Stony Brook University

Network Coding for Application Layer Multicast

A Dissertation Presented

by

Min Yang

to

The Graduate School

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Electrical Engineering

Stony Brook University

December 2009

Stony Brook University

The Graduate School

Min Yang

We, the dissertation committee for the above candidate for the Doctor of Philosophy degree, hereby recommend acceptance of this dissertation.

Yuanyuan Yang – Dissertation Advisor
Professor, Department of Electrical and Computer Engineering

Sangjin Hong – Chairperson of Defense
Associate Professor, Department of Electrical and Computer Engineering

Dantong Yu
Adjunct Professor, Department of Electrical and Computer Engineering

Esther M. Arkin
Professor, Department of Applied Mathematics & Statistics

This dissertation is accepted by the Graduate School.

Lawrence Martin
Dean of the Graduate School

Abstract of the Dissertation

Network Coding for Application Layer Multicast

by

Min Yang

Doctor of Philosophy

in

Electrical Engineering

Stony Brook University

2009

Today's Internet uses routing to deliver messages from end to end. Network coding is a generalization of routing which allows relay nodes to encode messages in addition to duplicating and forwarding messages. It is proved that network coding can achieve *multicast capacity* and therefore improve the throughput of a multicast network significantly. *Application layer multicast*(ALM) is a perfect candidate to apply network coding due to two reasons: first, ALM is built on peer-to-peer networks whose topology can be arbitrary so it is easy to tailor the topology to facilitate network coding; second, the nodes in ALM are end hosts which are powerful enough to perform complex encoding and decoding operations.

This thesis presents the following contributions to the theory and practice of network coding and its application to ALM. First, we propose a general approach to apply linear network coding to multicast networks. We investigate a series of minimal network coding problems and propose a systematic approach to solve them under a unified framework. Then we apply linear network coding to the peer-to-peer file sharing system and the peer-to-peer media streaming system respec-

tively. These two systems have different features and requirements. For the peer-to-peer file sharing system, we focus on the throughput and reliability. The overlay topology is constructed in such a way that it can be looked as a union of multiple combination networks. We propose a general linear network coding scheme for combination networks and adapt it to the peer-to-peer file sharing system. The simulation shows great improvement in both throughput and reliability compared to other systems without network coding. For the peer-to-peer media streaming system, we focus on the heterogeneity and bandwidth utilization of the access links of peers. We adopt the network model that the bandwidth bottleneck lies only at the edge of the network. The media content is encoded into multiple stripes through *Multiple Description Coding*(MDC). Peers subscribes to the stripes based on their download bandwidths. Random linear network coding is performed within the same stripe. By combining MDC and network coding, peers achieve much higher satisfaction in terms of received downloading rate. Besides, we investigate the inter-session linear network coding problem between multiple simultaneous multicast sessions. Two metrics are introduced to evaluate the network coding benefit based on which we propose a practical inter-session network coding scheme for multicast networks. The system throughput is increased by about 30% in terms of throughput in most cases when compared to intra-session network coding.

Contents

List of Figures

List of Tables

Acknowledgements

I am deeply indebted to my advisor, Professor Yuanyuan Yang, for her support, encouragement and invaluable advice during the course of my Ph.D. study.

I would like to thank Professor Sangjin Hong, Professor Esther M. Arkin and Professor Dantong Yu for serving on my thesis committee. Their invaluable suggestions provide a great resource for the improvement of this thesis.

My thanks also goes to all the collaborators in the High Performance Computing and Networking Research Laboratory at Electrical and Computer Engineering Department, State University of New York at Stony Brook. It has been a wonderful experience working together with them and this thesis benefit greatly from fruitful discussions with them.

Finally, this thesis is dedicated to my parents. Without their unconditional love and persistent support, this thesis would not have been possible.

Chapter 1

Introduction

Many network applications, such as audio/video conferencing, video-on-demand services, e-learning, e-health, distributed interactive simulation, software upgrading and distributed database replication, require *multicast communication*, a basic type of *group communications* involving more than two end hosts, over a large network such as the Internet. In multicast communication, messages from the source are delivered to all the members of a multicast group. The group members are also called receivers. The source can be one of the members in the multicast group or any end host in the network.

The demand for multicast communication from networking applications has been growing at an accelerated pace. Since Stephen Deering first introduced multicast in [1], a lot of works have been done on multicast routing protocols [2, 3, 4, 5, 6, 7]. Most works focus on extending routers to support multicast forwarding in addition to unicast forwarding. This requires routers to build a multicast routing table by exchanging multicast routing information. Usually, multicast routing protocols construct a multicast tree spanning the source and all the group members. The source is the root of the tree and the group members are the leaves. The messages are transmitted from the root to the leaves along the tree. Although multicast routing has been proposed for more than two decades, it is far from being deployed in today's Internet. There are both technical and non-technical reasons [8]. As a result, scalable and efficient support for multicast communication remains to be a critical and challenging issue in networking research.

1.1 Peer-to-peer and Application Layer Multicast

The emerge of *peer-to-peer* technology [9] provides a promising alternative solution for multicast communication. Peer-to-peer is referred to as a fully distributed network architecture which is in contrary to the traditional server-client model. In

server-client model, a server is providing centralized service requested by different clients, i.e., hosts[1]). Usually the address of the server is well known by the hosts in advance. The most obvious drawback of server-client model is its limited bandwidth and resource on the server side. Since the bandwidth and resource of the server is shared by all the hosts, the server can be easily overwhelmed by a huge number of simultaneous hosts. Peer-to-peer model allows hosts to form an adhoc logical overlay network on top of the physical network. The links of the overlay network are logical links each of which can be mapped to a physical path in the physical network. Hosts can share information as well as bandwidth with each other through the overlay network. This requires hosts be able to perform more complex operations such as routing and overlay topology construction/maintenance. The hosts are called peers as they are equivalent in terms of their functionalities. The advantages of peer-to-peer systems are obvious. First, the larger the system size is, the larger the total bandwidth is. As peers can share their bandwidth with each other, peers can contribute their bandwidth to the system. More peers join the system, more bandwidth the system has. Second, the processing is distributed, no *single point of failure* problem. In server-client model, the server is much more important than the hosts. If the server is down, the whole system is down as well. This is called single point of failure problem. In peer-to-peer systems, peers are of equal importance. The overlay network is formed in a distributed way such that the system can continue to work properly even if some peers leave the system.

In practice, a lot of peer-to-peer systems [10, 11] adopt a hybrid model. There is still a server holding the resource which is requested by a lot of hosts. To make the system scalable, the hosts form a peer-to-peer network and help each other to retrieve the resource. As a result, the server can serve much more hosts than that in server-client mode.

Peer-to-peer technology has many applications. In this thesis, we will focus on applying peer-to-peer technology to multicast communication, i.e. *application layer multicast*. Application layer multicast(ALM) [12] is proposed to circumvent multicast support in routers by implementing multicast related functionalities in hosts' application layer. A peer-to-peer network is formed between the source and all the receivers. Then a multicast tree is constructed over the peer-to-peer network. Similarly, the messages are transmitted from the root to the leaves. On the physical network level, the messages are transmitted through unicast along the paths indicated by the tree. As traditional multicast routing implements multicast support in network layer, it is often referred as *network layer multicast*. ALM is a promising alternative for multicast communication over a large scale network. It does not need router support, so it is easy to deploy. It can support infinite receivers in theory.

[1] In this thesis, we use host to represent the end users of the Internet. Every host is connected to a router to access the Internet.

Since receivers help forwarding messages for each other, the more the receivers, the higher the total uploading bandwidth.

1.2 Network Coding for Multicast Networks

Network coding is proposed recently as a generalization of routing [13]. Routing allows relay nodes to forward or duplicate messages. While network coding allows relay nodes to encode messages. Forwarding or duplicating is considered as a special case of encoding. Network coding has a lot of potential applications for both wired and wireless networks [14]. Multicast is one of the most important application of network coding. In [13], the authors prove that with network coding, a multicast network can achieve its maximum throughput. In this thesis, we will focus on applying network coding to multicast networks.

First we use an example to illustrate the advantage of network coding over routing to a multicast network. As shown in Fig. 1.1, node s is the source node, $t1$ and $t2$ are two receivers. All edges in the figure have capacity 1, which means the edge can only transmit 1 unit data (bit) per unit time (second). Source s has two bits, $x1$ and $x2$ to multicast to both $t1$ and $t2$. First we use traditional multicast without network coding as shown in Fig. 1.1(a). Without loss of generality, we use the red flow to represent bit $x1$, the blue flow to represent bit $x2$ and the black flow to represent both bits $x1$ and $x2$. Bit $x1$ can reach $t1$ in two seconds. Bit $x2$ can reach $t2$ in two seconds. When node c receives both bits, it forwards them in sequence. Suppose it forwards bit $x1$ first. Then $t1$ receives both bits in 4 seconds and $t2$ receives both bits in 5 seconds. Now consider using network coding on link cu. When node c receives both bits, it first mixes them by operation XOR. Then it sends the mixed bit to node u. When node $t1$ or $t2$ receives the mixed bit x, it can recover the original bits $x1$ and $x2$ by XOR the mixed bit and the other received bit. All the transmission can be completed in 4 seconds. Therefore the throughput of the network can be increased by $\frac{1}{3}$ using a very simple network coding.

Based on the type of the encoding function at the relay nodes, network coding can be categorized into two types: linear network coding where the encoding functions are linear functions and non-linear network coding where the encoding functions are non-linear functions. Li *et al.* proved [15] that linear network coding is sufficient for a multicast network to achieve its maximum throughput. In the rest of this thesis, we will focus on designing *linear network coding* schemes for ALM systems.

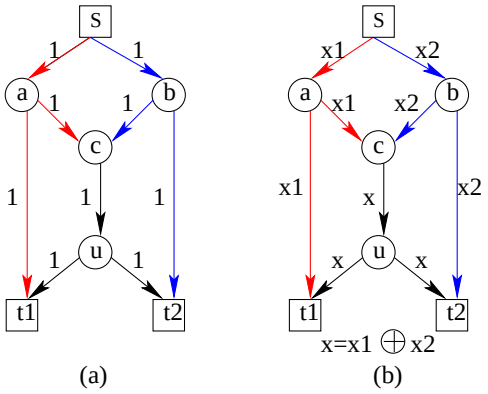


Figure 1.1: Illustration of network coding advantage over routing.

1.2.1 Linear Network Coding for Multicast

In this section, we present some preliminary theories about linear network coding for multicast. A network can be represented as a directed graph $G(V, E, C)$ where V is the set of nodes and E is the set of edges and C is the link capacity function, $C : E \rightarrow \Re^+$. Suppose s is the source, $t_1, ..., t_L$ are L receivers. We first give the definitions of two terms: *flow* and *cut*.

Definition 1 *(Flow)* $F = [F_{ij}, (i,j) \in E]$ *is a* flow *in* G *from* s *to* t_l *if for all* $(i,j) \in E$

$$0 \leq F_{ij} \leq C_{ij}$$

such that for all $i \in V$ except for s and t_l

$$\sum_{i':(i',i)\in E} F_{i'i} = \sum_{j:(i,j)\in E} F_{ij}$$

The value of flow F is defined as

$$\sum_{j:(s,j)\in E} F_{sj} - \sum_{i:(i,s)\in E} F_{is}$$

which is equal to

$$\sum_{i:(i,t_l)\in E} F_{it_l} - \sum_{j:(t_l,j)\in E} F_{t_lj}$$

F is a *max-flow* from s to t_l in G if F is a flow from s to t_l whose value is greater than or equal to any other flow from s to t_l. The max-flow F is the upper bound of

the transmission rate between s and t_l.

Definition 2 *(Cut) A* cut *in G between s and t_l means a collection of H of nodes which includes s but not t_l. An edge ij is said to be in the cut H if $i \in H$ and $j \in \bar{H}$. The value of the cut is the sum of the capacity of all the edges from H to $\bar{H}$.*

Theorem 1 *(Max-Flow Min-Cut Theorem) For every nonsource node t, the minimum value of a cut between the source and the node t is equal to the max-flow between the source and t.*

The Max-Flow Min-Cut Theorem provides a simple way to calculate the max-flow by finding the minimum cut.

In a unicast session, the maximum transmission rate between the source and the receiver is the max-flow between them. In a multicast session, we define *multicast capacity* as the minimum of max-flows between the source and the receivers. If the source must transmit messages to all the receivers at the same rate [2], it is easy to see that the multicast capacity is the upper bound of the transmission rate. Therefore multicast capacity is considered as the maximum throughput a multicast network can achieve.

In their pioneer paper [13], Ahlswede *et al.* prove that with network coding, the maximum throughput can be achieved for a multicast network, i.e. network coding can take full advantage of multicast capacity. In paper [15], Li *et al.* prove that linear network coding is sufficient to achieve this goal and even a stronger result: with linear network coding, each receiver can receive the messages at the rate determined by the max-flow between the source and the receiver simultaneously.

We give an example to explain the notion of linear network coding. Given a graph $G(V, E)$, each node has multiple incoming edges and outgoing edges. An edge e is represented by an ordered node pair (x, y), $x, y \in V$, where y is called the head of the edge and denoted as $y = head(e)$, and x is called the tail of the edge and denoted as $x = tail(e)$. The messages can only be transmitted from x to y. The incoming edge set of a node v is defined as

$$E_{in}(v) = \{(x, y)|(x, y) \in E, y = v\} \tag{1.1}$$

Similarly, the outgoing edge set of a node v is defined as

$$E_{out}(v) = \{(x, y)|(x, y) \in E, x = v\} \tag{1.2}$$

Assume each edge $e \in E$ can carry one symbol from a certain finite field F. Let y_e denote the symbol carried by edge e. Let $\mathbf{x} = (x_1, ..., x_h)$ denote the source

[2] In live streaming systems, receivers can receive the media content at their respective rates. We will discuss this in Chapter 4.

symbols available at source s. For notational consistency, we introduce h source edges, $s_1, ..., s_h$, which all end in s; the source edges $s_1, ..., s_h$ carry the h source symbols $x_1, ..., x_h$ respectively. Since each node encodes the incoming messages with a linear function, the symbol on an outgoing edge is a linear combination of the symbols on the incoming edges, namely

$$y_e = \sum_{e':head(e')=tail(e)} w_{e'e} y_{e'}$$

The coefficients W = $w_{e'e}$ is called *mixing coefficients*. The set of all mixing coefficients can be collectively represented by a $|E| \times (|E| + h)$ matrix. The structural restriction of W is that $w_{e'e} = 0$ unless head(e')=tail(e). Partition W into two parts as

$$W_{|E|\times(|E|+h)} = [A_{|E|\times|E|} \; B_{|E|\times h}], \tag{1.3}$$

where the subscript indicates the size of the matrix.

Equation 1.3 can be rewritten in a matrix form as

$$\mathbf{y} = A\mathbf{y} + B\mathbf{x}$$

Suppose the graph is acyclic [3], we can topological sort the edges. Then the matrix A is a lower triangular matrix with zeros on the diagonal line. Thus (I-A) is invertible and

$$\mathbf{y} = (I - A)^{-1} B\mathbf{x}$$

Therefore, y_e on any edge e is a linear combination of the source symbols. Define $Q_{|E|\times h} \equiv (I - A)^{-1} B$. The e^{th} row of Q is denoted as q_e, which is called the *global encoding vector* at edge e since $y_e = q_e x$.

Since each y_e is a linear combination of the source symbols, any receiver t receiving h symbols with linearly independent global coding vectors can decode the source symbols by solving the corresponding system of linear equations.

The above discussion is summarized in the following definition.

Definition 3 *Linear Network Coding Assignment: Given an acyclic graph G(V,E) a source node s, a finite field F, and a code dimension h, a linear network coding assignment W refers to an assignment of mixing coefficients $w_{e,e'} \in F$, one for each pair of edges (e, e') with $e' \in E \cup \{s_1, ..., s_h\}$, $e \in E$, and head(e') = tail(e). The global coding vectors resulting from a linear network coding assignment W are the set of row vectors of the matrix $Q = (I - A)^{-1} B$, where $W = [A \; B]$.*

In a linear network coding assignment W, the rank of a node v, $rank_v(W)$, refers to the rank of a linear space spanned by the global coding vectors for incom-

[3] If the graph is cyclic, we can find a subgraph which satisfies our requirement.

ing edges of v, i.e.,

$$rank_v(W) \equiv rank\{q_e(W), head(e) = v\} \tag{1.4}$$

For a receiver to decode the received messages successfully, the rank of the receiver must be no less than the coding dimension h. A valid linear network coding assignment is a linear network coding assignment that all the receivers can decode successfully.

Given a network topology, there may be multiple valid linear network coding assignments. We use the network topology shown in Fig. 1.1 as an example. In this example, $h = 2$ and the finite field is F=GF(2). The set of equations of symbols on each edge are

$$\begin{aligned}
y_{e1} &= y_{s1};\\
y_{e2} &= y_{s2};\\
y_{e3} &= y_{e1};\\
y_{e4} &= y_{e1};\\
y_{e5} &= y_{e2};\\
y_{e6} &= y_{e2};\\
y_{e7} &= y_{e3} + y_{e5};\\
y_{e8} &= y_{e7};\\
y_{e9} &= y_{e7};
\end{aligned}$$

The matrix W is

$$W = [A_{9\times 9}\ B_{9\times 2}] = \left(\begin{array}{ccccccccc|cc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
\end{array}\right)$$

The matrix $Q = (I - A)^{-1}B$ is

$$Q_{9\times 2} = (I-A)^{-1}B = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix}$$

With this linear network coding assignment, receiver $t1$ can decode the two source symbols since it observes two symbols with linearly independent global coding vectors [1, 0] and [1, 1]. Similarly, receiver $t2$ can decode the two source symbols as well.

In the above, a linear network coding assignment is defined by specifying the mixing coefficient matrix W . Following theorem gives the sufficient and necessary condition that a linear network coding assignment W can achieve the multicast capacity.

Theorem 2 *A capacity-achieving linear network coding assignment for an acyclic graph G = (V, E) exists if and only if we can assign vectors* $\{q_e\}_{e\in E}$ *to the edges that satisfy:*

$$q_e \in span\{q_{e'},\ head(e') = tail(e), \forall e \in E\}$$

$$rank\{q_{e'},\ head(e') = t\} = multicast\_capacity, \forall t \in T$$

1.2.2 Deterministic Network Coding vs. Random Network Coding

To apply linear network coding to multicast networks is to find a capacity-achieving linear network coding assignment for a multicast network. There are two ways to fulfill this task: deterministic network coding and random network coding.

Deterministic network coding adopts a centralized way to calculate the encoding mixing coefficients for each node in the network given the information about the network topology, source and receivers. Jaggi *et al.* proposed a polynomial deterministic linear network coding construction algorithm in [16]. After the linear network coding assignment is determined, it is distributed around the network. Each node then encodes the incoming messages based on the linear network coding assignment during the whole multicast session. As the linear network coding assignment is calculated centrally based on the complete information about the multicast

session, all the receivers are guaranteed to be able to decode properly. Moreover, the required field size can be as small as the number of receivers [17]. A drawback of deterministic linear network coding is its dependence on the stableness of the system topology. Once the topology is changed, the whole linear network coding assignment needs to be recalculated.

On the contrary, random network coding uses a complete distributed way to determine the encoding mixing coefficients for each node. Each node even does not need to collect any local information. The coefficients is randomly generated for each node. The random coefficients is attached to the corresponding encoded messages. After receiving the encoded messages, the relay nodes will encode again with a set of new generated random coefficients and replace the coefficients in the message with the new ones. At the end, the receivers will try to decode the messages based on the coefficients attached in the messages. Due to the randomness of the coefficients, there is a non-zero possibility that the receiver can not decode successfully. In this case, the receiver has to receive more messages to perform decoding. Obviously, if we increase the field size for the coefficients, the probability of failing to decode is reduced. The strength of random linear network coding is its resilience under dynamic network topology. As each node generated the coefficients independently and randomly, the topology change has no impact on coefficients generation. For details about random linear network coding, please refer to [18].

1.3 Thesis Outline

Fig. 1.2 shows the logical organization of this thesis.

Chapter 2 presents a formal unified approach that can convert the deterministic linear network coding problem for a multicast network into an equivalent graph theory problem. By doing this transformation, we provide a systematic approach to solve a series of minimal multicast network coding problems [19]. A minimal multicast network coding problem is to find a deterministic linear network coding assignment for multicast networks which minimizes some resource usage. To solve this type of problems, we map the network topology graph to a pseudo-dual hypergraph. And a linear network coding assignment is equivalent to a set of nodes called *cover* in the pseudo-dual graph satisfying certain constraints. By iterative refinements, an eligible cover can be found in polynomial time. Moreover, it introduces multiple preprocessing algorithms to further reduce the computation time required by the iterative refinements through reducing the graph size before transformation. Finally by assigning different weights to the edges, minimal network coding problems are reduced to the shortest path problems in the pseudo-dual graph.

Chapter 3 proposes a peer-to-peer file sharing scheme based on network coding called *PPFEED*. The scheme exploits a special type of network topology called

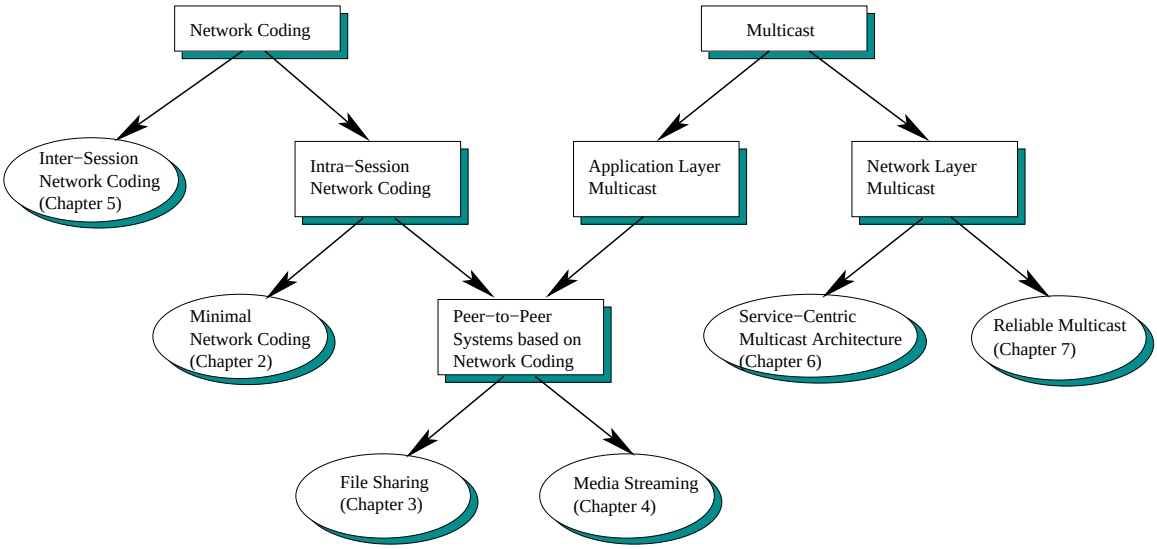


Figure 1.2: Thesis organization.

combination network. It was proved that combination networks can achieve unbounded network coding gain measured by the ratio of network throughput with network coding to that without network coding [20]. The proposed scheme encodes a file into multiple messages and divides peers into multiple groups with each group responsible for relaying one of the messages. The encoding function is designed to satisfy the property that any subset of the messages can be used to decode the original messages as long as the size of the subset is sufficiently large. To meet this requirement, it first defines a deterministic linear network coding scheme which satisfies the desired property, then the peers in the same group are connected to flood the corresponding message, and peers in different groups are connected to distribute messages for decoding. Moreover, the scheme can be readily extended to support link heterogeneity and topology awareness to further improve system performance in terms of throughput, reliability and link stress.

Chapter 4 investigates the optimal overlay construction problem for peer-to-peer streaming systems and proposes a peer-to-peer streaming scheme based on network coding. Media streaming is an important Internet application and has received more and more attentions in recent years. Heterogeneity is a common and important issue for peer-to-peer systems. Like bulk data distribution systems, live media streaming systems usually involve a server which hosts the media content and all the clients request the media content from the server. However, there is a fundamental difference between them. Live media streaming systems require real-time data delivery and can tolerate data loss to some extent. While bulk data distribution systems require

reliable data delivery and can tolerate delay to some extent. This difference leads to different consideration in system design. In this chapter, we consider optimizing the overlay construction for peer-to-peer streaming systems with heterogeneous access link bandwidths. Our goal is to maximize the total downloading rate and satisfy the heterogeneous downloading requirements when the uplink bandwidth is limited. We formalize optimal overlay construction problem into a problem of finding maximum number of edge disjoint trees in a graph which is an abstract model of the peers and their access link bandwidths. A centralized heuristic algorithm is given as a basis for a distributed algorithm which makes it scalable in a network environment. The distributed algorithm constructs an adaptive overlay topology that can adapt itself to the changing peers such that the end-to-end delay and link stress are minimized.

This chapter also proposes a scheme to apply network coding to heterogeneous peer-to-peer media streaming systems. As most peers in a peer-to-peer media streaming system are individual computers connected to the Internet through access links with heterogeneous link capacities, it is desirable to design an adaptive scheme to satisfy heterogenous requirements. Based on the overlay topology construction algorithm discussed previously, we propose an adaptive network coding scheme for heterogeneous peer-to-peer streaming systems. The media content is encoded into multiple stripes. The peers select one or more stripes to subscribe based on their own download bandwidths. The network coding is applied with each stripe. For each stripe, a subgraph is constructed such that the coding probability is maximized. We compare our scheme with another recently proposed scheme called LION [21] through simulations. Our simulation results show that the proposed scheme achieves higher satisfaction and less control overhead compared to LION.

