T. Do, “ZigZag: An Efficient Peer-to-Peer Scheme for Media Streaming,” in Proceedings of IEEE INFOCOM’03, San Francisco, CA, March-April, 2003, pp. 1283-1293.

[15] N. Magharei and R. Rejaie, “PRIME: Peer-to-Peer Receiver-drIven MEsh-based Streaming,” in Proceedings of IEEE INFOCOM, 2007.

[16] Z. Shen, R. Zimmermann, "ISP-Friendly Peer Selection in P2P Networks", in Proceedings of the seventeen ACM International Conference on Multimedia, Beijing, China, 2009.

[17] Z. Li, X. Zhu, A. C. Begen, B. Girod,"Peer-assisted packet loss repair for IPTV video multicast", in Proceedings of the seventeen ACM International Conference on Multimedia, Beijing, China, 2009.

[18] H. Chi and Q. Zhang, “Deadline-aware Network Coding for Video on Demand Service over P2P Networks”, in *Proceedings of the 15th International Packet Video Workshop* (PV06).

[19] Y. Amir, C. Danilov, M. Hilsdale, R. Musaloiu-Elefteri, N. Rivera, "Fast Handoff for Seamless Wireless Mesh Networks," in *Proceedings of ACM MobiSys,* June 2006.

[20] V. Navda, A. Kashyap, and S. R. Das, “Design and evaluation of iMesh: An infrastructure-mode wireless mesh network,” in Proceedings IEEE Int. Symp. World of Wireless, Mobile, Multimedia Netw,, Taormina, Italy, Jun. 2005, pp. 164–170.

[21] S. Mao, Y. T. Hou, X. Cheng, H. D. Sherali, and S. F. Midkiff, “Multi-path routing for multiple description video over wireless ad hoc networks,” in Proceedings of IEEE INFOCOM 2005, Miami, FL, March 13-17, 2005, pp.740-750.

[22] D. Li, Q. Zhang, C. N. Chuah, and S. J. Ben Yoo, "Error Resilient Concurrent Video Streaming over Wireless Mesh Networks", in Proceedings of Packet Video Workshop, April 2006.

[23] X. Zhu, J. P. Singh, and B. Girod, “Joint Routing and Rate Allocation for Multiple Video Streams in Ad Hoc Wireless Networks,” Journal of Zhejiang University, Science A, vol. 7, no. 5, pp. 900-909, May 2006.

[24] G. Liang and B. Liang, "Balancing interruption frequency and buffering penalties in VBR video streaming," in Proceedings of IEEE INFOCOM, May 2007.

[25] X. Zhu, T Schierl, T. Wiegand, and B. Girod, "Video Multicast over Wireless Mesh Networks with Scalable Video Coding (SVC)", Proceedings Visual Communication and Image Processing, VCIP 2008, San Jose, CA, January 2008.

[26] K. A. Hua, Y. Cai, and S. Sheu, “Patching: A Multicast Technique for The Video-on-Demand Services,” in Proceedings of ACM Multimedia’98, Bristol, UK, September, 1998, pp. 191-200.

[27] S. Sheu, K. A. Hua, and W. Tavanapong, “Chaining: A Generalized Batching Technique for Video-on-Demand Systems,” in Proceedings of the IEEE ICMCS’97, Ottawa, CA, 1997, pp. 110-117.

[28] D. L. Eager, M. K, Vernon, J. Zahorjan, “Minimizing bandwidth requirements for on-demand data delivery,” IEEE Transaction on Knowledge Data Engeering 13(5):742-757.

[29] J. Janotti, D. Gifford, K. Johnson, M. Kaashoek, and J. O’Toole, “Overcast: Reliable multicasting with an overlay network,” in Proceedings of the 4th Symposium on Operating Ssytems Design and Implementation, San Diego, CA, USA, 2000, pp. 197-121.

[30] T. T. Do, K. A. Hua, M. Tantaoui. “P2VoD: Providing Fault Tolerant Video-on-Demand Streaming in Peer-to-Peer Environment,” in Proceedings of the IEEE International Conference on Communications(ICC’04), Paris, June 2004, Vol 3, pp. 1467-1472.