Chapter 5 examines the inter-session linear network coding problem in multicast networks. Previous works on linear network coding for multicast primarily considered encoding the messages in a single multicast session. In this chapter, we consider the inter-session linear network coding for multicast. The basic idea is to divide the sessions into different groups and construct a linear network coding scheme for each group. We use two metrics to guide the group division: *overlap ratio* and *overlap width*. These two metrics measure the benefit that a system can achieve by inter-session network coding from two different aspects. The overlap ratio mainly characterizes the network bandwidth while the overlap width characterizes the system throughput. We also propose two heuristic algorithms, the deterministic algorithm and the random algorithm, to construct the linear network coding on the divided groups.

Besides exploiting the performance potential of application layer multicast through network coding, we have also conducted research on network layer multicast. The

last two chapters are focused on improving the scalability, efficiency and flexibility of network layer multicast from two different aspects.

Chapter 6 presents a new multicast architecture and the corresponding multicast routing protocol for providing efficient and flexible multicast services over a large scale network. Traditional multicast protocols construct and update the multicast tree in a distributed manner, which causes two problems: first, since each node has only local or partial information on the network topology and group membership, it is difficult to build an efficient multicast tree; second, due to the lack of complete information, broadcast is often used when transmitting control packets or data packets, which consumes a great deal of network bandwidth. In the newly proposed multicast architecture, a few powerful routers, called *m-routers*, collect multicast-related information and process multicast requests based on the information collected. The m-routers handle most of multicast related tasks, while other routers in the network only need to perform minimum functions for routing. The m-routers are designed to be able to handle simultaneous many-to-many communications efficiently. The new multicast routing protocol, called the Service Centric Multicast Protocol (SCMP), builds a shared multicast tree rooted at the m-router for each group. The multicast tree is computed in the m-router by employing the Delay Constrained Dynamic Multicast (DCDM) algorithm which dynamically builds a delay constrained multicast tree and minimizes the tree cost as well. The physical construction of the multicast tree over the network is performed by a special type of self-routing packets in order to minimize the protocol overhead.

Chapter 7 discusses improving the multicast performance through transportation layer. Reliable multicast is critical to multicast based applications as it provides reliability over an unreliable network. Although the primary function of reliable multicast is loss recovery and flow control which are similar to that of reliable unicast, the inherent property of multicast that multiple receivers coexist in one multicast session imposes new challenges such as acknowledge implosion and poor scalability. Among existing reliable multicast protocols, tree based reliable multicast protocols can achieve reliability in a scalable fashion. In such protocols, receivers are grouped into a hierarchy called the ACK tree. The ACK/NACK messages and retransmitted packets are transmitted between adjacent levels. Since current tree based reliable multicast protocols construct the ACK tree based on the multicast tree which is constructed by the multicast routing protocol, the protocol performance greatly depends on the multicast tree. In this chapter, we propose a peer-to-peer tree based reliable multicast protocol which constructs the ACK tree in a much more flexible way. Our protocol decouples the ACK tree construction from the multicast tree construction. Any two receivers can be adjacent nodes in the ACK tree. The ACK tree construction process is based on a heuristic function which is designed to minimize retransmission delay. A child node sends ACK/NACK mes-

sages to the parent node and receives retransmitted packets from the parent node. It includes a window based flow control mechanism to avoid fast sender floods slow receiver.

1.4 Thesis Contributions

The thesis contributions are summarized as below.

- It provides a general linear network coding solution for any multicast networks which can be modeled by directed graphs. The approach introduced in this thesis provides a unified way to solve a series of minimal linear network coding problems for multicast. The approach transforms the linear network coding problem into a tractable graph theory problem, and then propose an iterative refinement algorithm to solve the graph theory problem. To the best of our knowledge, this is the first work to give a systematic approach to solving the various minimal linear network coding problems. This is the primary advantage of this method compared with other existing linear network coding construction methods. It proposes several preprocessing algorithms which can be used to find a subgraph of the multicast network before applying any type of network coding approaches. The time complexity of the method is sensitive to the multicast capacity and the number of receivers. The preprocessing algorithms are designed to reduce the computation time by finding a minimal subgraph whose size is much smaller than the original graph. Due to the reduced graph size, both the computation time and the network bandwidth can be saved. The results are discussed in Chapter 2.

- It applies linear network coding to peer-to-peer file sharing and presents a high performance peer-to-peer file sharing system, PPFEED. PPFEED utilizes combination networks as its overlay topology prototype. Combination networks demonstrate great performance gain under linear network coding. It proposes a simple and efficient deterministic linear network coding scheme for combination networks and applies to PPFEED. As a result, PPFEED inherits its high performance when applied network coding and presents its superiority compared to other existing peer-to-peer file sharing systems. Our simulation results show that PPFEED can achieve 15%-20% higher throughput than Narada [12] which is an ALM system without network coding. Besides, it achieves higher reliability and resiliency. It is discussed in Chapter 3.

- It addresses the heterogeneity problem in peer-to-peer streaming system and formalize the problem of optimal overlay construction for peer-to-peer stream-

ing system with heterogeneous downloading requirements. To maximize both the respective satisfaction of the peers in terms of their downloading bandwidth and the system throughput, we formalize it into a problem of finding maximum number of edge disjoint trees in a graph which models the peers and their access link bandwidths. The problem formalization leads to a greedy heuristic algorithm and a practical distributed algorithm to construct the overlay topology under dynamic peer joining and leaving. The overlay topology can adapt itself to the changing peers in the system. Based on the optimal overlay construction scheme, we build a peer-to-peer streaming system and apply network coding to it. As the overlay topology is constructed with the consideration of link heterogeneity, the linear network coding should be adaptive to the link access bandwidths as well. We design an adaptive network coding scheme in which the media content is encoded into multiple stripes. The peers select one or more stripes to subscribe based on their own downloading bandwidths. Peers with different downloading bandwidths can receive the media content at different rates simultaneously. Both the above two topics are discussed in Chapter 4.

- It explores the inter-session linear network coding for multicast networks and proposes a practical inter-session network coding scheme. Most existing works focus on intra-session linear network coding, i.e. they consider only one multicast session when constructing linear network coding scheme. Actually, multiple simultaneous multicast sessions can also benefit from linear network coding if they are encoded together. We investigate inter-session network coding by providing heuristic algorithms and conducting extensive simulations. We propose an approach to identifying the situations where it is the most profitable to do inter-session network coding and which sessions should be encoded together. The benefit is quantified by introducing two different performance metrics. We study the performance of the proposed inter-session network coding scheme from both the deterministic coding and random coding perspectives. Our simulation results show that the inter-session network coding outperforms the intra-session network coding by about 30% in terms of throughput in most cases. It is discussed in Chapter 5.

- It investigates the problems in existing network layer multicast and provides a different angle to tackle these problems. We propose a new service-centric multicast architecture which includes a systematic solution for network layer multicast from hardware to software. In particular, the proposed multicast routing protocol SCMP is more scalable and efficient compared to existing multicast routing protocols. The centralized processing makes SCMP easier to be adapted to various applications with different QoS(Quality of Service)

requirements. Moreover, we apply peer-to-peer technology to multicast transportation layer and propose a scalable reliable multicast protocol. The reliable multicast protocol functions similar to TCP in unicast communication. It provides reliable transmission and flow control between the source and the receivers. Thanks to the peer-to-peer technology, the proposed approach is free from the acknowledge implosion problem in other reliable multicast protocols and the performance is improved significantly. These are discussed in Chapter 6 and 7.

Chapter 2

A Hypergraph Approach to Linear Network Coding in Multicast Networks

In [16], Jaggi *et al.* proposed a polynomial deterministic algorithm for linear network coding construction for a multicast network. However it is only suitable for the basic linear network coding scheme construction without any constraints. In practice, there are always some limitations on the resource used for network coding. For example, in many minimal linear network coding problems, it is desirable to minimize the number of packets experiencing coding, or to minimize the number of nodes performing coding [19, 22].

To solve these minimal network coding problems, we introduce a unified approach based on hypergraph in this chapter. Hypergraph is first used to solve linear network coding problem in [23] in which a hypergraph called *conflict graph* is constructed based on the network topology such that each node represents an edge in the network topology carrying some combination of the original messages, with each edge connecting the nodes that are incompatible under any valid linear network coding assignments. Then finding a valid linear network coding assignment is equivalent to finding a maximum independent set in the conflict graph. However, this work focused on the application of the network coding within a single multicast switch. Since finding a maximum independent set of a graph is a NP complete problem [24], it is impractical to use this approach to find a linear network coding assignment for large networks. In addition, the paper gave only some general guidelines without a formal description on how to construct the hypergraph and how to find such an independent set.

In this chapter, we propose a formal unified approach that can convert the network-wide linear network coding problem in a multicast network into an equivalent graph theory problem. We will also use a hypergraph to model the linear net-

work coding assignment but in a completely different way. In our approach, only compatible nodes (to be defined later) are connected by a hyperedge in the transformed hypergraph to reduce the problem complexity. We call the hypergraph a *pseudo-dual hypergraph* because it is constructed by mapping the nodes and edges in the graph representing the multicast network to the edges and nodes in the hypergraph respectively.

The main contributions of this chapter can be summarized as follows:

- We provide a general linear network coding solution for any multicast networks which can be modeled by directed graphs. We first transform the linear network coding problem into a tractable graph theory problem, and then propose an iterative refinement algorithm to solve the graph theory problem. By iterative refinements, a valid linear network coding assignment can be found in polynomial time.

- We provide a unified solution for a series of minimal linear network coding problems. To the best of our knowledge, this is the first work to give a systematic approach to solving the various minimal linear network coding problems. By transforming the linear network coding problem into a graph theory problem, minimal linear network coding problems can be transformed to the corresponding problems in graph theory and solved by algorithms in graph theory. This is the primary advantage of this method compared with other existing linear network coding construction methods.

- We propose several preprocessing algorithms which can be used to find a subgraph of the multicast network before applying any type of network coding approaches. The time complexity of the method is sensitive to the multicast capacity and the number of receivers. The preprocessing algorithms are designed to reduce the computation time by finding a minimal subgraph whose size is much smaller than the original graph. Due to the reduced graph size, both the computation time and the network bandwidth can be saved.

2.1 Problem Formalization

Suppose a multicast network is represented by a directed acyclic graph (DAG) $G = (V, E)$, where V is the node set and E is the edge set. Without loss of generality, we assume that there is a *source* node $s \in V$ which generates messages and a set of *receivers* $R \subset V$ which consume messages in the multicast network. Although network topology may contain circle, we use the DAG to model a multicast network because we focus on the subgraph used for multicast traffic transmission. Usually, such a subgraph is a tree spanning the source and receivers without circle. Similar

to the model described in 1, an edge e is represented by an ordered node pair (x, y), where $x, y \in V$, y is called the head of the edge and denoted as $y = head(e)$, and x is called the tail of the edge and denoted as $x = tail(e)$. The messages can only be transmitted from x to y. The incoming edge set of a node v is defined as

$$E_{in}(v) = \{(x, y)|(x, y) \in E, y = v\} \tag{2.1}$$

Similarly, the outgoing edge set of a node v is defined as

$$E_{out}(v) = \{(x, y)|(x, y) \in E, x = v\} \tag{2.2}$$

Table 2.1 lists all the notations to be used in the rest of the chapter.

In the graph, each node has one or more incoming edges and one or more outgoing edges except that the source node has no incoming edges and the receivers have no outgoing edges. The minimum cut between the source node and each receiver is h. According to network coding theory, source s can multicast messages to all the receivers at rate h using network coding. We first consider the special case when $h = 2$. We will discuss the general case in Section 2.3.

More generally, a network can have multiple source nodes that generate messages, or a receiver that not only consumes messages but also forwards messages. Such a network can be converted into a network satisfying our above assumptions without altering the solvability properties of the network as follows. In the case that there are k source nodes, a virtual source node s with k outgoing edges is added. These k outgoing edges are connected to the k real source nodes respectively. In the case that a receiver has one or more outgoing edges, a virtual receiver with h incoming edges is added. All the h incoming edges are originated from the real receiver.

As mentioned in the previous chapter, in a multicast network with linear network coding, the message carried by an edge e can be considered as a linear combination of the messages from source s. Suppose W is the message vector with h elements, and Q is a valid linear network coding assignment such that $Q(e)$ represents the global encoding vector of edge e. Then the message on edge e is $W \times Q(e)^T$, where $Q(e)^T$ is the transpose of $Q(e)$. If we consider the messages as a sequence of bits, the linear network coding assignment is constructed over the finite Galois field with field size 2, i.e., $GF(2)$. [1] The addition operation over $GF(2)$ is XOR and the multiplication operation over $GF(2)$ is AND. For each edge, theoretically there are $2^h - 1$ possible global encoding vectors which represent the $2^h - 1$ different combinations of the h messages (An edge with zero global encoding vector can be

[1] The field size is determined by many factors, such as multicast capacity h and network topology. Determining the field size is beyond the scope of this thesis. Interested readers may refer to the literatures, such as [25, 26], on this topic.

Table 2.1: Notations Used in This Chapter

$G = (V, E)$	G is a DAG representing a multicast network topology, where V is the node set and E is the edge set.
$G' = (V', E')$	pseudo-dual hypergraph of G, where V' is the node set and E' is the edge set.
G''	a subgraph of G'
$G_t = (V_t, E_t)$	a graph transformed from G'', where V_t is the node set and E_t is the edge set.
s	source node
R	a set of receivers
$head(e)$	head of edge e.
$tail(e)$	tail of edge e.
$E_{in}(v)$	incoming edge set of node v.
$E_{out}(v)$	outgoing edge set of node v.
h	minimum cut
Q	a valid linear network coding assignment in a multicast network
W	message vector
L	a group in G'
$P,\ P'$	a path in G and G', respectively
S_P	message flow function defined on path P
F	a feasible source in G'
T	a cover in G'
$e(v')$	corresponding edge $e \in E$ of node $v \in V'$
$U(v')$	global encoding vector of node $v \in V'$
N	upper bound on the number of incoming edges of a node in V
N_R	upper bound on the number of incoming edges of a receiver
M	upper bound on the number of outgoing edges of a node in V

deleted from the graph without altering the network solvability properties).

Given a graph G, we define a pseudo-dual hypergraph $G' = (V', E')$ which is constructed by mapping each node in G to multiple edges in G' and each edge in G to multiple nodes in G'. We use an example to illustrate the hypergraph construction. Suppose the network topology is the butterfly network, we label the edges as shown in Fig. 2.1.

In a graph an edge is connected to exactly two nodes, while in a hypergraph an

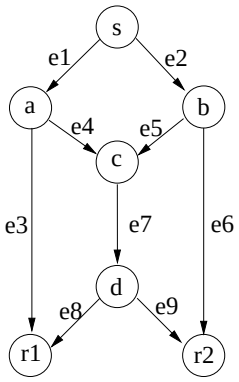


Figure 2.1: The network topology of the butterfly network.

edge can be connected to any number of nodes. In the rest of the chapter, we use v and e to represent a node and an edge in G respectively, and v' and e' to represent a node and an edge in G' respectively. Each edge e is mapped to $2^h - 1 = 3$ nodes in E'. These three nodes represent edge e when the global encoding vector of e is (0,1), (1,0) and (1,1), respectively. Thus the nodes in G' can be identified by a pair $v' = (e, U),\ e \in E$ and $U \in \{(0,1),(1,0),(1,1)\}$. We use $e(v')$ and $U(v')$ to represent the first and the second elements of node v'. If $v' = (e, U)$, then $e(v') = e$ and $U(v') = U$. For the nodes whose first elements are the same, we call them a $group$, denoted by L. For example, we have $L(e) = \{(e,(0,1)),(e,(1,0)),(e,(1,1))\}$. In G', there are $|E|$ groups and each group has three nodes, thus $|V'| = 3|E|$. To have a visualized idea of what the pseudo-dual hypergraph looks like, readers may refer to Fig. 2.2. We will discuss its construction in more detail later in the chapter.

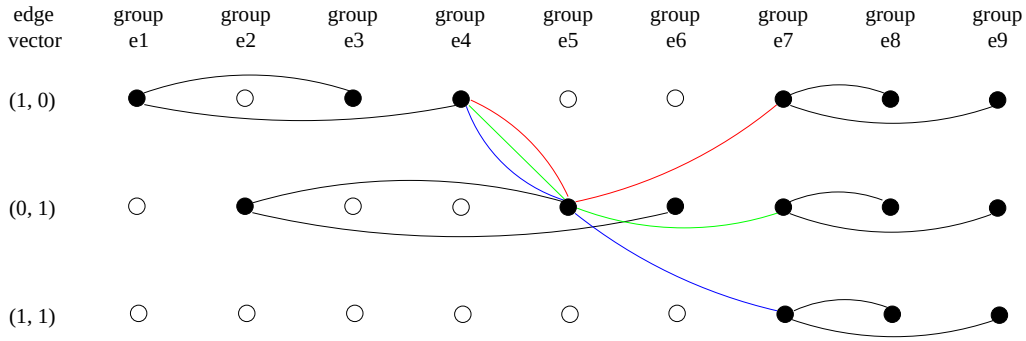


Figure 2.2: The pseudo-dual hypergraph of the multicast network.

In a linear network coding assignment, the global encoding vector of an outgoing edge must lie in the linear span of the global encoding vectors of all the

incoming edges. A linear span of a set of vectors is defined as the intersection of all subspaces containing the set. We use notation $\langle \bullet \rangle$ to denote the linear span. For example, over the finite field $GF(2)$, $\langle \{(0,1),(1,1)\} \rangle = \{(0,1),(1,0),(1,1)\}$ (we do not consider zero global encoding vector). Thus we have

$$Q(e) \in \langle \bigcup_{i=1}^{|E_{in}(v)|} Q(e_i),\ e_i \in E_{in}(v) \rangle,$$
$$\forall\ e \in E_{out}(v),\ \forall\ v \in V$$

Now we are ready to define the edges in G'. Each edge e' can be represented by an ordered pair (X, y) where $X \subset V'$ and $y \in V'$. Similarly, X is called the tail of the edge and y is called the head of the edge. Generally speaking, given a node v with incoming edge set $E_{in}(v)$ and outgoing edges set $E_{out}(v)$, the mapped edge e' is defined as follows

$$e' \in E',\ e' = (X, y) \quad \text{if and only if}$$
$$|X \cap L(e)| = 1, \forall\ e \in E_{in}(v) \text{ and}$$
$$y \in \bigcup_{i=1}^{|E_{out}(v)|} L(e_i),\ e_i \in E_{out}(v) \text{ and}$$
$$U(y) \in \langle \bigcup_{i=1}^{|X|} U(x_i) \rangle,\ x_i \in X$$

The above definition implies that if the global encoding vector of an outgoing edge lies within the linear span of the set of global encoding vectors of incoming edges, there should be an edge connecting the corresponding nodes in G'. Since the linear network coding assignment is over GF(2) with $h = 2$, any set of more than two global encoding vectors are linearly dependent. Thus it is sufficient to consider edges (X, y) where $|X| \leq 2$. Specifically, for every two incoming edges e_1 and e_2 and an outgoing edge e_3, there are $3 + 6 \times 3 = 21$ edges between them. The first term 3 represents the 3 edges where the global encoding vectors of two incoming edges are the same. In this case the linear span contains only one vector which is the same as the global encoding vectors of incoming edges. The second term 6×3 means that there are 6 different combinations of global encoding vectors of incoming edges which are linearly independent. Thus the linear span contains all the 3 possible vectors. For each such combination, there are 3 edges connecting it to the nodes corresponding to the outgoing edge with 3 different global encoding vectors respectively. Fig. 2.3 shows all the 21 edges. For simplicity, we omit the edges with the same edge tails. In the figure, the edges are differentiated by their

colors and line styles. The nodes connected by the lines of the same color and style are connected by an edge. The rightmost node is the edge head and the rest of the nodes belong to the edge tail. For example, the three bold (black) lines represent the three edges of the first term. Node a, f and g are connected by the dot-dash (red) line, which represents edge $(\{(e_1,(0,1)),(e_2,(1,1))\},(e_3,(0,1)))$. The other two edges with the same edge tail are $(\{(e_1,(0,1)),(e_2,(1,1))\},(e_3,(1,0)))$ and $(\{(e_1,(0,1)),(e_2,(1,1))\},(e_3,(1,1)))$.

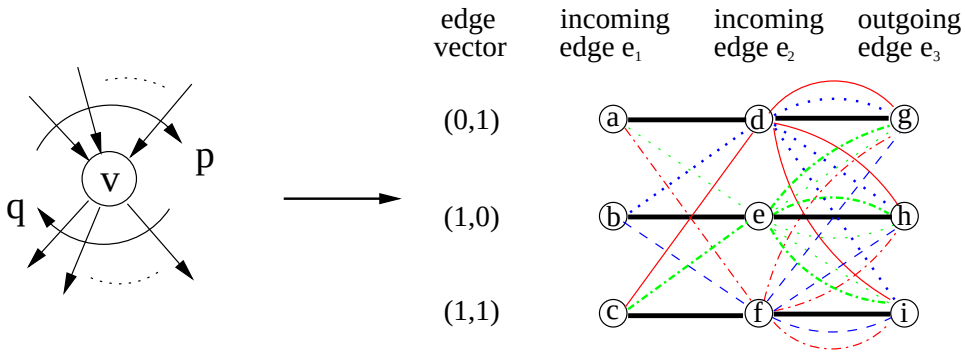


Figure 2.3: Edge construction in G'.

We have the following property concerning the edge construction.

Property 1 *For a node v with p incoming edges and q outgoing edges, the total number of mapped edges is: (i) $3q$, when $p = 1$; (ii) $\frac{21}{2}p(p-1)q$, when $p > 1$.*

Proof: Case 1: $p = 1$. In this case, since there is only one incoming edge, the only operation the node can perform is forwarding. For each outgoing edge, there are three edges connecting the nodes corresponding to the incoming edge and the outgoing edge with the same global encoding vector. Thus the total number of edges is $3q$.

Case 2: $p > 1$. As mentioned earlier, only edges $e' = (X, y)$ with $|X| \leq 2$ need to be considered. From Fig. 2.3, we can see that given a node in $L(e_1) = \{a, b, c\}$ and a node in $L(e_3) = \{d, e, f\}$, there is always an edge connecting the two nodes regardless of the global encoding vector of the second incoming edge e_2. In other words, for any edge (X, y) with $|X| = 1$, we can always find another edge (X', y) with $|X'| = 2$ and $X \subset X'$ to replace the edge without violating other constraints. Therefore we only need to count edges (X, y) with $|X| = 2$. p incoming edges have $p(p-1)/2$ different combinations of two different incoming edges. Given one of these combinations and one outgoing edge, there are 21 edges between them. Thus the total number of edges is $\frac{21}{2}p(p-1)q$. ■

A *path* P' in G' is defined as a sequence of edges

$$P' = e'_1 e'_2 \dots e'_m,\ e'_i \in E',\ i \in \{1, 2, \dots, m\} \text{ such that}$$
$$head(e'_i) \in tail(e'_{i+1}),\ \forall\, i \in \{1, 2, \dots, m-1\}$$

Given a path P' in G', we can always find a corresponding path P in G:

$$\begin{aligned} &P = v_1 v_2 \dots v_{m+1} \text{ such that} \\ &v_i = tail(e(head(e'_i))),\ \forall\, i \in \{1, 2, \dots, m\} \text{ and} \\ &v_{m+1} = head(e(head(e'_m))) \end{aligned} \tag{2.3}$$

An edge in path P corresponds to a node in path P'. An edge in path P' corresponds to a node in path P as well. A node in path P' not only determines an edge in P, but also determines the global encoding vector of the edge. Connecting multiple nodes, an edge in path P' determines the linear coding operation on the corresponding node v in path P with each connected node representing an incoming edge or outgoing edge of node v. Since only compatible nodes are connected by an edge, it is possible for node v to perform the linear network coding.

Path P' not only determines a corresponding path P in G, but also determines a message flow over path P. A *message flow* over a path $P = v_1 v_2 \dots v_{m+1}$ defines the global encoding vectors of incoming edges of nodes in $\{v_1, v_2, \dots, v_m\}$ by function $S_P(e) \in \{(0,1), (1,0), (1,1)\},\ e \in \bigcup_{i=1}^{m} E_{in}(v_i)$ such that $S_P((v_i, v_{i+1})) \in \langle \bigcup_{j=1}^{|E_{in}(v_i)|} S_P(e_j) \rangle,\ e_j \in E_{in}(v_i),\ \forall\, i \in \{1, 2, \dots, m\}$. From the definition we can see that a message flow over a path P gives an assignment of global encoding vectors along the path P which is a subset of a linear network coding assignment. On the other hand, given a linear network coding assignment and a path P, the global encoding vectors along the path form a message flow. Based on this observation, we propose an iterative refinement algorithm to find a valid linear network coding assignment by finding the message flows in G'. We will discuss the algorithm in detail in Section 2.2.

For a linear network coding assignment to be valid, the rank of the matrix composed by the global encoding vectors of the edges in $E_{out}(s)$ must be h which is 2. A *feasible source* F is defined to describe a feasible set of global encoding vectors of the edges in $E_{out}(s)$.

$$\begin{aligned} &F \subset V',\ |F| = |E_{out}(s)| \\ &\forall\, e \in E,\ |F \cap L(e)| = \begin{cases} 0 & e \notin E_{out}(s) \\ 1 & e \in E_{out}(s) \end{cases} \\ &Rank(U(v'_1), U(v'_2), \dots, U(v'_{|F|})) = 2, \\ &v'_i \in F,\ i \in \{1, 2, \dots, |F|\} \end{aligned}$$

Given a feasible source F, not every node in G' is feasible. i.e. for some edges, a certain global encoding vector is not possible in any valid linear network coding assignment. For a node in G' to be feasible, it must be on a path originated from one of the nodes in F. This path corresponds to a message flow which indicates how the messages are encoded along the path. The infeasible nodes and the related edges can be deleted from G'. Fig. 2.2 shows the pseudo-dual hypergraph G' of the multicast network in Fig. 2.1 where $F = \{(e_1, (1, 0)), (e_2, (0, 1))\}$. The feasible nodes are colored in black while the infeasible nodes are colored in white. Here we omit the edges connecting infeasible nodes. For each edge, the rightmost node is the edge head and the rest of the nodes form the edge tail.

A *cover* T in G' is defined as a set of nodes such that

$$T \subset V',\ |T \cap L(e)| = 1,\ \forall\, e \in E \tag{2.4}$$

Therefore, a cover contains $|E|$ nodes from $|E|$ different groups. A cover defines the global encoding vectors of all edges in G. Thus any linear network coding assignment can be represented by a cover, but not every cover represents a linear network coding assignment. As we mentioned earlier, given a linear network coding assignment and a path P in G, the corresponding path P' in G' must be a message flow over P. If we consider graph G as a union of multiple paths starting from the source node, in the pseudo-dual graph G', a cover represents a linear network coding assignment if and only if for every node in the cover, there is a path from a node in $\bigcup_i^{|E_{out}(s)|} L(e_i),\ e_i \in E_{out}(s)$ to it and the path contains the nodes in the cover only. A cover represents a valid linear network coding assignment if and only if the cover represents a linear network coding assignment and for each receiver, the rank of the matrix composed by the global encoding vectors of incoming edges is h.

Given the multicast network G, the source node s, the receiver set R and the pseudo-dual graph G', we formalize the conditions for a cover T to be a valid linear network coding assignment into following three constraints:

1. $\exists F \subset T$ such that F is a feasible source;
2. $\forall\ v' \in T \setminus F,\ \exists P' = (e'_1 e'_2 \dots e'_k), P'$ is a path in $G',$ such that $tail(e'_1) \subset F$ and $head(e'_k) = v'$ and $tail(e'_i) \subset T,\ \forall\, i \in \{2, 3, \dots, k\}$;
3. $\forall\, v \in R,\ Rank(U(v'_1), U(v'_2), \dots, U(v'_{|E_{in}(v)|})) = 2,$ where $v'_i \in T,\ head(e(v'_i)) = v,\ \forall\, i \in \{1, 2, \dots, |E_{in}(v)|\}$.

We call the three constraints *valid linear code constraints.*

Based on the above discussions, we have the following theorem.

Theorem 3 *Finding a valid linear network coding assignment in a multicast network is equivalent to finding a cover T in its pseudo-dual graph G' which satisfies the valid linear code constraints.*

If we add an artificial node v_{origin} in G' such that there is an edge from v_{origin} to all the nodes in $\bigcup_{i=1}^{|E_{out}(s)|} L(e_i),\ e_i \in E_{out}(s)$, the first two constraints can be combined and rewritten as follows.

$$\begin{aligned}&\forall v' \in T,\ \exists P' = (e'_1 e'_2 \dots e'_k),\ P' \text{ is a path in } G',\ \text{such that}\\ &tail(e'_1) = v_{origin},\ head(e'_k) = v' \ \text{ and}\\ &tail(e'_i) \subset T,\ \forall i \in \{2, 3, \dots, k\}.\end{aligned}$$

2.2 An Iterative Refinement Algorithm for Finding an Eligible Cover

In the previous section, we transform the linear network coding problem into a graph theory problem. The problem of finding a valid linear network coding assignment is formalized into a problem of finding a cover in the hypergraph. In this section, we propose an iterative refinement algorithm that can find an eligible cover satisfying the valid linear code constraints in polynomial time.

A naive way to find an eligible cover is to use a brute force method, i.e. to try every possible cover until an eligible cover is found. Apparently, this method is impractical and time consuming because there are a total of $(2^h - 1)^{|E|}$ different covers.

Since an eligible cover must be able to construct paths from node v_{origin} to the nodes corresponding to incoming edges of receivers, and for each receiver the nodes corresponding to incoming edges must satisfy the second valid linear code constraint, we can find an eligible cover by finding these paths one by one. Based on this idea, we present the following iterative refinement algorithm to find an eligible cover.

Step 1: For each receiver, select a combination of global encoding vectors of the receiver which are linearly independent, and group the corresponding nodes in G' into a set Z. For each node v' in Z, delete the nodes in the same group as v'. Let the resulting graph be G'';

Step 2: If Z is NULL, output T as an eligible cover. If not, pick a node v' from Z, and delete v' from Z, then find a path from v_{origin} to v' over G''. If no such a path exists, select the next combination, and go to Step 1; if such a path exists, go to Step 3;

Step 3 : For each newly added node in the path found in Step 2, put it in set T, delete other nodes in the same group and the corresponding edges and nodes in G'', and go to Step 2.

In the first step, if the receiver has p incoming edges, there are $3p(p-1)$ different combinations of global encoding vectors which are linearly independent. In fact, if there are some constraints on feasible source F, we can delete some impossible combinations by checking the feasibility of the nodes in the combination. A combination is possible only if all the nodes are feasible.

As for the second step, it is a generalization of the problem of finding a directed path in a directed graph. A common way is to use a depth-first search or width-first search starting from the start node of the path until the end node of the path is found. This method can be used to find a directed path in our hypergraph. We first transform hypergraph G'' to a graph $G_t = (V_t, E_t)$ according to the following rules:

1. $V_t = V''$;
2. $e_t \in E_t,\ e_t = (x_t, y_t)$ if and only if $e'' \in E'',\ e'' = (X, y)$ and $x_t \in X$

It is easy to see that G_t is a directed graph with each edge connecting two nodes. After finding a path in G_t using a depth-first search or width-first search, we map this path back to a path in G''.

After a new path is determined in the second step, G'' should be updated before searching the next path. Two types of nodes and the corresponding edges will be deleted. The first type is the nodes in the same group as the newly added node and the second type is the nodes which are not feasible after the first type of nodes are deleted. The first type of nodes can be easily identified and deleted. The second type of nodes can be tracked along the path originated from the first type of nodes. Table 2.2 lists the pseudo-code of the algorithm used in Step 3.

Fig. 2.4 gives an example of eligible covers of the pseudo-dual hypergraph for the network in Fig. 2.1. Fig. 2.4(a) shows an eligible cover when the incoming global encoding vectors of r_1 are (1,0) and (1,1), and the incoming global encoding vectors of r_2 are (0,1) and (1,1). Fig. 2.4(b) shows an eligible cover when the incoming global encoding vectors of r_1 are (1,0) and (0,1), and the incoming global encoding vectors of r_2 are (1,1) and (0,1).

We have the following theorem concerning the complexity of the refinement algorithm.

Theorem 4 *For a given number of receivers, the iterative refinement algorithm can find an eligible cover in polynomial time.*

Proof: Let $n_1, n_2, \ldots, n_{|R|}$ be the numbers of incoming edges of receivers and the upper bound on these numbers be N_R. In the worst case, the algorithm will

Table 2.2: Redundant Nodes Deletion Algorithm

INPUT: newly added node v'
OUTPUT: updated G''
BEGIN
1 Delete the nodes in the same group as v';
2 $V_d = v'$;
3 **while** $V_d \neq NULL$
4 pick a node v_d in V_d;
5 **if** $e(v_d)$ is the last incoming edge of $head(e(v_d))$
6 such that there is only one node of $L(e(v_d))$ left in G''
7 **foreach** edge e in $E_{out}(head(e(v_d)))$
8 **if** e has only one possible global encoding vector U_e
9 add node (e, U_e) into V_d;
10 delete other nodes in group $L(e)$;
END

try all possible combinations in Step 1, which has $\prod_{i=1}^{|R|} 3n_i(n_i - 1)$ different combinations. For each combination, Step 2 has to find paths from v_{origin} to $\sum_{i=1}^{|R|} n_i$ different nodes iteratively. Based on these paths, Step 3 deletes redundant nodes and edges. As the nodes and edges can only be deleted once for all the paths, the time for Step 3 is $O(|V'| + |E'|)$ for each combination. The time to find a path in G' is $O(|V'| + |E'|)$. Thus the total time for finding an eligible cover is

$$\prod_{i=1}^{|R|} 3n_i(n_i - 1)(\sum_{i=1}^{|R|} n_i \times O(|V'| + |E'|) + O(|V'| + |E'|))$$
$$= O(N^{2|R|+1}|R|(|E| + |V|N^2M)).$$

Considering that N and M are constants and $|R|$ is given, the time for finding an eligible cover is polynomial with respect to $|V|$ and $|E|$. ■

As we can see, the time complexity of the algorithm depends on the size of hypergraph G'. To further reduce the size of G', we can optimize graph G before transformation, which is called *merging*. Merging can be used when there exists a node v in G with one incoming edge and multiple outgoing edges. Clearly, such a node can only forward messages from the incoming edge to outgoing edges. In the pseudo-dual graph G', this case can be represented by a hyperedge connecting the one node representing the incoming edge and the other two nodes representing the two outgoing edges and all three nodes are mapped to the same global encoding vector. As their global encoding vectors are the same, we can further reduce the

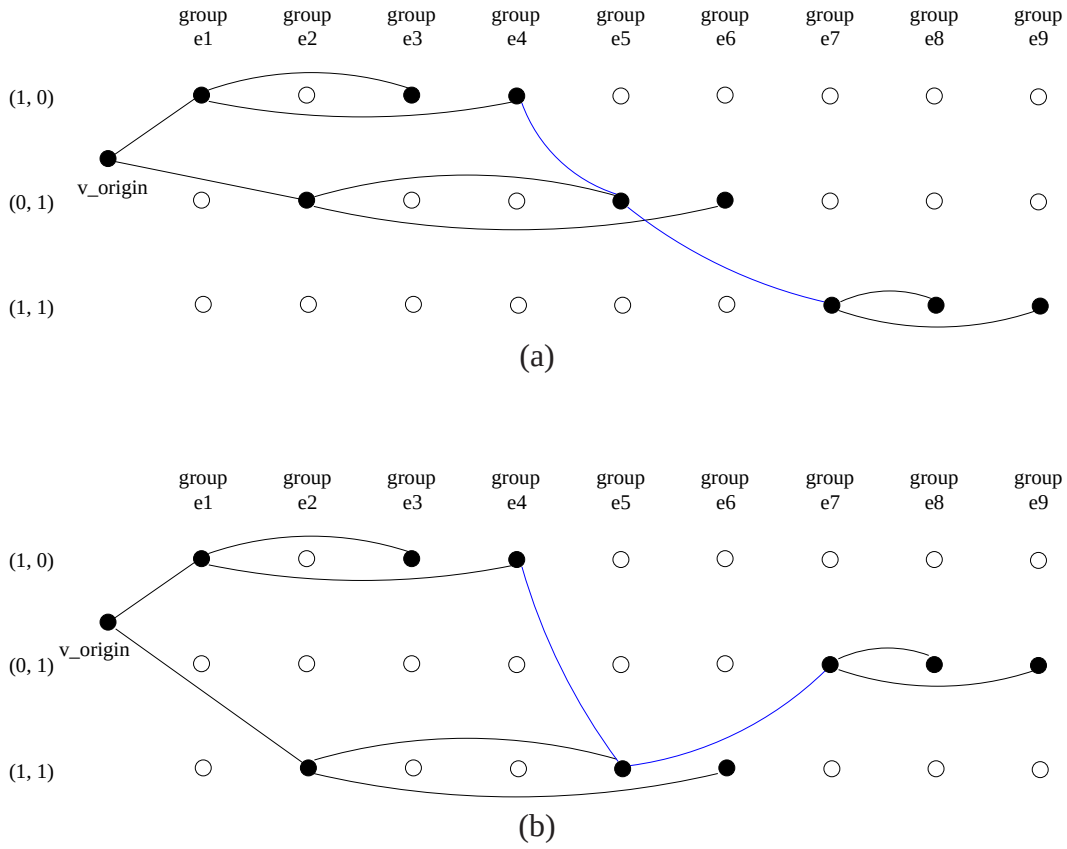


Figure 2.4: An example for eligible covers of the pseudo-dual hypergraph. (a) An eligible cover when $F = \{(e_1, (1, 0)), (e_2, (0, 1))\}$. (b) An eligible cover when $F = \{(e_1, (1, 0)), (e_2, (1, 1))\}$.

graph size by merging the three nodes without altering the result of the algorithm. Specifically, we use one node to represent the three nodes and delete the hyperedges between them. In practice, we first find a node with only one incoming edge in the topology graph. Then we execute the merging on the corresponding nodes in the hypergraph. The process is performed repeatedly until there is no such a node to be found. Fig. 2.5 shows one of the eligible covers after merging.

2.3 Extensions to General Minimal Network Coding Problems and Generalizations

It should be pointed out that the main purpose of the proposed hypergraph method is not to find a linear network coding assignment for a multicast network, instead,

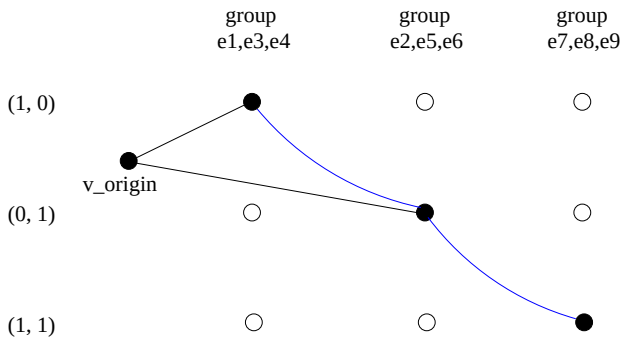


Figure 2.5: An eligible cover of the pseudo-dual hypergraph after merging.

is to provide a unified solution for a series of minimal network coding problems. This is the primary advantage that makes the proposed approach superior to other existing linear network coding construction algorithms. In fact, the method can be applied to many minimal network coding problems discussed in [19]. Next, we use two examples to illustrate how to extend the method to solve these problems.

In some applications, we may want to minimize the number of packets experiencing coding. This is particularly the case in optical networks where a conversion from optical signals to electrical signals is needed in order to encode the packets. To achieve this, we can add more weight to the edges in G' that experience message encoding. For example, edge $((1,0),(1,0),(1,0))$ means that the corresponding node forwards one of the messages from two incoming edges to the outgoing edge, edge $((1,1),(1,0),(1,0))$ means that the corresponding node forwards the message from the second incoming edge to the outgoing edge, and edge $((1,1),(1,0),(0,1))$ means that the corresponding node encodes the messages from two incoming edges before sending it to the outgoing edge. Note that in Step 2 of the algorithm for finding an eligible cover, we always find the minimum weight path for each node in Z.

In other applications, we may want to minimize the number of nodes performing coding, i.e. to maximize the number of nodes that only forward messages. To achieve this, we can construct the edges by considering all the incoming edges and outgoing edges. For a node v with p incoming edges and q outgoing edges, there is

an edge $(X, Y), X \subset V', Y \subset V'$ if

$$
\begin{aligned}
&|X| = p,\ |Y| = q,\\
&\forall\, e \in E,\ |X \cap L(e)| = \begin{cases} 0 & e \notin E_{in}(v) \\ 1 & e \in E_{in}(v) \end{cases}\\
&\forall\, e \in E,\ |Y \cap L(e)| = \begin{cases} 0 & e \notin E_{out}(v) \\ 1 & e \in E_{out}(v) \end{cases}\\
&\forall v'_y \in Y,\ U(v'_y) \in \langle \bigcup_{i=1}^{p} U(v'_i) \rangle, v'_i \in X
\end{aligned}
$$

We can assign less weight on the edges where $\bigcup U(v'_y) \subset \bigcup U(v'_x), v'_y \in Y$ and $v'_x \in X$ and assign more weight on other edges. Similarly, the number of nodes performing coding can be minimized by finding the minimum weight path in G''.

Next, we consider some generalizations of the method. We have focused on the case for $h = 2$ so far. We now generalize the method to the case for $h > 2$.

For $h > 2$, the hypergraph construction is similar. Each edge e is mapped to $2^h - 1$ nodes in G'. There is an edge between the nodes if the global encoding vector of the corresponding outgoing edge lies in the linear span of the set of global encoding vectors of corresponding incoming vectors. If a node v has more than h incoming edges, it is sufficient to consider the combination of less than or equal to h incoming edges only. However, when $h > 2$, it is likely that there does not exist any valid linear network coding assignment for some network topologies. In this case, no cover satisfies the valid code constraints unless it is permitted to use the linear network coding assignments over a larger finite field. Suppose we use linear network coding assignments over a finite field $GF(n)$, the number of nodes in G' mapped by one edge in G is $n^h - 1$. We can see that the size of the pseudo-dual graph grows exponentially when h increases. Thus, it is critical to reduce the size of the feasible solution set when h or field size is large. As the algorithm time complexity is proportional to the number of nodes and edges in the hypergraph, we propose to remove the redundant nodes or edges in the hypergraph to reduce the problem size. Clearly, such removal should not alter the solvability of the problem and the resulting hypergraph size should be small enough to apply the method in practice. In the next section, we propose several efficient algorithms to remove the redundant nodes and edges. These algorithms are called preprocessing algorithms because they should be applied to the hypergraph before we apply the iterative refinement algorithm to the hypergraph.

2.4 Preprocessing Algorithms

Before we introduce the preprocessing algorithms, we need to answer the following question: Is it possible to reduce the graph size without altering the solvability of the linear network coding problem? The answer to this question is yes based on the following observation: for a multicast network with a certain multicast capacity, there is at least one valid linear network coding assignment which can achieve the multicast capacity. This can be directly inferred by the result in [15] that the multicast capacity can be achieved by linear network coding. In other words, given a multicast network and one of its subgraphs (the subgraph should include at least the source and the receivers such that it is also a multicast network), there is always a valid linear network coding assignment as long as the multicast capacity of the subgraph is the same as the multicast network.

In this section, we discuss how to reduce the graph size in order to reduce the consumed time of the iterative refinement algorithm. The basic idea is to find a subgraph of the network topology graph such that both the source node and the receivers are in the subgraph and the multicast capacity of the subgraph is the same as the original graph. Then we apply the hypergraph transformation to the subgraph to find a valid linear network coding assignment. The linear network coding assignment for the subgraph can be easily adapted and applied to the original graph by assigning zero global encoding vectors to the edges which are in the original graph but not in the subgraph.

As the procedure involves the processing of the graph before the transformation, we call it preprocessing and the corresponding algorithms are called preprocessing algorithms. Besides reducing the computation time of the iterative refinement algorithm, preprocessing can save the network bandwidth consumption as well. Recall that the linear network coding assignment constructed based on the subgraph needs to be adapted before being applied to the original multicast network. The global encoding vectors of the edges which are not in the subgraph are set to be zero. In other words, the traffic is confined within the subgraph after preprocessing while the multicast capacity is still achieved by the linear network coding assignment. As a result, the network bandwidth is saved without sacrificing the performance. Another merit of the preprocessing is that it is a general optimization for any linear network coding assignment, not only for the scheme specially designed for the iterative refinement algorithm. In fact, any linear network coding construction algorithm can apply the proposed preprocessing algorithms first before constructing a linear coding assignment such that the graph size is reduced and the network bandwidth is saved. We will evaluate the performance of the preprocessing algorithms in terms of computation time and network bandwidth in Section 2.5.

It should be mentioned that the merging method discussed in the previous section is a type of preprocessing to reduce the size of the hypergraph. In this section,

we consider the preprocessing in depth by giving several different preprocessing algorithms with respect to different optimization objectives.

2.4.1 Greedy Preprocessing Algorithm

Theoretically, we can pick any subgraph with the same multicast capacity as the original graph. Here we want to minimize the size of the subgraph. Thus our goal is to find a minimal subgraph with the same multicast capacity. By a minimal subgraph we mean that if any node or edge is removed from the subgraph, the multicast capacity becomes smaller.

Recall that multicast capacity is the minimum of the maximum flows from the source to each receiver. For a multicast network with multicast capacity h, it is guaranteed that there are h edge-disjoint paths between the source and each receiver. A straightforward way to find the minimal subgraph is to find the h edge-disjoint paths for each receiver and add the paths to the subgraph one by one. However, the subgraph generated this way is not good enough as it does not consider minimizing the numbers of nodes and edges when searching for the paths. Therefore, we propose the first preprocessing algorithm which is called greedy minimal subgraph algorithm as shown in Table 2.3, where R is the set of receivers and T is the union of the paths that have been found. In the greedy minimal subgraph algorithm, we always find the shortest paths between the source and the receiver so that the numbers of nodes and edges are minimized in each iteration.

Table 2.3: Greedy Minimal Subgraph Algorithm

```
INPUT: Graph G
OUTPUT: Subgraph G_s
BEGIN
1  G_s = NULL;
2  foreach receiver r ∈ R
3    T = {};
4    do
5      Find a shortest path p between s and r in G − T;
6      T = T + p ;
7    until h paths are found;
8    G_s = G_s + T;
END
```

After applying the greedy minimal subgraph algorithm, the network topology is "compressed." Some nodes and edges are removed, which results in a reduced

hypergraph size. The "compression ratio" can be approximated as follows. Assume that the number of nodes is N, and there exists a link between any two nodes with probability p. Therefore, the average node degree is $p*(N-1)$, the number of edges is approximately $p*N*(N-1)/2$ and the diameter is $d = \log N/\log(p*(N-1))$. As end hosts are usually at the edge of the network, we can use the diameter to approximate the average path length from the source to the receivers. If h is the multicast capacity, for any receiver the number of nodes on the h edge-disjoint paths is $(d-1)h$ and the number of edges on the h paths is dh. That is, after the greedy preprocessing algorithm, the number of nodes in the "compressed" graph is at most $(d-1)hm$ and the number of edges is at most dhm. The size can be further reduced if there is overlap between the paths for different receivers. For example, if $N = 500$, $p = 0.01$, $h = 2$, and $m = 20$, the number of edges is $1,247$. After the "compression," the number of nodes is about 114, which is only 23% of the original graph, and the number of edges is about 154, which is only 12% of the original graph. We can see that the preprocessing algorithm can reduce the graph size dramatically. In Section 2.5, we will evaluate the benefit brought by the preprocessing algorithms in more detail through simulations.

After finding the subgraph, we can further reduce the size of the graph by deleting a special type of nodes which are redundant to any valid linear network coding assignment. When a node has one incoming edge and one outgoing edge, we say it is redundant because the edge function of the outgoing edge can only be forwarding. As a result, the node and the outgoing edge can be deleted and the incoming edge is connected to the next node of the deleted node. After the deletion, the multicast capacity is unchanged. The linear coding assignment based on the reduced graph can be adapted to the original graph by making the global encoding vector of the deleted outgoing edge equal to the global encoding vector of the incoming edge.

2.4.2 Weighted Preprocessing Algorithms

In some cases, we want to minimize the number of nodes or edges with some properties when we construct the subgraph. For example, it is preferred to use the nodes or edges which are already put in the subgraph to construct the path for a different receiver to maximize the overlap of different paths. When h is large, it is preferred to find a path that will not increase the maximum in-degree of the nodes in the subgraph, as the computation time is sensitive to maximum in-degree when h is large. We assign different weights to the nodes in such a way that the nodes with the desired properties have higher weights and other nodes have lower weights. Now finding a path in the graph becomes finding a minimum weight path in the graph.

Table 2.4 shows the pseudo-code of the weighted minimal subgraph algorithm

which gives higher priority (lower weights) to the nodes or edges already in the subgraph when finding the minimum weight path. Here the weight of the path is the sum of the weights of the edges on the path. In each iteration, the number of new nodes or edges put into the subgraph is minimized. As a result, the total number of nodes and edges in the subgraph is also minimized.

Table 2.4: Weighted Minimal Subgraph Algorithm 1

```
INPUT: Graph G
OUTPUT: Subgraph G_s
BEGIN
1  Set all the edges weights to 1;
2  foreach receiver r ∈ R
3    T = {} ;
4    do
5      Find a minimum weight path p between s and r in G − T;
6      T = T + p ;
7    until h paths are found;
8    Set the weights of the edges in G corresponding to the edges in T to 0;
9    G_s = G_s + T ;
END
```

Although the above algorithm minimizes the graph size by re-using the nodes or edges as much as possible, one side effect is that the maximum degree of the subgraph may increase due to the repeated re-use of the existing nodes. From the time complexity analysis of the iterative refinement algorithm, we know that the time has an order of the maximum in-degree of the graph to the power h. When h is large, the time consumed by the algorithm is more sensitive to the maximum in-degree than to the graph size. Thus it is critical to limit the maximum in-degree when h is large. To achieve this, we propose the second weighted minimal subgraph algorithm as shown in Table 2.5. In the algorithm, we assume W is the set of nodes with the maximum in-degree in G_s. The algorithm marks the nodes that have the most number of incoming edges after previous iteration and assign these nodes with higher weights (lower priority). Then in the next iteration, the algorithm finds a minimum weight path to minimize the maximum in-degree. Here the weight of the path is the sum of the weights of the nodes on the path.

Unlike the greedy preprocessing algorithm, it is difficult to approximate the "compression ratio" of the weighted preprocessing algorithms as it depends on the positions and the join orders of the receivers. However, the first weighted preprocessing algorithm will always achieve a "compression ratio" no less than that of

Table 2.5: Weighted Minimal Subgraph Algorithm 2

INPUT: Graph G
OUTPUT: Subgraph G_s
BEGIN
1 Set all the nodes weights to 1;
2 **foreach** receiver $r \in R$
3 $T = \{\}$;
4 **do**
5 Find a minimum weight path p between s and r in $G - T$;
6 $T = T + p$;
7 **until** h paths are found;
8 Set the weights of nodes in W in G to 0 and the weight of other nodes to 1;
9 $G_s = G_s + T$;
END

the greedy preprocessing algorithm, because it maximizes the reusability of the existing nodes. As for the second weighted preprocessing algorithm, it may have a lower "compression ratio" than the greedy preprocessing algorithm, because it may adopt longer paths for a lower maximum degree. We will further investigate their performance through simulations in Section 2.5.

Among the three preprocessing algorithms proposed in this section, each algorithm has its own specific applications. The first weighted minimal subgraph algorithm minimizes the number of nodes and edges of the subgraph without considering the maximum degree of the subgraph. Thus it is suitable for the network topologies where the degree is small. The second weighted minimal subgraph algorithm considers minimizing the maximum in-degree of the subgraph, which makes it suitable for the network topologies where the degree is large and h is large. When the degree is large but h is small, it is preferred to use the greedy minimal subgraph algorithm as the time is less sensitive to the maximum in-degree and the greedy algorithm achieves the best balance between minimizing the graph size and minimizing the maximum in-degree of the subgraph.