[31] PPLive, http://www.pplive.com

[32] PPStream, http://www.ppstream.com

[33] CoolStreaming, http://www.coolstreaming.us

[34] M. van der Schaar and D. Turaga, "Cross-layer packetization and retransmission strategies for delay-sensitive wireless multimedia transmission," IEEE Trans. Multimedia, vol. 9, no. 1, pp. 185-197, Jan. 2007.

[35] S. Krishnamachari1, M. Schaar, S. Choi, and X. Xu, “Video streaming over wireless LANs: A cross-layer approach,” in Proceedings of Packet Video Workshop, 2003.

[36] D. Hui, Si Mao, J. Reed, “On Video Multicast in Cognitive Radio Networks,” In Proceedings of IEEE INFOCOM 2009.

[37] R.M. Karp, “Reducibility among combinatorial problems,” In Complexity of computer computations, Plenum Press, New York, 1975, pp.85–103.

[38] X. Li, S. Tang, and O. Frieder, “Multicast Capacity for Large Scale Wireless Ad Hoc Networks,” in Proceedings of ACM MOBICOM’07, 2007.

[39] M. Cagalj, J. P. Hubaux, C. Enz, “Minimum-energy broadcast in all-wireless networks: NP-completeness and distribution issues,” in Proceedings of MOBICOM’02, Atlanta, Georgia, Spet. 23-28, 2002, pp. 172-182.

[40] S. Yang and J. Wu, "New Technologies of Multicasting in MANET," Design and Analysis of Wireless Networks, Y. Pan and Y. Xiao (eds.), Nova Science Publishers, 2005.

[41] C.Wu and Y. Tay. “AMRIS: AMulticast Protocol for Ad hocWireless Networks,” in Proceedings of IEEE MILCOM, November 1999.

[42] E. M. Royer and C. E. Perkins, “Multicast Operation of the Ad Hoc On-Demand Distance Vector Routing Protocol,” in Proceedings of ACM MOBICOM, Aug. 1999, pp. 207–218.

[43] K. Chen and K. Nahrstedt, “Effective Location-Guided Tree Construction Algorithms for Small Group Multicast in MANET,” in Proceedings of IEEE INFOCOM, 2002, pp. 1180–1189.

[44] S.-J. Lee, M. Gerla, and C.-C. Chiang. “On-Demand Multicast Routing Protocol,” in Proceedings of IEEE WCNC, September 1999.

[45] J. J. Garcia-Luna-Aceves and E.L. Madruga, “The Core-Assisted Mesh Protocol,” IEEE JSAC, Aug. 1999, pp. 1380–1394.

[46] C.-C. Chiang, M. Gerla, and L. Zhang, “Forwarding Group Multicast Protocol (FGMP) for Multihop, Mobile Wireless Networks,” AJ. Cluster Comp, Special Issue on Mobile Computing, vol. 1, no. 2, 1998, pp. 187–196.

[47] L. Ji, and M. S. Corson, “Differential Destination Multicast — A MANET Multicast Routing Protocol for Small Groups,” in Proceedings of IEEE INFOCOM, 2001, pp.1192–1102.

[48] J. G. Jetcheva and D. B. Johnson. “Adaptive Demand-Driven Multicast Routing in Multi-Hop Wireless Ad Hoc Networks,” in Proceedings of ACM MobiHoc, October 2001.

[49] S. Roy, D. Koutsonikolas, S. Das and Y. C. Hu, "High-Throughput Multicast Routing Metrics in Wireless Mesh Networks", in Proceedings of the 26th IEEE International Conference on Distributed Computing Systems (ICDCS'06), page 48, 2006.

[50] IEEE Std 802.16-2004. IEEE Standard for Local and metropolitan area networks Part 16: air interface for fixed broadband wireless access systems. Oct. 1, 2004.

[51] Y. Cai, K. A. Hua and K. Vu, “Optimizing Patching Performance”, In Proceedings of ACM/SPIE Multimedia Computing and Networking, Jan. 1999, pages 204-215.

[52] S. Sen, L. Gao, J. Rexford, and D. Towsley, “Optimal Patching Schemes for Efficient Multimedia Streaming”, Proceedings 9thInt’l Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV’99), Basking Ridge, NJ, June 1999.