2.5 Performance Evaluations

We have conducted extensive simulations to evaluate the effectiveness and efficiency of the proposed unified approach and preprocessing algorithms. We present the simulation results in this section.

2.5.1 Simulation Setups

The simulations are composed of two parts: first, we evaluate the performance of the preprocessing algorithms by observing how much the graph size can be reduced and how much time can be saved; second, we evaluate the multicast performance with network coding compared to that without network coding.

We first describe the simulation settings for the performance evaluation of the preprocessing algorithms. We implement the three preprocessing algorithms in C language and compile with GCC 3.4.3 on RedHat Enterprise Linux Version 4. The simulations are run on a Linux server with two 64-bit Intel Xeon processors at 3.8GHz and 8GB DDR-2 400 SDRAM. The network topologies are random topologies generated by GT-ITM [27] which is a degree-based Internet structural topology generator. The number of the nodes in the topology is a tunable parameter. Each pair of nodes are connected by a link with some probability that is also tunable. The randomness of the topology is the result of the link probability. As GT-ITM can only generate undirected graphs, we extend the generated undirected graphs to directed graphs by replacing each link with two directed links. The source and the receivers are chosen from the nodes randomly. The multicast capacity is determined after we choose the source node and the receivers. We change the tunable parameters to obtain network topologies with different multicast capacities.

We observe two performance metrics when evaluating the performance of the preprocessing algorithms:

- *Graph size*: Graph size is defined as the sum of the number of nodes and edges in the graph. The reason to use the sum instead of use them separately is that both the number of nodes and edges play similar roles in the time consumption based on the time complexity analysis of the iterative refinement algorithm: given the other parameters, the time complexity is linear in terms of either the number of nodes or that of edges. As the purpose of the preprocessing algorithms is to reduce the graph size, it is necessary to take this metric into consideration.

- *Computation time*: Computation time is defined as the time needed for the executable program compiled from the source code of the algorithm to finish on the dedicated Linux server. Graph size alone is not sufficient to evaluate the performance of the preprocessing algorithms as the time complexity of the algorithms also depends on other factors such as the maximum in-degree of the nodes. We use computation time to reveal how much time the algorithm needs and how much time it can save compared to other algorithms. Since the time is the virtual simulation time in NS2, the time unit is virtual as well.

Besides evaluating the performance of the preprocessing algorithms, we also evaluate the gain of network coding when applied to a multicast network compared

to that using multicast trees. The multicast tree approach is a common way to do multicast without network coding. A multicast tree is a Steiner tree spanning the source node and the receivers [28]. Although traditional multicast routing algorithms usually construct a single multicast tree, here we consider constructing multiple edge-disjoint multicast trees to maximize the utilization of network bandwidth of this approach. The approach of multiple multicast trees is widely used in overlay multicast [29, 30]. Given a multicast network, the problem of finding the maximum number of edge-disjoint multicast trees is called the Steiner tree packing problem and it had been shown to be NP-hard [31]. There are some works on the approximation algorithms and the bound of approximation ratio, see, for example, [32]. As the graphs discussed in this chapter have uniform link weights, we can adopt a simple heuristic rule to find the maximum number of multicast trees: The multicast trees are constructed in sequence: after a multicast tree is found, it is deleted from the graph, then the next multicast tree is found on the remaining graph, until no multicast tree exists. Although the simple heuristic does not guarantee an approximation ratio, it simplifies the implementation and provides a practical way to demonstrate the performance gap between these two schemes. As our goal is to evaluate the performance gap instead of obtaining an accurate theoretical bound, the simple heuristic is sufficient to satisfy our need.

We use NS-2 [33] as the network simulator. The network topologies we use are the same random topologies as that generated in the previous simulations. The source node sends data messages to all the receivers at a constant rate. To eliminate the randomness caused by the random topologies, for each simulation, we generate 10 random topologies. We run the simulation on the 10 random topologies and take the average of the 10 simulation results as the final result.

The simulations adopt following two performance metrics:

- *Throughput*: Throughput is defined as the service the system provides in one time unit. We let the source node send messages at a constant rate. Each message contains a sequence number which indicates its position in the message stream. After a certain period of time, we stop the message stream. The throughput is represented by the largest sequence number of the message that is received by all the receivers.

- *Bandwidth consumption*: We defined the network bandwidth used to deliver the messages as bandwidth consumption. As there are no control messages involved in the delivery, the bandwidth consumption is only caused by data messages. A data message going through one link contributes 1 unit to the bandwidth consumption.

2.5.2 Performance Evaluation of Preprocessing Algorithms

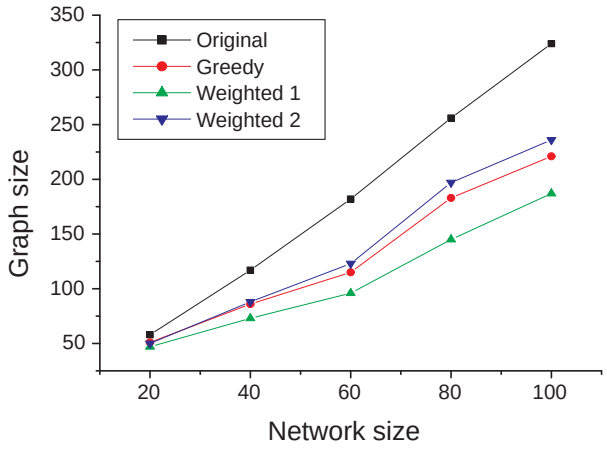
We first evaluate the performance of the preprocessing algorithms.

Fig. 2.6(a) shows the simulation results of the graph size under different network sizes. The sizes of the random network topologies range from 20 to 100. The multicast capacity is fixed at 3. The number of the receivers is fixed at 8. We can see that the original algorithm has the largest number of nodes and edges. All of the three preprocessing algorithms can effectively reduce the number of nodes. The decrease is more evident when the network size is large. Among the three preprocessing algorithms, the first weighted algorithm achieves the smallest number of nodes and edges. The performance of the greedy algorithm and the second weighted algorithm is similar. When the network size is 100, the first weighted algorithm can reduce the number of nodes and edges by about 40% compared to the original algorithm.

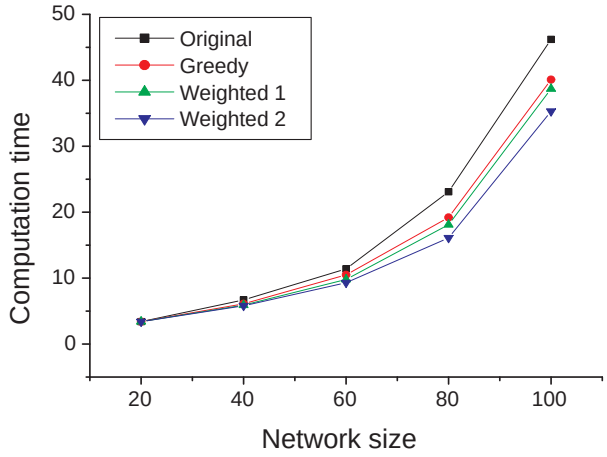
Fig. 2.6(b) shows the computation time evaluation under different network sizes. The original algorithm needs more time to complete than other three preprocessing algorithms. An interesting observation is that the second weighted algorithm needs the shortest computation time. This is another evidence that the proposed hypergraph approach is sensitive to the maximum in-degree of the graph. Compared to the original algorithm, the second weighted algorithm can reduce the computation time by 24% when network size is 100.

Fig. 2.7(a) shows the evaluation of the graph size under different multicast capacity (h) values. To compare the algorithms by the same criteria we use the same network topology whose multicast capacity is 4. The results for $h = 2$ or 3 are achieved by finding only 2 or 3 edge-disjoint paths in the corresponding algorithms. The network size is fixed at 200. The number of receivers is fixed at 8. As the original algorithm does not process the graph before the transformation, the numbers of nodes and edges remain the same under different h values. Among the three preprocessing algorithms, the first weighted algorithm achieves the smallest number of nodes and edges, while the second algorithm has the largest number of nodes and edges. With the increase of h values, the difference between the original algorithm and the preprocessing algorithms becomes smaller. This is because that the size of subgraph increases as h increases.

Fig. 2.7(b) shows the computation time evaluation under different multicast capacity (h) values. Similarly, the original algorithm needs more time to complete than the three preprocessing algorithms. However, the second weighted algorithm needs the longest computation time among the three preprocessing algorithms when $h = 2$. When h increases, the second weighted algorithm achieves less computation time than the greedy algorithm and the first weighted algorithm. It indicates that it is preferred to use the second weighted algorithm when h is large and use the greedy algorithm or the first weighted algorithm when h is small.



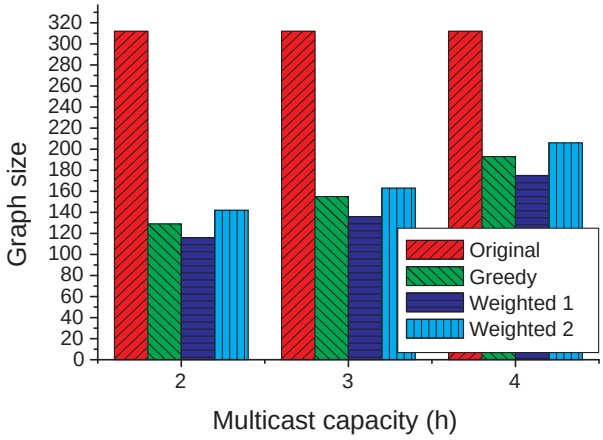
(a)



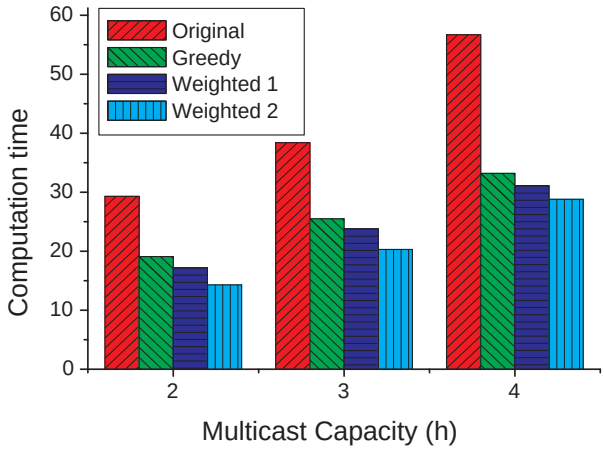
(b)

Figure 2.6: The performance comparison of the preprocessing algorithms under different network sizes. (a) Graph size evaluation; (b) Computation time evaluation.

Fig. 2.8(a) shows the evaluation of the graph size under different group sizes, i.e. the number of receivers. The graph size of the original algorithm remains constant as the graph is always the whole network topology regardless of the group size. When the group size is small, all the three preprocessing algorithms can effectively "compress" the graph. For example, the graph size is only about 30% of the original size after applying the preprocessing algorithms when the group size is



(a)



(b)

Figure 2.7: The performance comparison of the preprocessing algorithms under different h values. (a) Graph size evaluation; (b) Computation time evaluation.

5. With the increase of the group size, the "compression ratio" becomes smaller. However, the graph size after applying the preprocessing algorithms is still much smaller than the original graph size.

Fig. 2.8(b) shows the computation time evaluation under different group sizes. The computation time increases with the increase of the group size for all the algorithms. Regardless of the group size, the original algorithm needs the most compu-

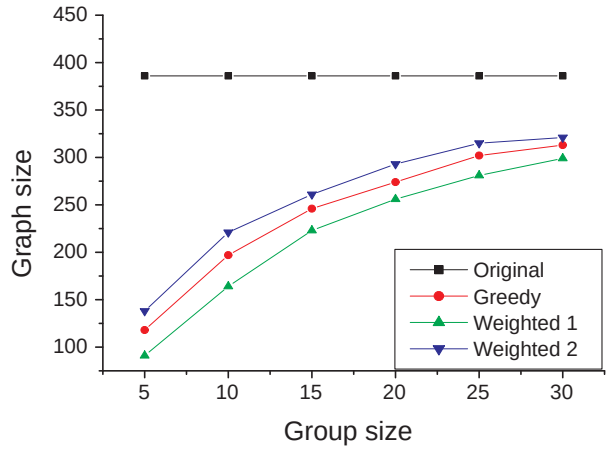
tation time, and the second weighted algorithm needs the least computation time. The derivative of the curve of the original algorithm is clearly greater than that of the three preprocessing algorithms, which means that the increase of the computation time of the original algorithm is faster than that of the three preprocessing algorithms. Therefore, the larger the group size, the more computation time is saved by the preprocessing algorithms. When the group size is 30, the computation time of the second weighted preprocessing algorithm is only half of the original algorithm. Note that the percentage of saved computation time in this figure is higher than that in Fig. 2.6. This is because that we adopted a larger network size to accommodate larger group sizes in this simulation. The figures also indicate that the preprocessing algorithms can effectively reduce the graph size especially when the ratio of the group size to the network size is small.

2.5.3 Performance Evaluation of Network Coding on Multicast

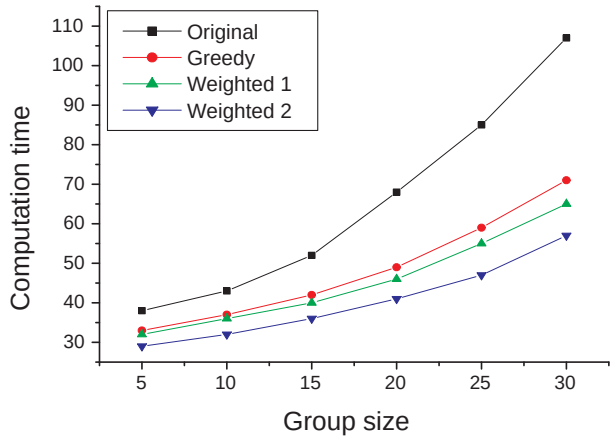
In this subsection, we present our simulation results on the performance of network coding compared to that of multiple multicast trees. We denote the multiple multicast tree approach by MT and the network coding approach by NC.

Fig. 2.9 shows the throughput evaluation under different group sizes when h is 3 and 4, respectively. Generally speaking, network coding achieves higher throughput than MT. When the group size increases, the throughput of MT drops quickly because it is more difficult to find multiple edge-disjoint trees to span all the receivers. On the other hand, the throughput of network coding becomes only slightly lower with the increase of the group size. This is due to the increase of the average delay from the source to the receivers as the group size increases. On average, the throughput is increased by 25% when applying network coding. When h increases, both approaches achieve higher throughput.

Fig. 2.10 shows the bandwidth consumption evaluation under different group sizes. We can see that the original algorithm always consumes the most network bandwidth because all the nodes and edges are used in the network coding assignment construction. The greedy algorithm and the second weighted algorithm consume similar network bandwidth which is about 80% of that consumed by the original algorithm. The first weighted algorithm consumes the least network bandwidth when the group size is small, while the MT consumes the least network bandwidth when the group size is large. This is because that when the group size is small, the first weighted algorithm finds a small subgraph to construct the linear network coding assignment on. Thus the messages are only delivered on the subgraph. When the group size is large, both the number of the delivered messages and the number of the multicast trees found by MT become smaller. As a result, the consumed network bandwidth drops as well. For the three preprocessing algorithms, when



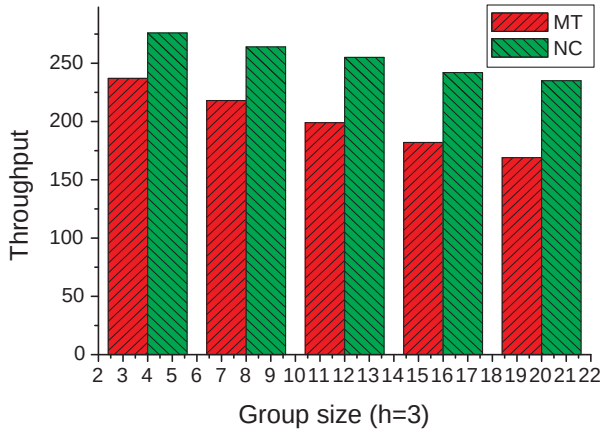
(a)



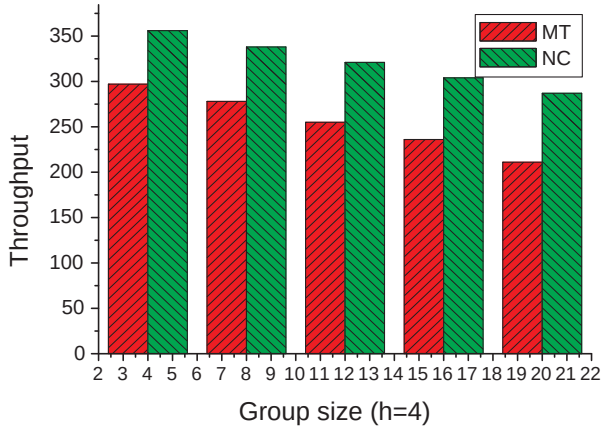
(b)

Figure 2.8: The performance comparison of the preprocessing algorithms under different group sizes. (a) Graph size evaluation; (b) Computation time evaluation.

the group size increases, the network bandwidth consumption increases as well because more nodes and edges are involved in the coding. For the original algorithm, the network bandwidth consumption drops slightly due to the minor decrease of the throughput (the number of delivered messages).



(a)



(b)

Figure 2.9: The throughput comparison between network coding and multiple multicast trees under different group sizes. (a) $h = 3$; (b) $h = 4$.

2.6 Summary

Network coding is a promising technique to improve the resource utilization in multicast networks. In this chapter, we have proposed a formal approach which can transform a linear network coding problem in multicast networks into a graph theory problem. Under such transformation, a valid linear network coding assignment for a multicast network is mapped to a cover in a hypergraph satisfying some valid

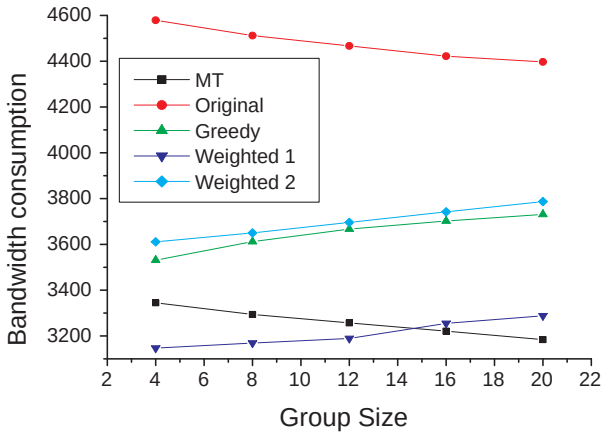


Figure 2.10: The bandwidth consumption comparison between network coding and multiple multicast trees under different group sizes.

code constraints. Given the minimum cut of the multicast network and the number of receivers, an eligible cover can be found in polynomial time with the proposed iterative refinement algorithm. In addition, the proposed preprocessing algorithms can effectively reduce the network bandwidth consumption and the computation time, which makes it practical to apply this approach to a large network. Our simulation results show that with the increase of the network size, more computation time can be saved. For example, for a network with hundreds of nodes, about $40 - 50\%$ of the computation time can be saved. Finally, the system with network coding achieves 25% higher throughput and less network bandwidth consumption compared to the system with the multiple multicast tree approach.

Chapter 3

Peer-to-Peer File Sharing Based on Network Coding

In the last several years, the Internet has witnessed tremendous increase of different types of web-based applications, ranging from web-based file sharing to video broadcasting/conferencing. Web-based applications have gained more and more interests due to the flexibility and easy accessibility. Many such applications involve one source (server) and multiple destinations (receivers). However, due to lack of multicast support over the Internet, these applications usually suffer from the scalability problem, which limits the number of receivers involved. *Peer-to-peer* is a promising technology that can implement multicast at the application layer. By incorporating peer-to-peer technology into web-based applications, the scalability problem can be eliminated.

In this chapter, we consider applying peer-to-peer technology and network coding to file sharing services, in which a web server or a file server holds a file that is requested by multiple clients (receivers). The topology of such an application can be represented by a multicast network. When peer-to-peer technology is used, the receivers help forward the file for each other besides receiving the file.

Peer-to-peer (overlay) networks are a perfect place to apply network coding due to two reasons: the topology of a peer-to-peer network is constructed arbitrarily. It is easy to tailor the topology to facilitate network coding; the nodes in a peer-to-peer network are end hosts which can perform more complex operations such as decoding and encoding than simply storing and forwarding messages. In [34], linear network coding was applied to application layer multicast, in which a rudimentary mesh graph is first constructed, and on top of it a rudimentary tree is formed. Then a multicast graph is constructed, which is a subgraph of the rudimentary mesh and a supergraph of the rudimentary tree. The multicast graph constructed this way is 2-redundant, which means that each receiver has two disjoint paths to the source. By taking advantage of the 2-redundancy property of the multicast graph, a

light-weight algorithm generates a sequence of 2-dimensional transformation vectors which are linearly independent. These vectors are assigned to the edges as their edge functions. However, the paper did not discuss how to process dynamic joining or leaving of peers, while dynamic membership is a common phenomenon in peer-to-peer networks. Moreover, the 2-redundancy property limits the minimum cut of the multicast graph, which in turn limits network throughput.

In [35], random network coding was applied to content distribution, in which nodes encode their received messages with random coefficients. Compared to deterministic network coding, random network coding is inherently distributed. In random network coding, nodes can determine the edge functions of its outgoing edges independently by generating random coefficients for the edge functions. The advantage of random network coding is that there is no control overhead to construct and maintain a linear coding assignment among nodes. However, the global encoding vectors of a receiver's incoming edges may not be linearly independent. In other words, a receiver may not recover the original messages even it receives k or more messages (here k is the multicast capacity of the multicast network). To reduce the probability of failing to decode messages, it is required to encode over a very large field. Another drawback of random network coding is the increased data traffic. As there is no deterministic path for data delivery, all the nodes take part in relaying the data to the receivers even if it is not necessary. As a result, the same message may be transmitted through the same link multiple times.

In this chapter, we aim at providing an efficient and reliable file sharing service over peer-to-peer networks by utilizing network coding. We call it *Peer-to-Peer FilE sharing based on nEtwork coDing*, or *PPFEED* for short. We utilize a special type of network with a regular topology called *combination network*. It was demonstrated in [20] that when the network size increases, this type of network can achieve unbounded network coding gain measured by the ratio of network throughput with network coding to that without network coding. The basic idea of PPFEED is to construct an overlay network over the source and the receivers such that it can be decomposed into multiple combination networks. Compared to [34], our approach can accommodate dynamic membership and construct a much simpler overlay network topology in different k values. Compared to [35], our network coding construction scheme is deterministic, which means that the validity of the network coding assignment is guaranteed. The data traffic is then minimized so that the same messages are transmitted through an overlay link at most once. Also, system reliability is improved dramatically with little overhead. In addition, PPFEED can be extended to support link heterogeneity and topology awareness.

3.1 Deterministic Linear Coding over Combination Networks

One of the advantages of network coding is that it can increase network throughput by achieving multicast capacity. *Network coding gain* is defined as the ratio of throughput with network coding to that without network coding. Clearly, it is always greater than or equal to 1. When applying network coding to a multicast network, the gain depends on the topology of the multicast network. For example, if the topology of a multicast network is a tree, i.e. the source is the root of the tree and the receivers are the leaves of the tree, network coding gain is 1. Intuitively, if each receiver has roughly the same number of disjoint paths to the source, the more overlap among the paths, the larger network coding gain can be achieved.

A combination network is a multicast network with a regular topology. The topology of a combination network is a regular graph which contains three types of nodes: *source node*, *relay node* and *receiver node*. A combination network contains a source node which generates messages, n relay nodes which receive messages from the source node and relay them to the receiver nodes, and C_n^k receiver nodes which receive messages from the relay nodes. There are n links connecting the source node to the n relay nodes respectively. For every k nodes out of the n relay nodes, there are k links connecting them to a receiver node. Since there are a total of C_n^k different combinations, the number of receiver nodes is C_n^k. The capacity of each link is 1. Fig. 3.1 shows a combination network for $n = 4$ and $k = 2$, usually denoted as a C_4^2 combination network. For clarity, the nodes are arranged into three layers: the first layer contains only one node, the source node; the second layer contains n relay nodes; the third layer contains C_n^k receiver nodes.

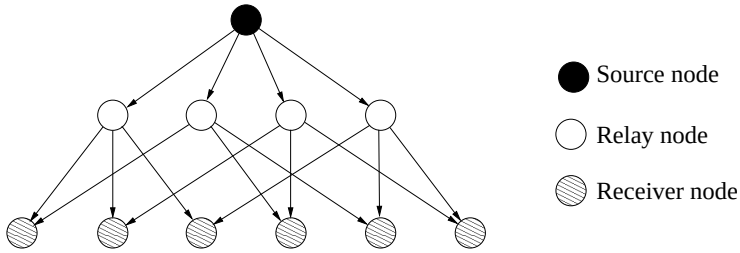


Figure 3.1: Combination network C_4^2.

Combination networks have good performance with respect to network coding gain. It was proved [20] that network coding gain is unbounded when both n and k approach infinity. For a C_n^k combination network, the multicast capacity is k as each receiver node has k disjoint paths to the source node. This implies that to achieve

the multicast capacity in a combination network, the minimum subgraph to deploy network coding is the entire topology of the combination network.

We have discussed the good properties of combination networks. The next issue is how to find a valid linear network coding assignment on a combination network. As mentioned earlier, existing network coding methods can be classified into two categories: random method and deterministic method. Random network coding [36] assigns random mixing coefficients to the edges. This method can only obtain a valid network coding assignment with a probability less than 1. In [16], a polynomial deterministic algorithm was proposed to construct a valid linear coding assignment for any multicast network. However, it is a centralized algorithm which requires all the topology information in advance. Moreover, since it is designed for general topologies, it involves many complex processes such as keeping track of the paths between the source and the receivers and maintaining an extra array to facilitate linear independence test. On the other hand, for a multicast network with a regular topology like a combination network, it is possible to design a tailored linear network coding assignment which takes advantage of the regular topology to achieve both effectiveness and efficiency. In this chapter, we propose a simple deterministic linear coding construction scheme which can guarantee a valid linear network coding assignment and achieve $O(1)$ time complexity.

Before we go into details of our linear network coding construction, we first determine an appropriate value for k, which is the multicast capacity of a combination network. On one hand, given the number of relay nodes n, the network coding gain reaches its maximum when $k = n/2$ [20]. On the other hand, k should not be too large because a large k will increase the link stress and overhead. Therefore, as a tradeoff, in this chapter we only consider combination networks with $n = 4$, $k = 2$ or $n = 6$, $k = 3$ where the network coding gain is 1.5 and 2, respectively. The n and k determine the size of the combination network. As we will see later, the size of the overlay network is independent of the size of the combination network. If we apply the same n and k to different overlay networks of various sizes, the performance difference is very small.

As mentioned earlier, constructing a valid linear network coding assignment is equivalent to assigning each edge a global encoding vector such that the vectors of the incoming edges of a receiver are linearly independent. Suppose the source has k symbols to multicast to the receiver nodes. In a C_n^k combination network, there are a total of $n + k\mathrm{C}_n^k$ edges. The first n edges connect the source node to n relay nodes with global encoding vectors $V_1', V_2', \ldots, V_n'$. For each relay node, there are $k\mathrm{C}_n^k/n$ outgoing edges connecting it to $k\mathrm{C}_n^k/n$ receiver nodes. As each relay node has only one incoming edge, the global encoding vector of its outgoing edge is the global encoding vector of its incoming edge multiplied by a constant. Since the constant does not change the linear independence property, we only consider the case that the

constant is 1. Each receiver has k incoming edges whose global encoding vectors are k vectors out of the n vectors $V_1', V_2', \ldots, V_n'$. For a receiver node to decode the original messages, the k global encoding vectors must be linearly independent. Thus the main issue is how to find a set of n k-dimensional vectors such that every k vectors of the n vectors are linearly independent.

Suppose $GF(q)$ is a given Galois field, where $|GF(q)| = q$, $GF = \{0, 1, 2, \ldots, q-1\}$, $q \geq n$. We give the rules for the linear network coding assignment construction for C_n^2 and C_n^3 combination networks as follows.

1. The linear network coding construction scheme for C_n^2 combination network is to assign vectors $(1, \alpha_1), (1, \alpha_2), \ldots, (1, \alpha_n)$, where $\alpha_1, \alpha_2, \ldots, \alpha_n$ are different symbols in $GF(q)$, to n edges connecting to n relay nodes as global encoding vectors. The global encoding vectors of the edge outgoing from one relay node is the same as the global encoding vector of the edge incoming to the relay node.

2. The linear network coding construction scheme for C_n^3 combination network is to assign vectors $(1, \alpha_1, \alpha_1^2 \mod q)$, $(1, \alpha_2, \alpha_2^2 \mod q)$, $\ldots$, $(1, \alpha_n, \alpha_n^2 \mod q)$, where $\alpha_1, \alpha_2, \ldots, \alpha_n$ are different symbols in $GF(q)$, to n edges connecting to n relay nodes as global encoding vectors. The global encoding vectors of the edge outgoing from one relay node is the same as the global encoding vector of the edge incoming to the relay node.

We have the following theorem concerning the linear network coding assignment.

Theorem 5 *The proposed linear network coding assignment for combination networks is valid.*

Proof[1]: Case 1: $k = 2$. For any two vectors (1, α), (1, β), we evaluate the determinant of matrix $\begin{pmatrix} 1 & \alpha \\ 1 & \beta \end{pmatrix}$. We have

$$\begin{vmatrix} 1 & \alpha \\ 1 & \beta \end{vmatrix} = \beta - \alpha \tag{3.1}$$

Since $\alpha \neq \beta$, the determinant is not equal to 0. We conclude that the two vectors are linearly independent.

[1]Since the mod operation is distributive over addition and multiplication, we omit $\mod q$ in the equations.

Case 2: $k = 3$. For any three vectors (1, α, $\alpha^2 \mod q$), (1, β, $\beta^2 \mod q$) and (1, γ, $\gamma^2 \mod q$), where $\gamma > \beta > \alpha$, the determinant of matrix $\begin{pmatrix} 1 & \alpha & \alpha^2 \\ 1 & \beta & \beta^2 \\ 1 & \gamma & \gamma^2 \end{pmatrix}$ is

$$\begin{aligned} \begin{vmatrix} 1 & \alpha & \alpha^2 \\ 1 & \beta & \beta^2 \\ 1 & \gamma & \gamma^2 \end{vmatrix} &= \beta\gamma^2 + \alpha\beta^2 + \alpha^2\gamma - \alpha^2\beta - \alpha\gamma^2 - \beta^2\gamma \\ &= \beta\gamma(\gamma - \beta) + \alpha\beta(\beta - \alpha) + \alpha\gamma(\alpha - \beta + \beta - \gamma) \\ &= (\beta - \alpha)(\gamma - \beta)(\gamma - \alpha) \end{aligned} \tag{3.2}$$

Since $\alpha \neq \beta \neq \gamma$, the determinant is not equal to 0. We conclude that the three vectors are linearly independent. ■

3.2 Peer-to-Peer File Sharing Based on Network Coding (PPFEED)

In this section, we present the PPFEED scheme that provides a peer-to-peer file sharing service over the Internet. We first give an overview of PPFEED and describe the basic idea of constructing an overlay network composed of multiple combination networks and applying network coding on the overlay network. Then we go into details by describing the processes of peer joining/leaving and data dissemination. Finally, we discuss some optimizations which can improve the reliability and resilience of the system.

3.2.1 Overview of PPFEED

We assume that there is a server holding the file to be distributed. Peers interested in the file form an overlay network through which the file is distributed. The construction of the overlay network is based on the idea of combination networks. Similar to the combination networks, there are two system parameters n and k in the overlay network. The server encodes the file into n different messages using the linear network coding assignment given in the previous section. Thus any k messages out of the n messages can be used to decode the original file. The peers are divided into n disjoint groups and each group is assigned a unique *group ID*. Each group of peers is responsible for relaying one of the n encoded messages. To accelerate the distribution of the messages, peers in the same group are connected by an unstructured overlay network according to some loose rules. Each peer in a group is connected to at least other $k - 1$ peers which are in $k - 1$ different groups respectively.

Although the topology is constructed based on combination networks, it is more a mesh network than a multi-tree network. If we color peers with n different colors, the entire overlay topology can be looked as a mesh with the constraint that each peer must be connected to at least k peers with k different colors. Therefore, PPFEED enjoys the scalability and resilience of a mesh network. In section 3.4, we will quantify its performance through simulations.

Fig. 3.2 shows an example of the overlay network constructed by PPFEED for $n = 3$ and $k = 2$. There are three groups of peers which are colored in black, blue and red, respectively. The colored links represent corresponding messages with arrows pointing to the transmission directions. We can see that each peer has at least two links with different colors pointed to itself. All the peers are able to decode the received messages.

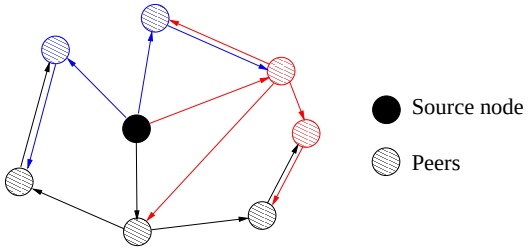


Figure 3.2: An example overlay network constructed by PPFEED.

The overlay links are added to the system on demand as new peers join the system. There are two key issues in the overlay network construction: first, how to connect the peers in the same group; second, how to connect the peers in different groups.

The first issue involves how to distribute the n messages in a scalable way. When the number of peers is small, it is acceptable to connect all the peers to the source node, i.e. the server. When the number of peers increases, the peers resort to each other instead of the server to distribute the messages. Here we adopt some loose rules similar to the unstructured peer-to-peer network Gnutella [37]: for a newly joined peer, it is responded by a random list of the existing peers, and the new peer creates connections with these peers. The reason is three folds. First, it is resilient to peer join. A new peer can be connected to any existing peer. Second, the overlay construction and maintenance overhead is low. There are no constraints on the overlay topology as long as it is connected. If some peers leave the system, it is convenient for the rest of the peers to reconnect through the server. Third, it is easy to flood messages on the resulting overlay network.

The second issue involves how the peers receive other $k-1$ messages to decode the messages. As any k different messages can be used to decode, we only need to

find $k-1$ peers in different groups. Since there are $n-1$ eligible groups, we first connect the peer to $k-1$ random peers in different groups. Then perform a local topology adjustment to find the $k-1$ peers such that the average latency between the peer and $k-1$ peers is minimized.

Some erasure codes, such as Reed-Solomon or Tornado, have similar property to PPFEED in terms of decoding the original messages from a group of encoded messages. In [38], the authors have applied Tornado code to reliable distribution of bulk data to a variety of heterogeneous population of receivers. The erasure codes encode the original messages into a sequence of encoded messages such that any subset of the encoded messages can be used to reconstruct the original messages as long as the size of the subset is sufficiently large. However, they have different objectives and applications. The erasure codes are designed to avoid retransmissions, while network coding is used to improve system throughput. In some applications, retransmissions may be impossible or undesirable. For example, in multicast applications, retransmission requests may greatly overwhelm the sender if the group size is large. In satellite networks, where the back channel has high latency and limited capacity, retransmission requests are costly and unreliable. With the erasure codes, the sender can keep sending the sequence of encoded messages regardless of whether the receivers receive them or not. As long as the receivers receive enough encoded messages, they can extract the original messages from the received messages. On the other hand, network coding is applied when multiple flows share a common link where the messages from different flows can be mixed together to save bandwidth. Network coding can be applied to multicast networks or wireless networks where messages sharing common links can be encoded together to improve system throughput.

As mentioned earlier, we consider the combination networks with $n = 4$, $k = 2$ or $n = 6$, $k = 3$ in this chapter. When we apply PPFEED to a large overlay network, as n and k are fixed, the number of peers in the same group increases. Each peer is still connected to the other $k-1$ peers in different groups. The scalability of PPFEED is derived from the scalability of the overlay network composed of peers within the same group. As described earlier, the overlay network is constructed by some loose rules to accommodate scalability. Therefore, PPFEED can be applied to overlay networks of various network sizes without performance degradation.

Next, we discuss how PPFEED works in more detail.

3.2.2 Peer Joining

We assume that the server is well-known whose IP address is known to all the peers by some address translation service such as DNS. When a peer wants to retrieve a file hosted by the server, it initiates a join process by sending a JOIN request to the

server.

The server keeps track of the number of peers in each group and maintains a partial list of existing peers in the group and their IP addresses. The purpose of maintaining a partial list instead of a full list is to achieve a balance between scalability and efficiency. On one hand, when the network size is large, it is resource-consuming to maintain a full list of peers in the server. On the other hand, if the list is too short, it may cause much longer latency for new peers to join when the peers in the list crash. Although the server is responsible for bootstrapping the peers, it will not be the bottleneck of the system because once each peer receives the list, it communicates with other peers for topology construction and data dissemination. When the server receives a peer's join request, it assigns the peer to a group such that the numbers of peers in different groups are balanced. Then the server sends the list of peers of that group to the joining peer and updates the number of peers in that group.

After receiving the list of peers, the new peer will contact them and create overlay links with them. These peers are called intra-neighbors of the new peer because they are within the same group. In contrast, the neighbors which are in different groups are called inter-neighbors. The new peer asks one of its intra-neighbors picked randomly to provide a list of its inter-neighbors. The new peer then takes the list of peers as its inter-neighbors.

The topology of the peer-to-peer network can be considered as a combination of multiple unstructured peer-to-peer networks, each of which is composed of the peers within the same group. The topology within one group is arbitrary as long as it is connected. The only constraint is on the edges between different groups. It is required that each peer is connected to at least $k-1$ peers in $k-1$ different groups respectively.

3.2.3 Local Topology Adjustment

As the overlay topology is formed by always connecting the new peers to a random list of existing peers in the network, the overlay links among peers may not be good with respect to latency or link stress. To alleviate the performance degradation, we optimize the overlay topology by a process called *local topology adjustment*.

The idea of local topology adjustment is to replace the direct neighbors with peers with better performance through a local search. Each peer periodically sends QUERY messages to its neighbors. There are two types of QUERY messages: one for intra-neighbor searching and one for inter-neighbor searching. Suppose peer u sends out a QUERY message to one of its neighbors, say, v. After receiving the QUERY message, node v will send a RESPONSE message back to u and a QUERY-2 message to each of its neighbors except for u. The RESPONSE mes-

sage includes a TIMESTAMP field which records the time when node v sends the RESPONSE message. After node u receives the message, it calculates the latency between u and v by the difference between the time when the RESPONSE message is received and the time indicated by the TIMESTAMP in the RESPONSE message. The QUERY-2 message includes the IP address of node u. After a neighbor of v, say, w, receives the QUERY-2 message, it sends a RESPONSE-2 message to node u. It also includes a TIMESTAMP field which records the time when w sends the RESPONSE-2 message. After node u receives the message, it calculates the latency between u and w by the difference between the time when the RESPONSE-2 message is received and the time indicated by the TIMESTAMP in the RESPONSE-2 message. If the latency between u and w is less than that between u and v, node w will replace node v and become the neighbor of node u.

Fig. 3.3 illustrates an example for local topology adjustment. In the figure, an overlay topology with $n+2$ nodes connected as a list. We use the distance between nodes to represent the latency roughly. At the beginning, node v_1 is the neighbor of node u. After n steps of local topology adjustment, node w becomes the neighbor of node u as the topology on the right shows.

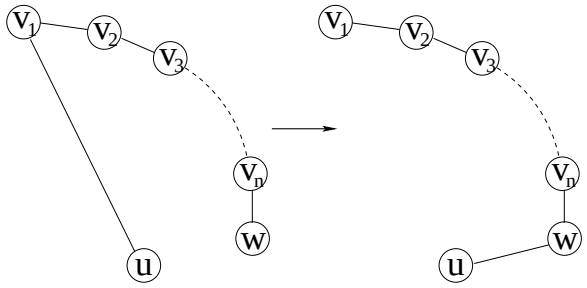


Figure 3.3: Illustration of local topology adjustment.

Local topology adjustment is a distributed light-weight overlay topology optimization process. It can detect a better neighbor after searching the nodes nearby with a search radius of 2. The process is done periodically as the topology evolves. However, in some cases, local topology adjustment cannot find the best neighbor with the shortest latency. Fig. 3.4 shows one of these cases, where although node u and node w are close to each other, they cannot detect each other through local topology adjustment. In Section 3.3.2, we describe a landmark based solution to solve this topology mismatch problem systematically.

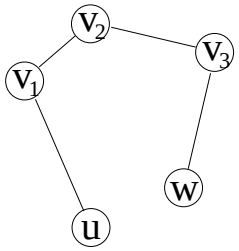


Figure 3.4: An example that local topology adjustment does not work.

3.2.4 Peer Leaving

There are two types of peer leaving: friendly or abruptly. Friendly leaving means that the leaving peer initiates a leaving process so that the system is aware of its leaving and can make necessary updates accordingly. Abruptly leaving means that the leaving peer leaves the system without any notification, mainly due to link crash or computer crash.

For the friendly leaving, the leaving peer will initiate a leaving process by sending LEAVE messages to both of its intra-neighbors and inter-neighbors. The leaving of the peer may impair the connectivity between peers in the same group. To rebuild the connectivity, the intra-neighbors will initiate the join process with the group ID in the join request. After receiving a join request with a designated group ID, the server temporarily acts as one of the intra-neighbors of the peer to guarantee the connectivity. The topology is further adjusted by the local topology adjustment process. Here the server acts as a connection "hub" for the peers that were connected to the leaving peer. This may increase the data forwarding burden on the server temporarily. Nevertheless, the situation will be improved after the local topology adjustment process is completed. Moreover, it can achieve strong robustness with little control overhead. For example, it can handle concurrent peer leavings. The connectivity is rebuilt after the server receives all the join requests from the intra-neighbors of the leaving peers.

After receiving the LEAVE message from the leaving peer, the inter-neighbors will ask one of its intra-neighbors for a new inter-neighbor to replace the leaving one. In addition, the leaving peer will also send a LEAVE message to the server. This LEAVE message will make the server update the number of peers in the group.

For the abruptly leaving, peers send HELLO messages to its neighbors periodically and maintain a HELLO timer for each neighbor. Receiving a HELLO message triggers a reset of the corresponding HELLO timer. The neighbors detect the abruptly leaving by the timeout of the HELLO timer. Similarly, intra-neighbors initiate the join process after detecting the sudden leave. Inter-neighbors ask their

intra-neighbors for a replacement of the left peer. Moreover, one of the neighbors is chosen to send a LEAVE message to the server on behalf of the abruptly leaving peer so that the server can update the number of the peers in the group. To minimize the selection overhead, we choose the inter-neighbor whose group ID is next to that of the leaving peer to perform this task. For example, if we assume the encoding vector for the group corresponding to the leaving peer is $(1,\ \alpha)$ (or $(1,\ \alpha,\ \alpha^2\ mod\ q)$), inter-neighbor corresponding to the group whose encoding vector is $(1,\ (\alpha+1)\mod n)$, or $(1,\ (\alpha+1)\mod n,\ ((\alpha+1)\mod n)^2\mod q)$ is chosen.

3.2.5 Data Dissemination

Before sending out the file, the server needs to encode the file. The encoding is over a Galois field $GF(q)$. The file is divided into multiple blocks with each block represented by a symbol in $GF(q)$. The first k blocks are encoded and then the second k blocks, and so on. In the case that there are not enough blocks to encode, the file is padded with zero string. Assuming the field size is $q(>n)$, each k blocks are encoded into n different messages using the linear network coding assignment given in Section 3.1. Therefore, any k messages of these n messages can be used to decode the original k blocks.

After encoding, the server sends the encoded n messages to the peers in the n groups respectively. The group ID and the encoding function form a one-to-one mapping. Peers can learn which messages they receive based on the sender of the messages: if the sender is the server, the messages correspond to the group ID of the peer itself; if the sender is a peer, the messages correspond to the group ID of the sender.

The peers forward all the messages they receive based on the following rules:
Rule 1. If the message comes from the server, the peer forwards it to all its intra-neighbors and inter-neighbors.
Rule 2. If the message comes from one of its intra-neighbors, the peer forwards it to other intra-neighbors except for the sender.
Rule 3. If the message comes from one of its inter-neighbors, the peer does nothing.

Peers forward messages in a push style, which means that the messages are forwarded under the three rules as soon as they arrive at a peer. A peer decodes the messages right after it receives k different messages. Thus data dissemination is fast and simple in PPFEED.

3.2.6 Improving Reliability and Resilience to Churn

Besides the throughput improvement, PPFEED can provide high reliability and high resilience to *churn* which refers to frequent peer joins or leaves. All we need to do

is to add a redundant overlay link for each peer.

In the previous subsections, each peer is connected to $k-1$ inter-neighbors. The $k-1$ messages received from them plus the message received from the source or the intra-neighbors should be sufficient to decode the original file blocks. However, no links are 100% reliable. In case that the messages are lost or damaged, the sender has to retransmit the messages until they are received correctly. Clearly, retransmission will cause longer latency, larger buffer size and reduce system throughput. PPFEED can reduce the retransmission probability by introducing a redundant link to each inter-neighbor. Now each peer is connected to k instead of $k-1$ peers in different groups. If one of the links fails, the peer can still decode the original file blocks based on the remaining $k-1$ messages. The failure probability of this scheme can be quantitatively analyzed as follows. Suppose each overlay link will fail with probability $1-p$. In the old scheme, the peer fails to decode with probability $P_{failure} = 1-p^{k-1}$, while in the improved scheme, the peer fails to decode with probability $P'_{failure} = 1-(p^k + k(1-p)p^{k-1})$. The ratio of the two probabilities is $P_{failure}/P'_{failure} = (1-p^{k-1})/(1-(p^k+k(1-p)p^{k-1}))$. Fig. 3.5 plots the curves of the ratio for different p values when $k=2$ and $k=3$. We can see that when $p=0.9$, the failure probability of the old scheme is about 10 times of the improved scheme.

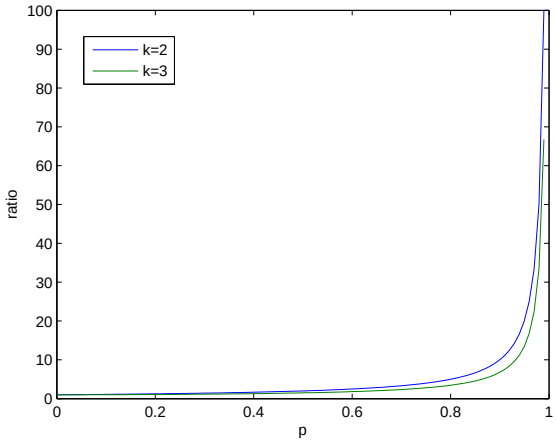


Figure 3.5: Failure probability ratio of the old scheme to the improved scheme.

Another advantage of connecting peers with more inter-neighbors than necessary is that it can improve the system resilience to churn. The rationale is similar to the reliability improvement. For example, if we connect a peer with k other peers, it can tolerate one peer's leave without affecting the download speed. Peers experiencing unstable neighborhood should be connected to more than one redundant

inter-neighbors to alleviate the performance degradation.

3.3 Some Extensions

In this section, we extend the proposed scheme to support some extra features, such as link heterogeneity and topology awareness.

3.3.1 Support Link Heterogeneity

Link heterogeneity is a common phenomenon in networks. In peer-to-peer networks, link heterogeneity refers to the fact that peers have different access link capability. Common access links include dial-up connections, ADSL and high speed cable. The ratio of the link capacities of the fastest link to the slowest link can be more than 1000.

Due to the link capacity imbalance, if a peer with high link capacity is receiving messages from a peer with low link capacity, its download speed is upper bounded by the download speed of the low link capacity peer. In other words, the bandwidth of the high link capacity peer is wasted.

We now consider how to maximize the utilization of link capacity of each peer. Suppose the bottleneck is always on the access links. Thus we can construct as many as possible overlay links connected to the peer as long as its access link capacity permits. As mentioned earlier, the overlay construction between the peers in the same group is arbitrary. We take advantage of this property to construct an overlay network such that the number of links of a peer is proportional to its link capacity. For example, if a peer has much higher link capacity than other peers that have low link capacities, the topology of the overlay network is a star with the high link capacity peer in the center.

In practice, every peer first reserves link capacity for the k links that are used to connect inter-neighbors. The list kept in the server includes the peers with higher link capacities. Peers periodically send messages to the server that include their remaining link capacities for the server to update the list.

3.3.2 Support Topology Awareness

Since overlay networks are logical networks on top of physical networks, the overlay links are logical links. Each logic link is composed of one or more physical links. The overlay links are added arbitrarily as needed. As a result, the topology of the overlay network may be different from the topology of the physical network. Two nodes which are close to each other in the overlay network may be far away in the physical network. Such topology mismatch may greatly increase the *link stress*

and degrade the performance. Here link stress is defined as the number of copies of a message transmitted over a certain physical link.

In Section 3.2, we mentioned that peers can use local topology adjustment to find peers with better performance than current neighbors. If we define the performance to be the latency, then peers can dynamically adjust the local overlay topology to alleviate the mismatch. However, a problem with this method is that its convergence speed is slow and its accuracy is limited. We need a more efficient and accurate method to minimize the mismatch. Here we propose a *topology clustering scheme* by adopting the idea of the binning scheme introduced in [39] to construct a topology aware overlay network.

In this scheme, the server is responsible for choosing some peers as *landmarks*. Each new peer will receive the list of landmarks before it receives the list of peers. The new peer sends probe messages to the landmarks to learn the distances between them and itself. The landmark peers are listed in an ascending order of distances. The ordered list acts as the coordinate of the peer in the system. The coordinate is sent to the server, then the server assigns the new peer a group ID based on its coordinate. Peers with the same coordinates form a cluster. The heuristic rules of assigning peers to groups are: each cluster has at least k different groups; the peers in the same group should span as few clusters as possible. The first rule is to guarantee that peers can receive enough messages to decode within its cluster. The second rule is to minimize the number of links across clusters. To implement this, the server should keep track of the numbers of different groups in each cluster. After receiving a peer's join request and its coordinate, the server first checks whether the corresponding cluster has k different groups. If yes, the peer is assigned to one of the groups such that the numbers of peers of different groups are as balanced as possible. Otherwise, the peer is assigned to a group which is different from the existing groups in the cluster.

In addition, every two landmark peers should not be too close to each other. A new peer cannot be a landmark if its coordinate is the same as one of the existing landmark peers. A landmark peer is removed from the landmark peers if it has the same coordinate as another landmark peer.

3.4 Performance Evaluations

In this section, we study the performance of PPFEED through simulations. We compare our scheme with a peer-to-peer multicast system called Narada [12] and a peer-to-peer file sharing system called Avalanche [35]. Narada first constructs an overlay mesh spanning over all the peers. The overlay mesh is a richer connected graph which satisfies some desirable performance properties. The multicast tree is a spanning tree on top of the mesh and is constructed on demand of the source peer.

Avalanche is a peer-to-peer file sharing system based on random network coding. We choose these two schemes as the comparison counterparts in order to evaluate the benefit a tailored deterministic network coding brings.

The simulation adopts following three performance metrics:

Throughput: throughput is defined as the service the system provides in one time unit. Here we let different systems transmit the same file, thus throughput can be simply represented by the time consumed by the transmission. The shorter the time consumed, the higher the throughput. We start transmitting the file from time 0. Then the consumed time is the time when the peers finish receiving the file, denoted by *finish time*.

Reliability: this performance metric is used to evaluate the ability of the system to handle errors. We use the *number of retransmissions* to characterize this ability. A system with higher reliability will have a smaller number of retransmissions, and thus higher throughput.