[53] M. Cao, V. Raghunathan and P.R. Kumar, “A tractable algorithm for fair and efficient uplink scheduling of multi-hop wimax mesh networks”, in Proceedings of WiMesh 06, page: 93-100. 2006.

[54] D. P. Heyman and T. V. Lakshman, “Source models for VBR broadcast-video traffic”, IEEE/ACM Transactions on Networking (TON), v.4 n.1, p.40-48, Feb. 1996.

[55] X. Cheng, X. Huang, D. Li, W. Wu, and D.-Z Du, “Polynomial-Time Approximation Scheme for Minimum Connected Dominating Set in Ad Hoc Wireless Networks,” Networks, Vol. 42, No. 4, pp. 202-208, 2003.

[56] K. Alzoubi, X. Li, Y. Wang, P. Wan and O. Frieder, “Geometric Spanners for Wireless Ad Hoc Networks”, IEEE Transaction on Parallel and Distributed Systems, Vol. 14, NO.5, May 2003.

[57] J. Wu, S. Yang, and F. Dai, "Iterative Local Solutions for Connected Dominating Set in Ad Hoc Wireless Networks," accepted to appear in IEEE Transactions on Computers.

[58] B. N. Clark, C. J. Colbourn and D. S. Johnson, “Unit Disk Graphs,” Discrete Mathematics, Vol.86, 1990, pp. 165-177.

[59] S. Butenko, X. Cheng, C. Oliveira, and P. M. Pardalos. “A new heuristic for the minimum connected dominating set problem on ad hoc wireless networks,” In S. Butenko, R. Murphey, and P. Pardalos, editors, Recent Developments in Cooperative Control and Optimization, pages 61-73. Kluwer Academic Publishers, 2004.

[60] L. Kou, G. Markowsky and L. Berman, “A fast algorithm for Steiner Trees,” Acta Informatica, no. 15, vol. 2, 1981, pp. 141-145.

[61] RealPlayer, http://www.real.com/linux

[62] RTSP, RFC 2326: Real Time Stream Protocol

[63] RTP/RTCP, RFC 3550: RTP: A Transport Protocol for Real-Time Applications

[64] Netfilter, http://www.netfilter.org/

[65] Wireshark, http://www.wireshark.org/

[66] The Network Simulator – NS2, http:// www.isi.edu/nsnam/ns/

[67] ffmpeg, http://ffmpeg.org/

[68] AODV-UU, http://core.it.uu.se/core/index.php/AODV-UU

[69] Wireshark, http://www.wireshark.org/

[70] Y. Amir, C. Danilov, M. Hilsdale, R. Musaloiu-Elefteri, and Nilo Rivera, "Fast Handoff for Seamless Wireless Mesh Networks," in Proceedings of ACM MobiSys 2006, pp 83 – 95. Sweden 2006.

[71] V. Navda, A. Kashyap, and S. R. Das, "Design and Evaluation of iMesh: an Infrastructure-mode Wireless Mesh Network," in Proceedings of the Sixth IEEE International Symposium on World of Wireless Mobile and Multimedia Networks (WOWMOM 2005), pp 164 – 170, Washington D.C., USA, 2005.

[72] "Locusworld," http://locustworld.com.

[73] "Tropos networks," http://www.tropos.com.

[74] "Cisco," http://www.cisco.com/en/US/netsol/ns621/networking_solutions_package.html.

[75] Transparent Mobile IP, http://www.slyware.com/projects_tmip.shtml.

[76] T. Clausen and P. Jacquet, "Optimized link state routing protocol (OLSR)." RFC 3626, October 2003.

[77] V. Mhatre, K. Papagiannaki, "Using Smart Triggers for Improved User Performance in 802.11 Wireless Networks," in Proceedings of MobiSys 2006, pp. 246 – 259, Sweden, 2006.

[78] H. Wu, K. Tan, Y. Zhang, and Q. Zhang, “Proactive Scan: Fast Handoff with Smart Triggers for 802.11 Wireless LAN,” in Proceedings of IEEE INFOCOM 2007, pp. 749 – 757, AK, USA 2007.