Link stress: link stress is defined as the number of copies of the same message transmitted through the same link. It is a performance metric that only applies to an overlay network due to the mismatch between the overlay network and the physical network. We use it to evaluate the effectiveness of the topology awareness improvement and the efficiency of the system.

We study the performance of the system in four different configurations:

(i) Baseline configuration. In this configuration, peers have uniform link capacities, and overlay links are constructed randomly. The file is sent after the overlay network is formed.

(ii) Dynamic peer join/leave configuration. In this configuration, peers have uniform link capacities, and overlay links are constructed randomly. Peers join the system during the file transmission and stay in or leave the system after downloading the whole file.

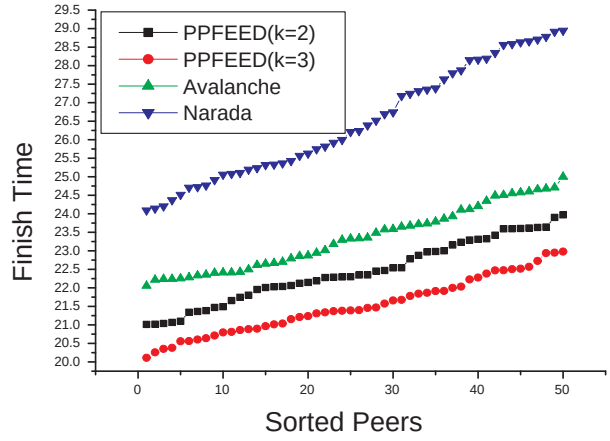
(iii) Heterogeneity configuration. In this configuration, peers have heterogeneous link capacities, and overlay links are constructed by taking into consideration of link heterogeneity. The file is sent after the overlay network is formed.

(iv) Topology awareness configuration. In this configuration, peers have uniform link capacities, and overlay links are constructed by taking into consideration of topology mismatch. The file is sent after the overlay network is formed.

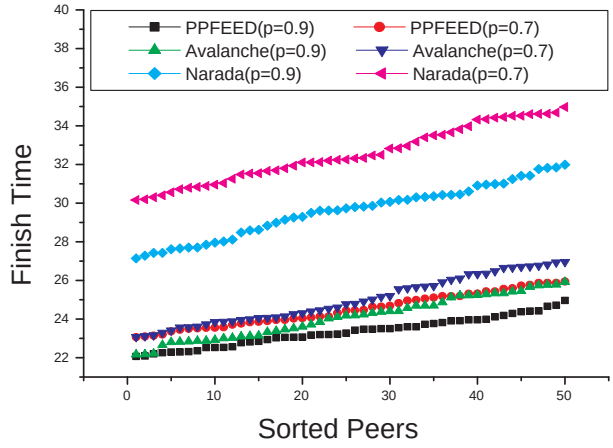
The network topologies are random graphs generated by GT-ITM [27]. We conducted the same simulations for $k = 2$ and $k = 3$. In the rest of this section, we will omit the result for $k = 2$ when it is similar to that of $k = 3$.

3.4.1 Baseline Configuration

In the baseline configuration, peers have uniform link capacities. The simulation is divided into two steps. First, overlay construction period: A number of random nodes are picked to join the system in sequence. Second, file transmission period: The server sends the file to the overlay network.



(a)



(b)

Figure 3.6: Baseline configuration. (a) Finish time; (b) Finish time with link failures.

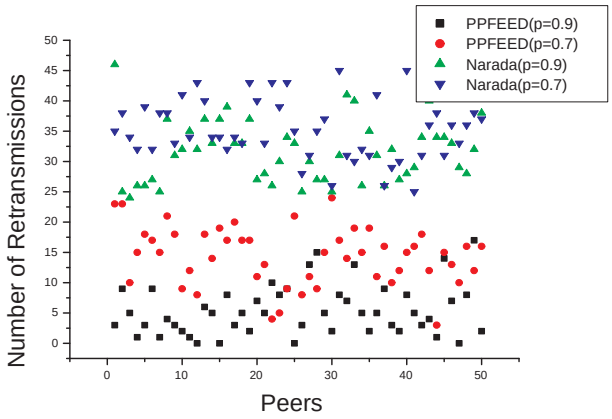
We plot the finish time curves of PPFEED and Narada in Fig. 3.6(a). We simulate PPFEED with different k values as shown in the figure. The finish time

of nodes is sorted in an ascending order. It can be seen that the average finish time of PPFEED is $15 - 20\%$ shorter than that of Narada and $8 - 10\%$ shorter than Avalanche. We notice that the finish times of Narada has a larger variance than that of Avalanche and PPFEED. This is because that in Narada, the two peers with the biggest difference in finish time are a child of the root and a leaf, respectively. This difference may be very large depending on the overlay topology. On the contrary, Avalanche and our scheme construct a mesh to distribute the file. As a result, the distance between the highest level peer and the lowest level peer is shortened. From the figure we can see that the throughput of PPFEED is higher for $k = 3$ than that for $k = 2$. This can be explained by the fact that when $k = 3$, each peer is connected to more peers. Thus the download capacity of peers can be better utilized.

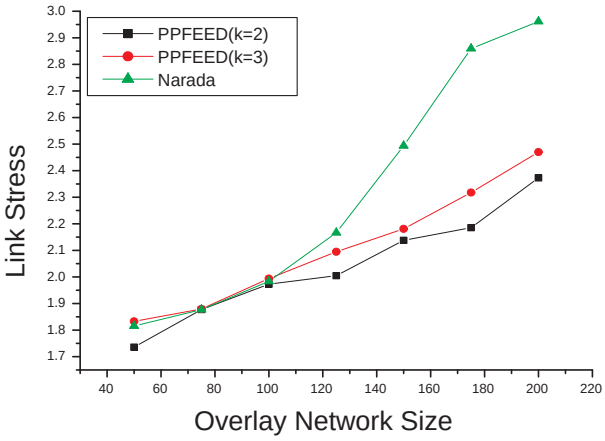
Fig. 3.6(b) shows the finish time when we set the physical link failure probability to $1 - p$. Note that the link failure probabilities of different physical links are independent. In the analysis in Section 3.2.6, we simply assumed that the link failure probability of an overlay link is $1 - p$ to simplify the analysis. However, the link failure probabilities of overlay links are dependent due to sharing common physical links. Here we use link failure probabilities of physical links to simulate real networks more accurately. We can see that the finish times of PPFEED and Avalanche are much shorter than that of Narada. With the same link failure probability, the finish time of PPFEED is slightly shorter than that of Avalanche. Compared to the previous simulation result without link failure probabilities, the increase of finish time is the least for Avalanche and the largest for Narada. The reduction of throughput is little for both Avalanche and PPFEED when physical links are unstable.

The number of retransmissions is shown in Fig. 3.7(a). As Avalanche does not require retransmission, we only plot the curves for PPFEED and Narada. We use colored dots to denote the numbers of retransmissions of peers. We can see that Narada needs more retransmissions than PPFEED. The less the p is, the more retransmissions are needed. The average number of retransmissions of PPFEED is about 5 when $p = 0.9$ while that of Narada is about 30. Both Fig. 3.6(b) and Fig. 3.7(a) reveal that PPFEED has a good fault tolerance ability.

Each packet in Avalanche is unique, the link stress of Avalanche is always 1. We compare the link stresses of PPFEED and Narada as shown in Fig. 3.7(b). When the number of peers is small, PPFEED and Narada have similar link stress. However, when the number of peers is beyond 100, the links stress of PPFEED is less than Narada. The reason for this is similar to that for the larger finish time variance of Narada. If we track a message in Narada, the paths it travels through form a tree spanning all the peers. If we track a message in PPFEED, the paths it travels through form a mesh spanning a portion of the peers (roughly peers if the overlay network is well balanced, where $1/n$ is for the peers within the same group,



(a)



(b)

Figure 3.7: Baseline configuration. (a) The number of retransmissions with link failures; (b) Link stress.

$(k-1)/n$ is for the peers in different groups). Fewer overlay links will reduce the probability that the same message travels through the same physical link. The link stress of PPFEED is higher when $k = 3$ than $k = 2$. When k is larger, the number of peers in the same groups is reduced. On the other hand, each peer is connected to more peers in different groups. As a result, the increased links between different groups outnumber the reduced links due to the reduced group size. Thus the link stress is increased.

3.4.2 Dynamic Peer Join/Leave Configuration

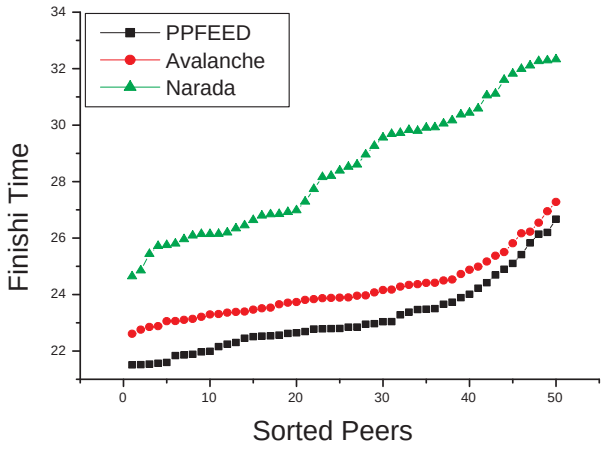
In this configuration, we allow peers to join and leave the system during the file transmission to evaluate the system resilience to churn. The simulations are conducted in two scenarios: First, we let peers join the system randomly and the file transmission starts as long as there are peers requesting it. After receiving the file, peers stay in the system; Second, the peers still join the system randomly. When a peer successfully receives the file, it leaves the system right away. In both scenarios, we calculate the finish time of a peer by the difference between the time it finishes downloading the file and the time it joins the system.

Fig. 3.8(a) shows the finish time comparison when peers stay in the system after receiving the file. We can see that the average finish time of all three schemes increases compared to the baseline configuration. For PPFEED and Avalanche, this is due to the lack of peers which help forwarding the file. For Narada, this is due to the join latency of peers. The increase of average finish time for Narada is more than that of PPFEED or Avalanche, which indicates that peer-to-peer systems achieve better resilience to dynamic joins than tree-based approaches. The peers with different finish times are not evenly distributed as in the baseline configuration. The number of peers with larger finish time increases. The largest finish time for Avalanche is close to that of PPFEED. This is because PPFEED needs to download the file from k peers from k different groups while Avalanche has no such requirements.

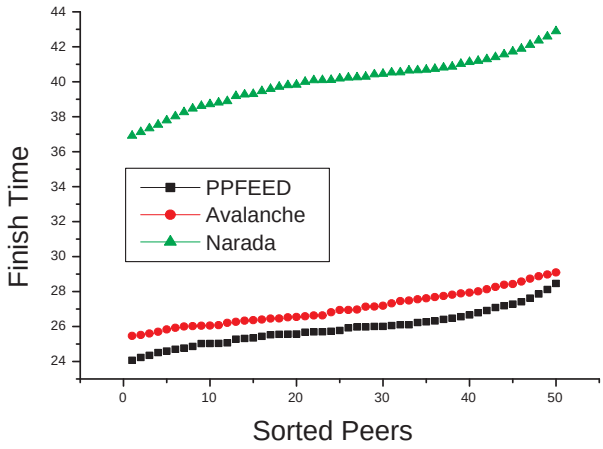
Fig.3.8(b) shows the finish time comparison when peers leave the system after receiving the file. We can see that the average finish times of PPFEED and Avalanche are increased by around 15% while that of Narada is increased by around 50%. Tree-based approaches are extremely vulnerable to churn as each departure will disconnect all the downstream peers and the tree needs to be rebuilt. Both PPFEED and Avalanche have similar resilience under the churn. However, PPFEED achieves slightly higher throughput. It indicated PPFEED achieves great resilience under dynamic peers join/leave. Although PPFEED adopts deterministic network coding, the overlay topology in PPFEED is quite flexible. In addition, by adding redundant links, the resilience can be improved dramatically.

3.4.3 Heterogeneity Configuration

Now we evaluate the ability of PPFEED to handle peers with heterogeneous link capacities. The random topologies generated by GT-ITM are flat random graphs with high speed links. Peers are added to the nodes randomly with each peer connected to one node by an access link and the access link is set to a certain link capacity. We set the peers with heterogeneous link capacities such that $1/3$ with the highest link capacities, $1/3$ with the lowest link capacities and $1/3$ with the medium link



(a)

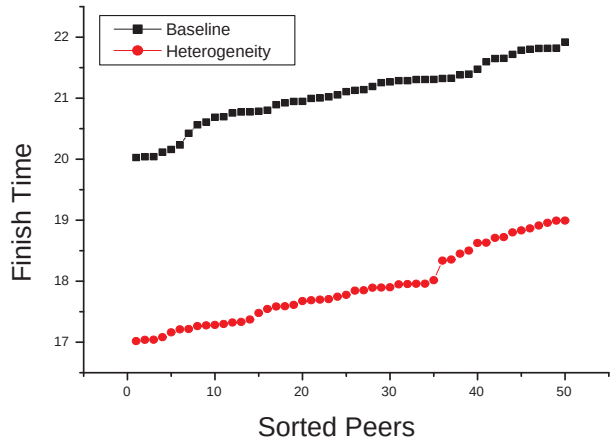


(b)

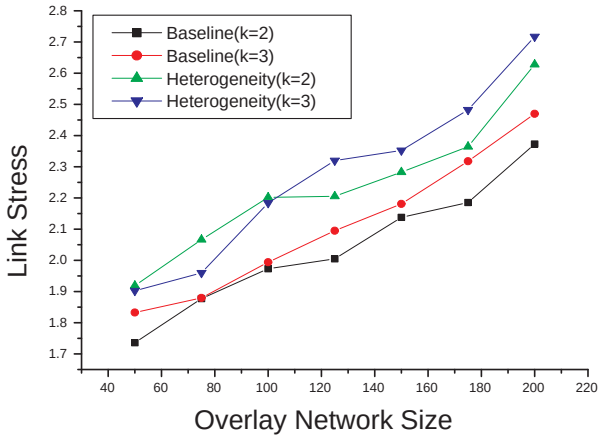
Figure 3.8: Finish time of the dynamic peer join/leave configuration. (a) Peers stay in the system after receiving the file; (b) Peers leave the system after receiving the file.

capacities. The highest link capacity is 10 times of the lowest one.

Fig. 3.9(a) shows the finish time comparison of the heterogeneity configuration between the overlay construction with heterogeneity consideration and without heterogeneity consideration. We can see that the finish time with heterogeneity consideration is much shorter than the baseline which does not consider heterogeneity. However, the variance of the finish time is almost the same. It indicates that the



(a)



(b)

Figure 3.9: Heterogeneity configuration. (a) Finish time; (b) Link stress.

peers with higher link capacities are helpful to increase system throughput, but they cannot reduce the finish times of themselves.

Fig. 3.9(b) shows the link stress comparison. In contrary to the finish time, the link stress with heterogeneity consideration is larger than that without heterogeneity consideration. One reason is that the access links of the high capacity peers are used by many other peers to construct overlay links. As a result, the messages sent by a high capacity peer are more likely transmitted through the same physical

link. When the size of the overlay network is small, peers are distributed around the network sparsely. The link stress is mainly determined by the positions of the peers. In some cases, the link stress when $k = 2$ is even greater than that when $k = 3$.

3.4.4 Topology Awareness Configuration

We now study the performance of PPFEED when considering the physical network topology during the overlay network construction.

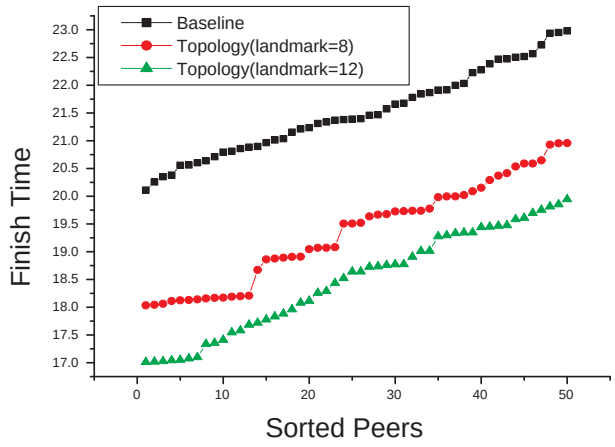
Fig. 3.10(a) shows the finish time at different number of landmarks. The highest curve is the same curve in the baseline configuration when $k = 3$. We can see that topology clustering reduces the finish time by about 10% compared to that without topology clustering. Increasing landmarks can increase the accuracy of topology clustering, thus the finish time is shortened. We notice that the curve of the finish time when the number of landmarks is 8 has a staircase shape. This is because that the peers in the same cluster may finish receiving the file at roughly the same time. While the finish times between different clusters may be longer. When the number of landmarks is 12, the cluster size is reduced. Peers may not receive $k-1$ different messages within the same cluster, thus the staircase disappears.

We set the physical link failure probability to $1-p$. Fig. 3.10(b) shows the number of retransmissions. Compared to the baseline configuration, the average number of retransmissions of the topology awareness configuration is slightly smaller. Generally speaking, the improvement of topology clustering on the number of retransmissions is small. The main reason of the improvement is due to redundant links.

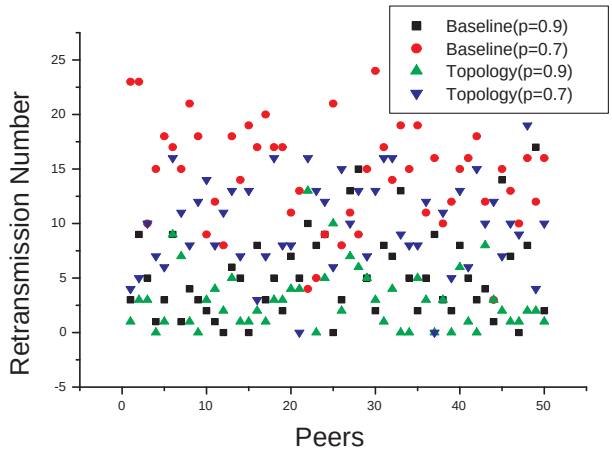
One of the biggest advantages of topology clustering is to reduce the link stress. From Fig. 3.11, we can see that the link stress of the topology awareness configuration is reduced by about 37% compared to that of the baseline configuration when the number of landmarks is 12. With the increase of the overlay network size, the difference is even bigger. Increasing landmarks makes the link stress smaller.

3.5 Summary

In this chapter, we have proposed a peer-to-peer file sharing scheme based on network coding, PPFEED. The scheme can serve as a peer-to-peer middleware created within the web services framework for web-based file sharing applications. Compared to other file sharing schemes, the advantages of our scheme can be summarized as follows. (a) Scalability. Files are distributed through a peer-to-peer network. With the increase of the network size, the total available bandwidth also increases. (b) Efficiency. The linear network coding construction scheme is deterministic and easy to implement. There is no requirement for peers to collaborate



(a)



(b)

Figure 3.10: Topology awareness configuration. (a) Finish time; (b) The number of retransmissions.

to construct the linear network coding assignment on demand. All the peers need is the mapping between the group ID and the encoding function, and this mapping does not change with time. Compared to random network coding, the receiver can always recover the original messages after receiving k different messages and the data dissemination is more efficient as data messages are transmitted through the same overlay link at most once. (c) Reliability. The redundant links can greatly

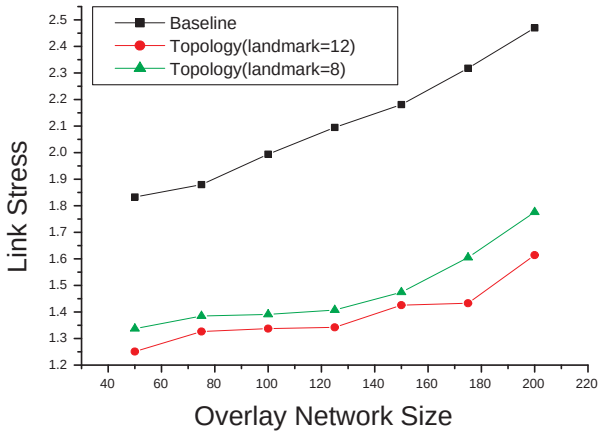


Figure 3.11: Link stress of the topology awareness configuration.

improve the reliability of the system with little overhead. (d) Resilience. Churn is a common problem in overlay networks. By adding redundant links, the negative effect of churn is alleviated. (e) Topology awareness. Simulation results show that the proposed topology clustering scheme can greatly reduce link stress and improve throughput. (f) Heterogeneity support. In case that links have different link capacities, PPFEED can arrange the overlay topology to maximize the utilization of each peer's link capacity.

Chapter 4

Network Coding for Heterogeneous Peer-to-Peer Streaming Systems

Live media steaming systems are important Internet applications and contribute a significant amount of today's Internet traffic. Like bulk data distribution systems, live media streaming systems usually involve a server which hosts the media content and all the clients request the media content from the server. However, there is a fundamental difference between them. Live media streaming systems require real-time data delivery and can tolerate data loss to some extent. While bulk data distribution systems require reliable data delivery and can tolerate delay to some extent. As we will see later, this requirement difference leads to different consideration in system design.

A naive way to implement such systems is to create a unicast connection between each client and the server. However, this approach scales poorly because a surge of client population could easily overwhelm the media server's resources or bandwidth. Network layer multicast provides an efficient way for one-to-many communication but its limited deployment on the Internet makes it impractical. A new technology called CDN (Content Distribution Network) is applied to media streaming. Usually CDN deploys a number of CDN servers at the edge of the Internet and clients request media content from the closest CDN servers. CDN servers have dedicated storage space and out-bound bandwidth to support high-quality media streaming. However, as CDN distributes the media server's load only to multiple CDN servers, it can only alleviate the scalability problem instead of solving it completely. In addition, CDN servers are expensive to deploy and maintain. For example, users have to pay a subscription fee to watch streaming videos from CNN.com. In recent years, peer-to-peer technology has been considered as a promising candidate for media streaming. Peer-to-peer systems build an overlay topology on top of the physical network where the nodes, also known as peers, are the end hosts owned by individuals or companies and the links between peers have only logical

meanings and are realized by finding a physical path connecting the two peers. The flexibility of the overlay topology construction and the decentralized control of the peer-to-peer network make it suitable for distributed applications.

When applied to media streaming systems, peer-to-peer technology can completely eliminate the scalability problem caused by the server-client transmission model. Peer-to-peer media streaming systems employ the clients (peers) to help forward the media content, that is, the systems leverage the upload bandwidth of peers to distribute the media content. Peers forward the media content after they receive it from the server or other peers. As most peers are individual computers which are connected to the Internet through access links with limited bandwidth, a practical peer-to-peer streaming system should meet the following requirements:

1. Accommodation of link heterogeneity. Most peers are individual computers connected to the Internet through access links. Due to different access link technologies (ADSL, Ethernet, Wireless LAN, etc), peers have diverse upload and download bandwidths. The overlay topology should utilize the upload bandwidths in an efficient way to meet different requirements of playback quality by the pees.

2. Low end-to-end delay. Media streaming applications are real-time applications which are sensitive to end-to-end delay. The overlay topology should allow peers to receive the media content within a limited latency after the server sends it.

3. Adaptation to dynamic peers join/leave. Peers can join or leave the media streaming system at will. As the join order is random, the overlay topology may become far from the optimal one. The overlay topology should adapt itself to the changing peers in the system.

There has been much work on peer-to-peer streaming systems in the literature in recent years, see, for example, [11, 40, 41, 42, 43, 44, 45]. Based on the overlay topology construction, these peer-to-peer streaming systems can be categorized into two types: tree-based and mesh-based. Tree-based peer-to-peer streaming systems build one or more trees rooted at the server. The media content flows from the server to the peers along the trees. Multiple trees are employed to make use of the uplink bandwidth of the leave peers. Mesh-based peer-to-peer streaming systems build a loosely connected mesh where each peer establishes connection with several neighbors. Peers exchange the availability information of the media content with their neighbors and help each other to deliver the media content. Most of the works take little or no consideration of huge diversity of the link bandwidth of peers.

With the rapid expansion of the Internet, more and more individual computers are connected to the Internet which become potential peers for peer-to-peer

streaming systems. Diverse and limited peer access link bandwidth is an important characteristics when these individual computers join a peer-to-peer streaming system. First, as different peers have different download bandwidths, it is desirable that peers with higher download bandwidth receive the media content at a higher bit rate while peers with lower download bandwidth receive the media content at a lower bit rate. This is achievable through the media streaming encoding technology (e.g., MDC [46] and layered encoding [47, 48]). Besides, the overlay topology should guarantee a path from the source to the peer with sufficient bandwidth. Second, the limited uplink bandwidth should be utilized wisely so that the total downloading rate is maximized.

In this chapter, we investigate the heterogeneity problem in peer-to-peer media streaming system. We present our solution in two steps. First, we propose a topology construction scheme to optimize the overlay topology construction for peer-to-peer streaming systems with heterogeneous downloading requirements. Although the scheme is designed for live peer-to-peer media streaming systems, the result can also be applied to peer-to-peer content delivery or file downloading systems. Second, we apply network coding to heterogeneous peer-to-peer media streaming systems based on the proposed topology construction scheme.

4.1 Optimal Overlay Topology Construction for Heterogenous Peer-to-Peer Streaming Systems

As we mentioned earlier, the overlay topology for peer-to-peer streaming systems can be divided into two types: tree-based and mesh-based. Tree-based peer-to-peer streaming systems build one or more trees for the media content delivery transmission. The direction and the transmission rate of the media content are fixed by the trees. Mesh-based peer-to-peer streaming system build a loosely connected mesh in a sense that peers can switch from one neighbor to another any time as they want. The direction and the transmission rate of the media content delivery depend on the availability and current residual bandwidth. One thing needs to be clarified here is that a scheme constructing a mesh overlay topology is still a tree-based scheme if the mesh is formed by a union of multiple trees. In other words, the division of whether a scheme is tree-based or mesh-based is dependent on the media content transmission paths not on the resulting topology. Overall, tree-based approaches provide more stable transmission paths while mesh-based approaches provide a more flexible topology. For a heterogeneous peer-to-peer streaming system with limited uplink bandwidth, we believe the tree-based approach is more suitable than the mesh-based approach. Therefore, we will mainly discuss several typical tree-based peer-to-peer streaming systems and introduce one mesh-based peer-to-peer

streaming system for completeness in this section.

DONet (Data-driven Overlay Network for live media streaming) [11] is a mesh-based data-centric peer-to-peer media streaming system in which the media content is transmitted on-demand. In DONet, peers are connected by randomly selected links among them and adjacent peers are called partners. Each peer maintains a list of partners and a buffer map which indicates the media content the peer contains. Peers continuously exchange the buffer map with their partners. The key operation in DONet is that every peer periodically exchanges buffer map with its partners and retrieves the missing media content from its partners. A heuristic scheduling algorithm is designed to determine the order and the supply partners of the media content retrieval.

Next, we introduce two tree-based approaches which arrange the peers into a hierarchical structure that implicitly defines the topology of the tree.

Developed by University of Maryland, NICE [40] is a project of designing a cooperative framework for scalably implementing distributed applications including peer-to-peer streaming systems over the Internet. The NICE protocol organizes peers into a cluster hierarchy where each layer is composed of one or more clusters of peers. For each cluster, a cluster leader is selected among the cluster members. Each layer is composed of the peers which are the cluster leaders in the lower layer. The size of a cluster is between k and $3k - 1$, where k is a constant and set to 3 in the paper. The peers in a cluster are close to each other and the peer at the cluster center is chosen as the cluster leader. The bottom layer contains all the peers while the top layer contains only one peer. The cluster hierarchy implicitly defines the media content delivery paths. When designing a peer-to-peer streaming system over NICE, the server is the top peer. A multicast tree is built for media content distribution in such a way that the root is the server and the parent of a peer is its cluster leader.

One problem in the NICE protocol is that the cluster leader is responsible for forwarding the media content to all the clusters it belongs to, so that in the worst case, the cluster leader in the top layer forwards the media content to $O(k \log_k N)$ other peers, where N is the number of peers in the system. ZIGZAG [41] is another hierarchical peer-to-peer system which limits the out-degree of a peer by an upper bound. ZIGZAG organizes the peers into a multi-layer hierarchical cluster which is similar to the NICE protocol. The difference is that ZIGZAG introduces the concept of *associate head* in the cluster hierarchy. Each cluster has a cluster head (similar to the cluster leader in the NICE protocol) and a cluster associate head. The size of a cluster is between k and $3k$, where k is a constant and $k > 3$. ZIGZAG tries to avoid the bottleneck in the NICE protocol by limiting the out-degree of the peers in the distribution tree. The cluster associate head, instead of the cluster head, is responsible for forwarding the media content to the cluster it belongs and obtaining

the media content from one peer in the upper layer. As a result, the degree of the distribution tree in ZIGZAG is bounded by $6k - 3$.

Both of the above schemes construct a single tree for the media content delivery which causes two problems: First, some peers, for example, the cluster leader in NICE or the cluster head in ZIGZAG, are more important than other peers. Thus it suffers point of failure problem; Second, the uplink bandwidth of the leaf peers is wasted. Next, we discuss two schemes that employ multiple trees to alleviate these problems.

SplitStream [29] is proposed to overcome the unbalanced forwarding load in conventional tree-based approaches and the traffic stoppage in peer failure or sudden departure. The key idea in SplitStream is to split the media content into n stripes and to multicast each stripe using a separate tree. Peers join different trees to receive different stripes respectively. The goal of SplitStream is to construct a forest of trees such that each peer is an interior node in one tree and a leaf peer in all the remaining trees while minimizing the delay and link stress across the system.

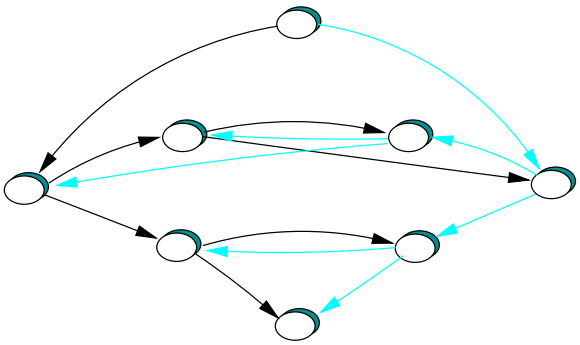


Figure 4.1: Illustration of multiple distribution trees in SplitStream.

As Fig. 4.1 shows, two trees are constructed to span the same set of peers. Each peer, except the root, receives two stripes and forwards a strip twice. In this way, the forwarding load is distributed among the peers evenly and a peer failure will cause the data loss of at most one stripe at the downstream peers. When coupled with some media encoding scheme, peers can always reconstruct the media content with the quality proportional to the number of the stripes it receives. SplitStream is similar to the scheme proposed in this section. But there are several differences. SplitStream treats every peer equally without considering the heterogeneous access link bandwidths. It focuses more on distributing forwarding load evenly around all the peers. While the proposed scheme focuses on maximizing the satisfaction of peers by treating peers differently with respect to their access link bandwidths.

More recently, a scheme to minimize the end-to-end delay in the overlay topology of a peer-to-peer streaming system is proposed in [49]. It assumes that peers

have heterogeneous uplink bandwidths and uniform downloading rates. It first formalizes the problem into Minimum Delay Mesh problem (We will use MDM to represent the scheme in the rest of the chapter). Then it proposes a power-based distributed algorithm. *Power* is defined between a child and its parent which is the ratio between the parent's residual uplink bandwidth and the delay from the child to the root through the parent. The parent with higher power is preferred since it offers higher bandwidth or less delay. When a new peer joins the system, it sorts the existing peers in a descending order of their powers and chooses the peers as its parents from the beginning until its downloading rate requirement has been satisfied.

Our approach proposed in this section is also based on the idea of adopting multiple trees for constructing overlay topology for peer-to-peer streaming system. However, there are two critical differences distinguishing our approach from the existing work. First, our approach does not require the tree to span all the peers as required in SplitStream. Second, our approach does not require the downloading rates of the peers to be uniform as required in MDM. Thus, our approach is much more general than the previous schemes. The basic idea of our approach is to model the peer-to-peer streaming system by a graph G on which we find maximum edge disjoint trees as many as possible. As a result, our approach can be used for peer-to-peer streaming systems where the downloading rate requirement is heterogeneous and the uplink bandwidth of the peers is limited.

The main contributions of this section can be summarized as follows.

- Formalize the problem of optimal overlay construction for peer-to-peer streaming system with heterogeneous downloading requirements. To the best of our knowledge, this is the first work formalizing the problem. We show the hardness of the problem through the comparison with a known NP hard problem and present a greedy heuristic algorithm to solve it.
- Propose a distributed algorithm to construct the overlay topology under dynamic peer joining and leaving. The overlay topology can adapt itself to the changing peers in the system.
- Evaluate the performance of proposed algorithms and conduct comparison with other approaches through simulations. Simulation results show that our algorithm achieves 30% higher peer satisfaction and less link stress.

4.1.1 Problem Formalization

In a typical peer-to-peer streaming system, a streaming server hosts the media content and all the peers requesting the media content retrieve the streaming data from the server directly or from other peers. As peers can be any computers connected to

the network, different peers have different access link bandwidths, which is called link heterogeneity. Link heterogeneity is a common phenomenon in today's Internet. Any practical peer-to-peer streaming system should take link heterogeneity into consideration. In this section, we consider the system where each peer has asymmetric access links, which is common for the ISP providers nowadays. Each peer downloads the media content through *downlink* at a speed upper bounded by the downlink bandwidth (BW_{down}), and uploads the media content through *uplink* at a speed upper bounded by the uplink bandwidth (BW_{up}). The playback quality of a peer is determined by the downloading rate of the peer (here we assume that the peer always prefers to download the media content at a higher bit rate if the downlink bandwidth permits. In the case that the downlink bandwidth is higher than the required bit rate, we change the downlink bandwidth to the required bit rate). We adopt the network model where all the peers are connected to a high speed network core with their access links. The bandwidth bottleneck only lies on the network edge, while the network core has sufficient bandwidth to transmit data among different peers simultaneously. We formulate the network model into a graph $G = (V, E)$ as shown in Fig. 4.2. For each peer, there are two artificial nodes which are used to model the downlink and uplink respectively. The link between the ingress artificial node is the downlink with bandwidth BW_{down}. The link between the outgress artificial node is the uplink with bandwidth BW_{up}. The source node representing the server has only an outgress node and the bandwidth of the link between the source node and its outgress node equal to the uplink bandwidth of the server. Each outgress node is connected to all the ingress nodes with sufficiently large bandwidth except the ingress node corresponding to the same peer as the outgress node.

As different peers have different downlink bandwidths, it is preferred that each peer is able to download the media content at its maximum downloading rate, i.e. BW_{down}. In peer-to-peer streaming systems, peers download the media content not only from the server, but also from other peers. A mesh overlay is constructed to represent the data flow among peers. To achieve the best playback quality and utilization of access link bandwidth, the overlay topology must be constructed carefully to meet the streaming quality requirement and avoid bandwidth wasting.

We first discuss some issues that need to be considered in the overlay topology construction. First, the uplink bandwidth of the server is the upper bound of the downloading rate of all the peers. This is clear because a peer cannot download the media content at a speed higher than the server can provide. Therefore, if the downlink bandwidth of a peer is greater than the uplink bandwidth of the server, its downlink bandwidth cannot be fully utilized. In the network model graph G, we change the downlink bandwidth which is greater than the upper bound to the upper bound. This change will not alter the resulting overlay topology. Another necessary

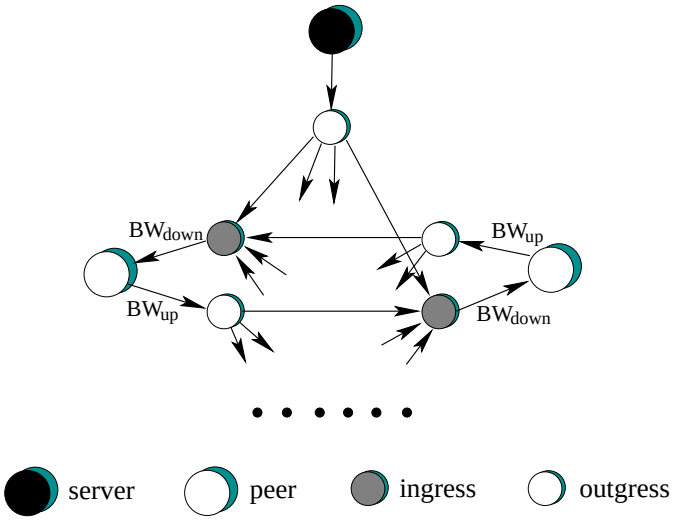


Figure 4.2: Illustration of the network model for heterogeneous peer-to-peer streaming systems.

condition for all the downlink bandwidth to be fully utilized is that the total uplink bandwidth is greater than the total downlink bandwidth. This is a necessary but not a sufficient condition. Its insufficiency can be explained by the following example as shown in Fig. 4.3.

Suppose there are a total of $n+1$ peers in the system as shown in Fig. 4.3(a). The uplink bandwidth of the server is 2. Peer 1 has a downlink bandwidth of 1 and an uplink bandwidth of $2n-1$. The remaining n peers have downlink bandwidth of 2 and zero uplink bandwidth. Therefore, the total uplink bandwidth is $2n+1$, and the total downlink bandwidth is also $2n+1$. Note that although the total downlink bandwidth of the n peers is the same as the total uplink bandwidth of the server and the peers, it is impossible to construct an overlay topology which can transmit the media content to all the peers at their maximum downloading rates. In the best case as shown in Fig. 4.3(b), only two peers (peer 1 and peer 2) can download the media content at their maximum downloading rates, and all the remaining peers can only download the media content at half of its maximum downloading rate.

We can see that the downlink bandwidth cannot be fully utilized even if there is enough uplink bandwidth. Our goal is to maximize the utilization of downlink bandwidth given the network model graph G. The overlay topology plays a critical role in the utilization of downlink bandwidth. In the previous example, if the server dedicates all its uplink bandwidth to a peer with a downlink bandwidth of 2, then all the peers except that peer cannot receive any media content through the server or other peers. Given a network model graph G and an overlay topology, the down-

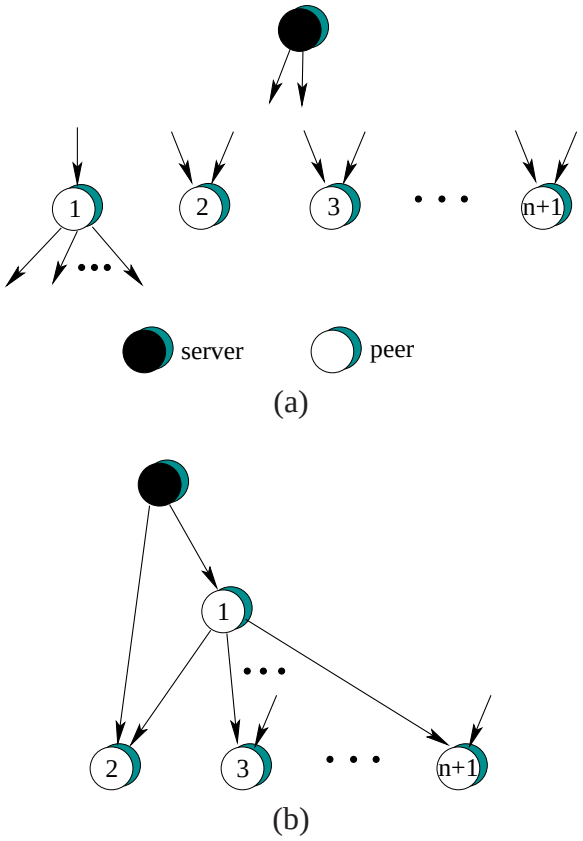


Figure 4.3: Illustration of the necessary but insufficient condition for overlay topology for peer-to-peer streaming systems. (a) The bandwidth configuration of the server and the peers. (b) The overlay topology to achieve the maximum total downloading rate.

loading rate of a peer is equal to the minimum cut between the source node and the peer. Here the overlay topology includes not only the links between peers, but also the bandwidth allocation on the links.

According to the above discussion, the problem of maximizing the utilization of downlink bandwidth can be formalized as follows: Given a peer-to-peer streaming system G, construct an overlay topology such that the sum of the minimum cuts between the source and each peer is maximized.

The problem formalization does not provide a clue in solving the problem. We notice that although the overlay topology is a mesh, the media content delivery is actually through multiple trees. If we divide the stream into multiple unit substreams such that one unit substream occupies one unit of bandwidth, the overlay topology can be decomposed into multiple trees each of which represents a substream. Then the downloading rate of a peer is equal to the number of trees connected to the peer.

If we define the length of a tree as the number of its nodes minus one, the total downloading rate of a substream is equal to the length of the tree. Therefore, the problem can be transferred to:

Given G, find a set of edge disjoint trees rooted at the source node such that the sum of the length of the trees is maximum.

We call it Maximum Downloading Rate problem (MDR). To show the hardness of the problem, we first take a look at the steiner tree packing problem.

Steiner tree packing problem: Given a graph G, find the maximum number of edge disjoint subgraphs that connect a given set of nodes.

It has been shown that steiner tree packing problem is an NP hard problem [32]. The difference between the MDR problem and the steiner tree packing problem is that steiner tree requires each tree spans all the nodes in the given set, while in MDR problem, the trees are allowed to span part of the peers as long as the total tree length is maximum. In other words, the steiner tree packing problem is a special case of the MDR problem. Therefore, the hardness of the MDR problem is no less than that of the steiner packing tree problem, thus MDR is NP hard.

4.1.2 The Greedy Heuristic Algorithm

Due to the NP hardness of the MDR problem, as a starting point, in this section we propose a greedy heuristic algorithm to find the maximum number of edge disjoint trees given the network topology G. It is a centralized algorithm which requires the complete information on the peers and their access link bandwidth. As we know, in practice, this information is dynamic and usually cannot be obtained in advance. We will further propose a distributed algorithm to handle the dynamics of peers in the next section. Nonetheless, the centralized heuristic algorithm serves as the foundation for the distributed algorithm and the benchmark when we evaluate the performance of the distributed algorithm in Section 4.1.4.

The basic idea of the greedy heuristic algorithm is to pick out a maximum edge disjoint tree from graph G one by one until there is no such a tree. Before we apply the algorithm to graph G, we need to modify the graph by replacing each link of bandwidth b with b parallel links each of which has bandwidth 1. This modification will not change the result of the algorithm. Therefore, each tree represents a substream with bit rate of 1. The receiving bit rate of a peer is equal to the number of trees the peer is on. Each tree must be rooted at the source node. To ensure the server transmits as many substreams as possible, the number of children of the root is limited to one for each tree. Two factors are taken into consideration for the tree construction. First, the height of the tree should be minimized. The height of the tree determines the delay between the source and the peer. To minimize the height of the tree, we should push the peers with higher uplink bandwidths to the source

node as close as possible. Second, fairness bandwidth allocation between different trees should be maximized. When a peer receives the media content from several different trees, the delays to the source along the trees may be different. The final end-to-end delay from the source to the peer is determined by the worst case, i.e. the longest delay among all the delays. One way to minimize the difference among these delays is to minimize the difference of the tree heights, as the height of the tree represents the longest end-to-end delay approximately. Fairness here means that the uplink bandwidth should be shared among different trees in a fair manner. As a result, the height difference among different trees is minimized. The fairness is realized by a property of the peer called *fanout*. Fanout is a value which is used to evaluate the forwarding ability of a peer. The fanout of a peer is equal to (uplink bandwidth)/(downlink bandwidth). The larger the fanout, the more children the peer can have. We use fanout as the upper bound on the number of children a peer can have in one tree. The reason is two folds. On one hand, if the number of children is smaller than the fanout, some uplink bandwidth will be wasted definitely even if the downlink bandwidth is filled up with the substreams. On the other hand, if the number of children is more than the fanout for one tree, it means that the number of children is less than the fanout for another tree. This imbalance of uplink bandwidth allocation contradicts the fairness principle.

Table 4.1 lists the pseudo-code of the greedy heuristic algorithm, where the *maximum_tree* sub-procedure is used to find a maximum tree in the residual graph G'.

4.1.3 The Distributed Algorithm

The greedy heuristic algorithm works well if the information of the peers and their access link bandwidths are given in advance. However, in practice, peers join or leave the peer-to-peer system frequently which is known as *churn*. Due to the frequent churn rate, it is impossible for the server to run the greedy heuristic algorithm every time when there is a peer joining or leaving. When a new peer joins the system, it should be grafted to the system in an efficient and distributed fashion. The topology mismatch is another issue when we design a practical peer-to-peer streaming system. Here mismatch means that two nodes close to each other in overlay topology are far away in physical network topology or vice versa. The topology mismatch should be minimized to reduce network bandwidth consumption and end-to-end delay. As peers join the system in a random order, the mismatch between the overlay topology and the physical network topology may become larger and larger. A practical peer-to-peer streaming system should be able to adapt itself to the changing overlay topology by alleviating the mismatch. In this section, we propose a distributed algorithm which can handle the frequent peer join/leave in a

Table 4.1: Greedy Heuristic Algorithm

Greedy heuristic algorithm
Input: graph G.
Output: a set of maximum trees.
Begin:
  Foreach peer v in G
    $fanout(v)$ =
        uplink bandwidth/downlink bandwidth;
  Put peers in list l in a descending order of fanout;
  G' = G;
  T = maximum_tree(G', l);
  While T != NULL
    output = output+T;
    G' = G'-T;
    l = l-peers with no residual downlink bandwidth;
    T = maximum_tree(G', l);
End

Maximum_tree(G**,** l**)**
Input: graph G, list l.
Output: a maximum tree T.
Begin:
  v = the first peer in l;
  $pointer$ = the second peer in l;
  While $pointer$ has not reached the end of list l
    Do
      If $pointer$ has residual downlink bandwidth
        Connect v with peer $pointer$;
        $pointer = pointer + 1$;
    Until peer v is connected to $fanout(v)$ peers
          or $pointer$ reaches the end of list l;
    $v = v + 1$;
End

distributed fashion and adjust the topology to alleviate the mismatch.

Peer Joining

We assume that the server is well-known whose IP address is known to all the peers by some address translation service such as DNS. The server maintains a partial list of existing peers in the system and their IP addresses.

The order of the peers in the list is determined by two factors: fanout and end-to-end delay. As we discussed earlier, fanout represents a peer's forwarding ability. A peer with a larger fanout can have more children, so we need to move the peers with larger fanout values to the server as close as possible in the tree. Similarly, the end-to-end delay between a peer and the server represents the distance between the peer and the server. To reduce the mismatch, it is reasonable to move the peers with shorter end-to-end delay closer to the server. We use a tunable parameter α to control the weights of these two factors. We let $level = \alpha * fanout + (1 - \alpha)/$end-to-end delay, where $level$ is a variable suggesting the position of the peer in the trees approximately. The larger the value of $level$, the closer the peer should be put to the server in the tree. The partial list maintained in the server is in an ascending order of $level$ to facilitate the join procedure. We will evaluate the impact of the tunable parameter on the system performance in Section 4.1.4.

When a peer wants to receive the media content, it initiates a join process by measuring the delay between the server and itself. Then it sends a JOIN request with the calculated value of $level$ to the server. The server will respond with a list of peers which are picked in the partial list. The picked peers have similar $level$ values as the joining peer. Compared to the schemes in which the server responds with a random list of peers, the topology constructed by our scheme will converge to the optimal topology much faster, as the new peer is already put at a near-optimum position during the join procedure and less topology adjustment is needed afterwards. The new peer will establish connections with the peers in the list in a descending order of their values of $level$ until its downlink bandwidth is filled up.

It should be pointed out although the server is responsible for bootstrapping the peers, it will not be the bottleneck of the system, because once each peer receives the list of peers, it communicates directly with the peers for overlay topology construction and media content dissemination.

Peer Leaving

We use simple peer leaving process to accommodate resilience and alleviate the impact of churn. When a peer leaves the system, we do not reconfigure the topology

which will cause a lot control overhead and sometimes even streaming stoppage. Instead, we let the disconnected peers to simply rejoin the system.

There are two types of peer leaving: friendly or abruptly. Friendly leaving means that the leaving peer initiates a leaving process so that the system is aware of its leaving and can make necessary updates accordingly. Abruptly leaving means that the leaving peer leaves the system without any notification, mainly due to link crash or computer crash.

For the friendly leaving, the leaving peer will initiate a leaving process by sending LEAVE messages to both of the server and its neighbors (parents and children). The leaving of the peer will cause the peers which are its descendants in the trees to disconnect from the trees and lose some substreams. To reconnect these peers to the trees, the children of the leaving peer will initiate the join procedure like a new peer. For the parents of the leaving peer, they will free the uplink bandwidth used by the leaving peer for the future use. Here the server acts as a connection "hub" for the peers that are connected to the leaving peer. This may increase the processing burden on the server temporarily. Nevertheless, it can achieve strong robustness with little control overhead. For example, it can handle concurrent peer leavings. When the server receives the LEAVE message, it will check whether the peer is on the partial list and remove it from the list if it is on the list.

For the abruptly leaving, peers send HELLO messages to its neighbors periodically and maintain a HELLO timer for each neighbor. Receiving a HELLO message triggers a reset of the corresponding HELLO timer. The neighbors detect the abruptly leaving by the timeout of the HELLO timer. After detecting the leaving of the peer, the parents and the children will perform similar operations to that in the friendly leaving. Besides, the parents will send LEAVE messages to the server on behalf of the leaving peer so that the server can update the list.

Topology Adjustment

Peers join and leave the system in a random manner. According to the join procedure, a new peer is always a descendant of the existing peers after it joins the system. As a consequence, the overlay topology is changing as the peers join and leave and is dependent on the order of the peers joining and leaving. Due to the randomness of the order of peer joining and leaving, the overlay topology may be far from the one constructed by the greedy heuristic algorithm. We propose a topology adjustment procedure to help the peer-to-peer system handle the dynamic peer joining and leaving.

The intuition behind the adjustment procedure is to locate the peers whose positions do not match their $level$ values and move these peers to a proper position. It is composed of three steps. (1) Step 1. The peer (We use v to denote the peer in the rest of this section) sends a REQUEST message upstream along the tree towards the

root. The REQUEST message includes the value of $level$ of v and a hop counter. Each peer receiving the REQUEST message will forward it to its parent and add 1 to the hop counter. The REQUEST message will stop when it reaches the root. (2) Step 2. Each peer receiving the REQUEST will compare the value of $level$ in the message with its own value of $level$. If its own $level$ is less than that in the message, it will send a GRANT message back to v. The GRANT message contains its own value of $level$, its parent, the residual uplink bandwidth of its parent and the value of the hop counter. (3) Step 3. If v does not receive any GRANT message, it does nothing. If v receives one or more GRANT messages, it will choose one peer to be its new parent based on the following rules: sort the peers in a descending order of their hop counts; select the first peer whose parent has non-zero residual uplink bandwidth and take its parent as the new parent of v. The peer v and the subtree rooted at v will become a subtree of the new parent. If there is no such a peer whose parent has non-zero residual uplink bandwidth, the peer checks itself to see if there is any residual uplink bandwidth. If no, the peer does nothing. If yes, the peer v will select the peer with the largest hop count and take its parent as the new parent. As there is no residual uplink bandwidth of the parent, one child of the parent will be replaced by v. The old child will become a child of v. The rationale behind the rules is to move the peer as close to the root as possible while minimizing the disturbance to the existing tree structure.

4.1.4 Performance Evaluation

In this section, we study the performance of the proposed algorithms through simulations. We have implemented the proposed scheme in NS-2 [33]. The network topologies used in the simulations are random transit-stub network topologies generated by GT-ITM software [27]. Peers are selected randomly from the stub networks and the bandwidth of the links in the transit network is set to be sufficiently high (1000 Mbps in the simulations).

We compare our scheme with the MDM algorithm proposed in [49], as it is the closest work to ours in the sense that both of the algorithms try to optimize the overlay topology for peer-to-peer media streaming systems. The main difference is that our scheme considers heterogeneous downloading rates while MDM assumes a uniform downloading rate.

The simulation adopts following three performance metrics:

Satisfaction. Even though the overlay topology is carefully constructed, it is inevitable that a peer may not receive the media content at its preferred bit rate. We use *satisfaction* to evaluate the extent the peer is satisfied with its received media streaming bit rate. A peer's satisfaction is defined as the ratio of the received media streaming bit rate to its downlink bandwidth. In the simulation, we use the average

satisfaction of all peers to represent the satisfaction of the system.

End-to-end delay. This performance metric is used to evaluate the end-to-end delay between the source and the peers. The end-to-end delay is measured along the overlay links towards the source instead of the physical links. Since there are usually multiple overlay paths towards the source, we use the delay of the longest path as the end-to-end delay of the peer. Again, we average the end-to-end delay of all peers in the simulations. Since the time is the virtual simulation time in NS2, the time unit is virtual as well.

Link stress. Link stress is defined as the number of copies of the same message transmitted through the same link. It is a performance metric that only applies to an overlay network due to the mismatch between the overlay network and the physical network. We use it to evaluate the effectiveness of the topology adjustment and the efficiency of the system.

Satisfaction

We first compare the average satisfaction of peers under different sizes of the peer-to-peer streaming system. We use the number of peers in the system to represent the size of the system.

Fig. 4.4 shows the average satisfaction as we change the system size. We can observe that the greedy heuristic algorithm demonstrates the best satisfaction and the distributed algorithm outperforms MDM by about 30% on the average. It suggests that our scheme can make more efficient use of the uplink bandwidth of peers than MDM. With the increase of the system size, the average satisfaction drops for all the three schemes. This is due to the fact that when the trees become large, the wasted uplink bandwidth of the leaf peers increases as well even though multiple trees are employed. From the trend of the curves, we can see that our scheme is more stable then MDM with the increase of system size. The decrease of the satisfaction of the distributed algorithm is about 26% while that of MDM is about 45%.

To examine the system performance under different access link bandwidth constraints, we let the ratio between the total uplink bandwidth and the total downlink bandwidth equal to 4, 2 and 1, respectively. Fig. 4.5(a) shows the simulation results. We can see that the average satisfaction of the distributed algorithm is always higher than that of MDM. Moreover, the average satisfaction of the distributed algorithm when the total uplink bandwidth equals to the total downlink bandwidth is higher than that of MDM when the total uplink bandwidth is 2 times of the total downlink bandwidth. It indicates that the proposed scheme performs well especially when the total uplink bandwidth is limited.

The tunable parameter α is another factor that affects the system performance. Fig. 4.5(b) shows the average satisfaction under different values of α and differ-

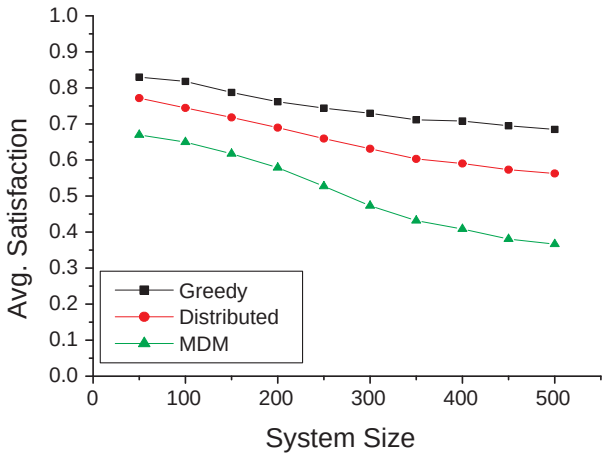


Figure 4.4: Average satisfaction under different system sizes.

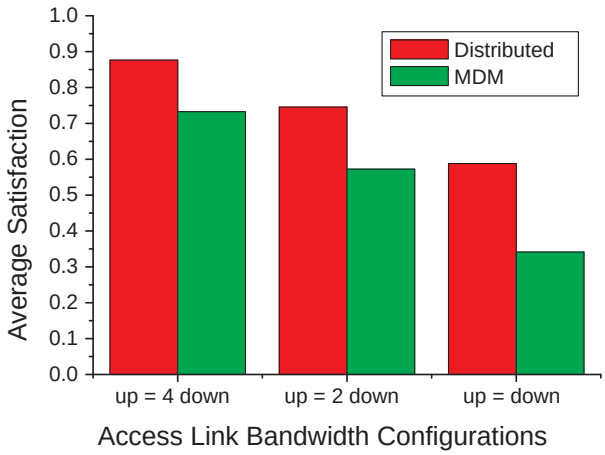
ent access link bandwidth configurations. We can see that the average satisfaction increases with the increase of α. When α is small, end-to-end delay enjoys more weight in the overlay topology construction. The resulting topology tends to have a shorter end-to-end delay at the expense of the satisfaction. The slope, i.e., the increasing rate of the average satisfaction, is greater when α is small than that when α is large. This can be used as a system design guide to find an optimal α value for both satisfaction and end-to-end delay.

End-to-end Delay

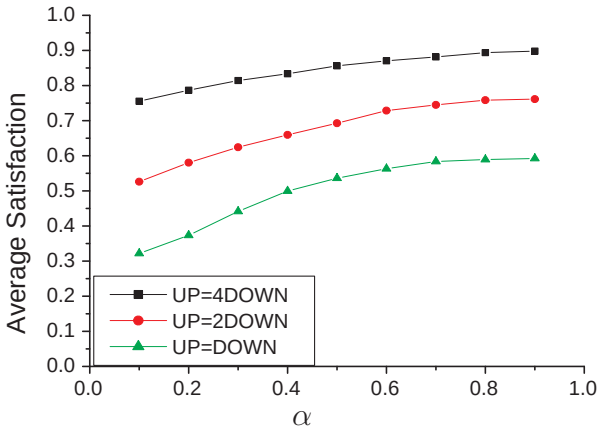
The end-to-end delay is an important performance metric for peer-to-peer media streaming systems. As we can see later, sometimes end-to-end delay and satisfaction are two conflicting performance metrics and the best solution is a tradeoff between these two metrics.

Fig. 4.6 shows the end-to-end delay under different system sizes. We can see that with the increase of the system size, the end-to-end delay increases as well due to the large tree heights. When the system size is small, the greedy heuristic algorithm achieves the shortest end-to-end delay. As the system size becomes large, the end-to-end delay of MDM is shorter than that of other two schemes. The reason is that MDM is a scheme focused on minimizing the end-to-end delay. The advantage of MDM is more obvious when the system size is large.

Now we investigate the impact of the parameter α on the end-to-end delay. As shown in Fig. 4.7, the end-to-end delay increases with the increase of α. The ra-



(a)



(b)

Figure 4.5: Average satisfaction evaluation. (a) Average satisfaction under different access link bandwidth configurations; (b) Average satisfaction under different values of α.

tionale behind this is similar to that of the results for satisfaction. As we put more weight on $fanout$ when we construct the overlay topology, the resulting end-to-end delay becomes longer. When the uplink bandwidth constraint is tight, the increase of the end-to-end delay is faster compared to that when the uplink bandwidth constraint is loose. This attributes to the larger tree height due to the limited uplink bandwidth when the uplink bandwidth constraint is tight.

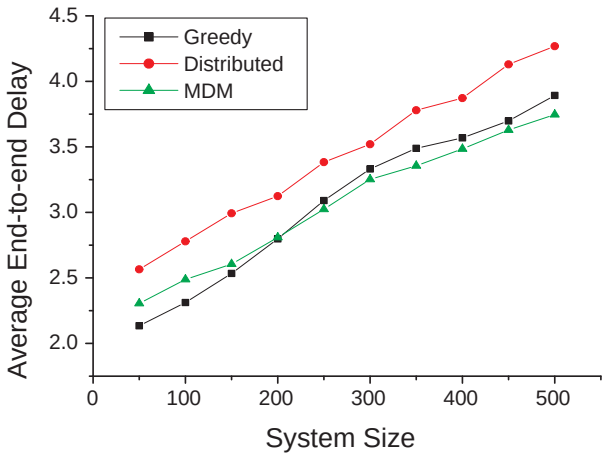


Figure 4.6: Average end-to-end delay under different system sizes.

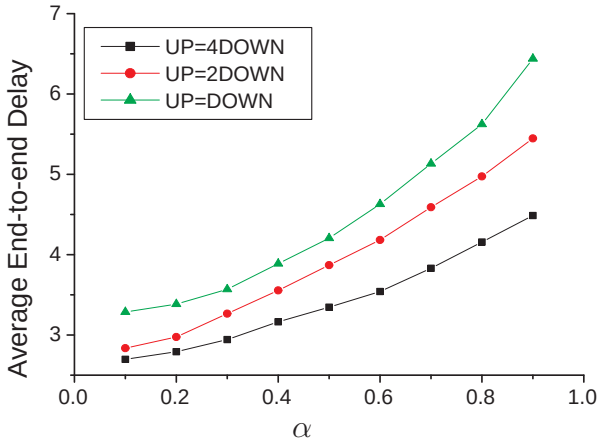


Figure 4.7: Average end-to-end delay under different values of α.

Link Stress

Link stress is an indicator of the efficiency of the overlay topology. Higher link stress will cause higher delay and bandwidth wasting.

We first look at the link stress under different system sizes as Fig. 4.8 shows. With the increase of the system size, the link stress increases as well. The greedy

heuristic algorithm achieves the least link stress thanks to the complete peer information before the overlay topology construction. The link stress of MDM is slightly higher than that of the distributed algorithm. This is because that although the proposed distributed algorithm considers heterogeneous download rates while MDM only considers a uniform downloading rate, many trees constructed in the distributed algorithm have smaller heights as they do not span all the peers. As a result, the probability that a packet passes the same physical link is reduced and the link stress is reduced as well.

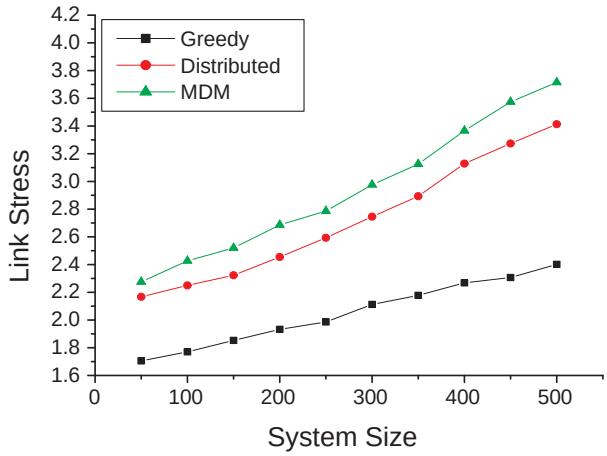


Figure 4.8: Link stress under different system sizes.

Fig. 4.9 shows the link stress when we tune the parameter α. The impact of parameter α on link stress is notable. With the increase of α, link stress increases remarkably. As mentioned earlier, a larger α means shorter end-to-end delay. While the end-to-end delay is a good approximation of physical distance between two peers. When the end-to-end delay is reduced, the mismatch between the overlay network and physical network is reduced as well. The impact of the uplink bandwidth constraint on link stress is small. When the uplink bandwidth constraint is loose, the total download bandwidth is increased. However, this increase will not cause the increase of the link stress, because it becomes more likely to find a peer nearby to exchange the media content with.

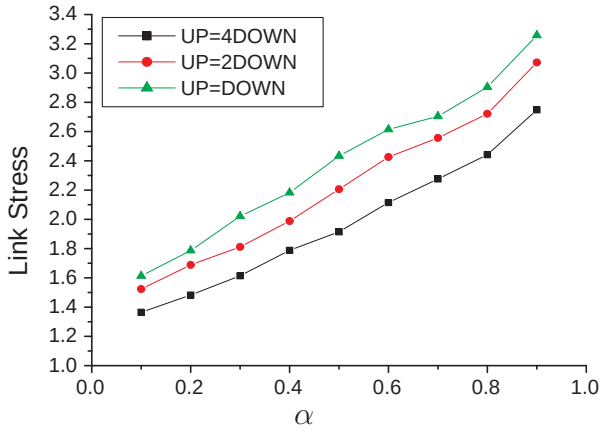


Figure 4.9: Link stress under different values of α.

4.2 Adaptive Network Coding for Peer-to-Peer Media Streaming Systems

Like peer-to-peer file sharing systems, peer-to-peer streaming systems are a perfect place to apply network coding. The paper in [50] gives a theoretical analysis on how network coding can improve the performance of peer-to-peer streaming systems. A random network coding scheme was proposed for peer-to-peer media streaming systems in [51]. The scheme focuses on utilizing random network coding to simplify the scheduling algorithm for peers to exchange the media content. However, it does not consider peers' heterogeneity in peer-to-peer streaming systems. A network coding scheme based on layered encoding was considered in [52] and [21] for media streaming systems. The scheme proposed in [52] was designed for media streaming multicast networks while the scheme proposed in [21] was designed for peer-to-peer streaming systems. Layered Coding (LC) [47] is a multimedia encoding technique which encodes the media content into a base layer and multiple ordered enhancement layers. The receiver can reconstruct the media content if it receives the base layer and a subset of the enhancement layers. It is a cumulative encoding technique in the sense that an enhancement layer can be used for decoding only if the base layer and all the enhancement layers before it are received by the receiver. The quality of reconstructed media content is proportional to the layers used. In [52], the receivers are divided into several groups based on their max-flow values to the source. Then a subgraph is constructed for each group of receivers. Given a subgraph, a deterministic linear network coding scheme is

determined by the algorithm proposed in [16]. In [21], the authors used the overlay construction algorithm proposed in [12] to construct a basic overlay network. Then layered meshes are constructed for layered media content. Each mesh is responsible for one layer of the media content. Peer join different meshes to receive different layers of the media content.

In this section, we propose an adaptive network coding scheme which can optimize the bandwidth utilization in a heterogeneous peer-to-peer media streaming system. The main differences between the existing schemes and our scheme are: First, we use Multiple Description Coding (MDC) to encode the media content instead of LC. MDC was first proposed to enhance the robustness of multimedia data over unstable channels. The basic idea of MDC is to fragment a single media stream into multiple independent sub-streams referred to as descriptions. In order to decode the media stream, any description can be used. However, the quality improves with the increase of the number of descriptions received. Here we use MDC to encode the media content into multiple stripes which are equally important when used to reconstruct the media content. The quality of the reconstructed media content is proportional to the number of stripes used. This leads to a great advantage over LC in the sense that the peer can always reconstruct the media content as long as it receives one or more stripes. While in LC, if the base layer is missing, the media streaming has to stop. If an enhancement layer is missing, all the enhancement layers after it are of no use which causes bandwidth wasting. Second, since most peers are individual computers which are connected to the Internet through access links, peers may have different bandwidth in practice. Our proposed scheme aims at such practical networks by considering asymmetric access links where uplink bandwidth and downlink bandwidth are bounded by a link capacity. This is common for the ISP providers nowadays. Third, we use the overlay construction algorithm described in the previous section to construct the overlay network. It is optimized for heterogeneous peer-to-peer media streaming systems. While other schemes usually adopt an existing general purpose overlay topology construction algorithm. For example, the scheme in [21] used the algorithm proposed in [12].

Our scheme first encodes the media content into multiple stripes using MDC (Multiple Description Coding) technology [46]. Then peers subscribe to a subset of these stripes based on their downlink bandwidths. Network coding is performed within one stripe. The playback quality of a peer is proportional to the number of stripes it subscribes to. The overlay topology construction is also tailored to optimize the bandwidth utilization of the peers' access links. Although the scheme is designed for live peer-to-peer media streaming systems, the result can also be applied to peer-to-peer content delivery or file downloading systems. The main contributions can be summarized as follows.

- Formalize the problem into a mathematical optimization problem.

- Propose an overlay topology construction algorithm which is tailored to heterogeneous peer-to-peer media streaming systems. The overlay topology can maximize the bandwidth utilization of the access links.

- Propose an adaptive network coding scheme for heterogeneous peer-to-peer media streaming systems. Our scheme can satisfy the heterogeneous download requirements by encoding the media content into multiple stripes. Meanwhile, it achieves good scalability and resilience.

- Evaluate the performance of the proposed scheme and conduct comparison with other approaches through simulations. Simulation results show that our algorithm achieves higher peer satisfaction and throughput.

4.2.1 Problem Formalization

Without loss of generality, the topology of a peer-to-peer streaming system can be modeled as a multicast network which can be represented by a directed graph $G = (V, E, C)$ where V is the set of network nodes and E is the set of links each of which connects two nodes. Each link can be represented by an ordered node pair (v_1, v_2) where $v_1, v_2 \in V$. v_2 is called the head of the link and v_1 is called the tail of the link. The messages can only be transmitted from v_1 to v_2. C is a real non-negative function $C : E \rightarrow R^+$ which maps each link e to a real non-negative number $C(e)$ which is the transmission capacity of the link. The media content is generated at a source node $s, s \in V$ and flows to a set of receivers $R, R \subseteq V$.

With the help of MDC, the media content is encoded into multiple stripes at the source node before being sent out. In our model, we encode the media content into k stripes each of which has the same bit rate b. The receiver can reconstruct the media content with any subset of the k stripes. The playback quality of a receiver is proportional to the number of stripes it receives. Without network coding, MDC is realized by finding multiple disjoint multicast trees spanning the source and the receivers. Each tree is responsible for one stripe. With network coding, the bandwidth can be further utilized by applying network coding to the flows in the same stripe. It is possible to encode flows from different stripes. However, it requires decoding in the relay nodes in addition to the receivers. Therefore, due to the complexity, in this section we do not consider network coding between different stripes.

Suppose each stripe is distributed to the receivers through a subgraph (Given a graph $G = (V, E, C)$, a subgraph G' can be defined as $G' = (V', E', C')$ where $V' \subset V, E' \subset E, C'(e') <= C(e')$). To apply network coding to the flows within one stripe, the subgraph must be a mesh instead of a tree. This implies that we should divide the stripe to multiple flows and transmit these flows along different paths to a receiver in order to increase the probability for nodes to perform network

coding. The receivers subscribe to one stripe if they want to receive the correspondent media content. The number of stripes a receiver subscribes to is upper bounded by the bandwidth of its downlink divided by b. By subscribing to different numbers of stripes, the utilization of heterogeneous downlink bandwidths can be maximized.

Our goal is to maximize the total receiving rate of all the receivers. Now the problem is transformed to how the receivers subscribe to the stripes such that the total receiving rate is maximized. Assuming that the subset of stripes receiver r subscribes to is $F(r)$, the problem can be formalized as a mathematical optimization problem as follows:

$$\text{maximize} \sum_{i=1}^{|R|} |F(r_i)| \tag{4.1}$$

subject to

$$\sum_{head(e)=v} x_i^j(e) - \sum_{tail(e)=v} x_i^j(e) = \sigma_i^j(v), \quad \forall\, v \in V, r_i \in R, j \in F(r), \tag{4.2}$$

$$\sum_j \phi^j(e) <= c(e), \tag{4.3}$$

where

$$\sigma_i^j(e) = \begin{cases} -b & \text{if } v = s \\ b & \text{if } v = r_i \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

$$\phi^j(e) = \text{Max}_i\{x_i^j(e)\} \tag{4.5}$$

In the above, $x_i^j(e)$ is the flow rate on link e for receiver r_i on stripe j. As each stripe has the same bit rate, to maximize the total receiving rate is equal to maximize the number of total stripes the receivers subscribe to. Equation (2) means that only the source node can generate flows and only the receivers can consume flows, while all the remaining nodes perform relaying. Equation (3) means that flows from different stripes can not share the bandwidth, and the summation of their bit rates can not exceed the link capacity. Equation (4) means that all the stripes have a constant bit rate b. Finally, Equation (5) means that the flows from the same stripe can share the bandwidth.

In addition to achieving the maximum throughput, we also want to maintain fairness among receivers. Fairness is defined as follows: the receiver with a larger max-flow value from the source node will receive no less stripes than the receiver with a smaller max-flow value. Then the optimization problem can be rewritten by adding one more constraint:

$|F(r_i)| \leq |F(r_j)|$ if max-flow(r_i)$\leq$ max-flow(r_j)

4.2.2 Adaptive Network Coding for Heterogeneous Peer-to-peer Media Streaming Systems

The solution to the above mathematical optimization problem requires centralized processing with all the topology and bandwidth information available. In a large scale distributed system such as the Internet, it is not scalable to deploy such a centralized algorithm. To accommodate scalability, it is necessary to develop a distributed algorithm implemented by network protocols. However, the problem formalization provides some insights and guidelines leading to the distributed solution.

In this section, we propose a distributed adaptive network coding construction scheme based on the discussion in the previous section.

Overlay Topology Construction

The first step is to construct an overlay network spanning the peers over which our adaptive network coding scheme can be applied. This can certainly be achieved by using some existing algorithms in the literature, such as those in [12, 53]. However, as most of existing algorithms are of general purposes, we will use the previously introduced overlay topology construction algorithm tailored for heterogeneous peer-to-peer media streaming systems for efficiency purpose.

We still use fanout to evaluate the forwarding ability of a peer. The topology construction heuristic is based on the observation that the probability that a peer with a higher fanout relays media content to a peer with lower fanout is higher than the reverse. Fanout can be roughly considered as the number of children in media content relay. Therefore, if a peer has a higher fanout, it tends to have more children to relay media content, which increases the probability that it relays media content to other peers. The reason we use fanout instead of uplink bandwidth as the metric of the peer's forwarding ability is to minimize the average end-to-end delay from the source to the receivers. If we track each single message (either in its original form or in an encoded form), it is distributed through a tree. Assigning higher priority to nodes with larger fanout can reduce the average tree height, therefore reduce the end-to-end delay.

We adapted the proposed overlay topology construction algorithm slightly by adding two more constraints.

1. Constraint 1: if the number of outgoing links is equal to $2 * fanout$, no more outgoing links are added;

2. Constraint 2: if the summation of the download bandwidth of a peer's parents is no less than twice of its download bandwidth, no more incoming links are added.

The selection of the ratio value 2 is based on our simulations, which achieves a good tradeoff between system performance and computing complexity.

The pseudo-code of the overlay topology construction algorithm is listed in Table 4.2.

Table 4.2: Overlay Topology Construction Algorithm

```
INPUT: BW_up(i), BW_down(i)
//BW_up(i) and BW_down(i) are upload bandwidth and
//download bandwidth of node i
OUTPUT: overlay topology
BEGIN
  foreach node i
    fanout(i) = BW_up(i)/BW_down(i);
  Sort nodes in a descending order of fanout into list q;
  E = NULL;
  foreach node i in q
    tail = i;
    head = i + 1;
    while head! = NULL
      if sum_{j,(j,head)∈E} BW_down(j) < 2 * BW_down(head)
        E = E + (tail, head);
      head + +;
      if |E_out(tail)| == 2 * fanout(tail)
      //E_out(tail) = {l, l ∈ E, tail(l) = tail}
        break;
END
```

Peer Joining

The media content is encoded into multiple stripes at the source node. As there is no coding between different stripes, it is possible to decompose the network topology graph G into multiple disjoint subgraphs each of which is corresponding to one stripe. Therefore, when a peer joins the system, it will select some of the stripes to subscribe to.

When a peer wants to receive the media content, it will send a JOIN request to the source node. Upon receiving a JOIN request, the source node initiates a

process to determine the maximum flow between the source node and the joining peer. There are many existing algorithms for this problem such as Ford-Fulkerson algorithm proposed in [54] and push relabel algorithm proposed in [55]. Here we adopt the push-relabel algorithm because it is more efficient and it is a distributed algorithm.

We first give a brief review of the push-relabel algorithm. Given a graph $G = (V, E, C)$, a source node $s, s \in V$ and a destination node $t, t \in V$, push-relabel algorithm can find the maximum flow between s and t. In the push-relabel algorithm, each node is assigned a *height* value and an *excess* value. Height is used to control the flow direction. A flow can only be pushed from a higher node to a lower node between two neighbors. Although the difference between the flow entering a node and the flow leaving a node is zero when the algorithm terminates (except nodes s and t), during the execution of the algorithm, the flow difference may be positive, i.e., the flow entering a node is more than the flow leaving a node. We use $excess$ to denote the amount of flow difference which is a non-negative value. The value of a flow pushed between two neighbors can not exceed the residual bandwidth of the link connecting the two neighbors. Initially, the source node has a height of $|V|$ and the destination node has a height of 0. The height of a node v ($v \neq s, v \neq t$) is the number of hops along the shortest path from s to v. The excess of the source node is infinity, i.e. initially the source node will push flow to its neighbor nodes as much as possible. The excesses of other nodes are 0. There are two operations in the push-relabel algorithm: (1) *push* operation which is to push a flow from a node with a larger height to one of its neighbors with a smaller height. The value of the flow is the minimum of the excess of the node and the residual bandwidth of the link over which the flow is pushed. When a flow is pushed over a link, an artificial link connecting the same pair of nodes is added to the topology. The direction of the artificial link is opposite to the link and the capacity of the artificial link is equal to the value of the flow; (2) *relabel* operation is used to update the height values of nodes when no legal push operation can be done. If all the nodes except nodes s and t have 0 excess, the algorithm terminates and the excess of t is the maximum flow between s and t. Otherwise, pick a node with positive excess and increase its height such that it can push a flow to a neighbor.

The push-relabel algorithm is perfect for a distributed system as the flow is determined gradually and locally between a pair of nodes. Thus, we adopt the push-relabel algorithm in our system. Each link is associated with an information vector which includes the following information: link capacity, the stripes which have flows on the link and the bandwidths the stripes occupy, respectively. The link information is used by a peer to decide which stripe to subscribe to and it is collected when a push operation is performed. In particularly, in the push operation, in addition to pushing the excessive flow from the higher node to the lower node, the

algorithm records the corresponding information which is carried along the flow.

When the algorithm terminates, the joining peer should have following information:

1. The paths from the source to the peer,
2. For each link in a path, the stripes which are transmitted over it and the bandwidths occupied by them.

Based on this information, the joining peer selects the stripes to subscribe to. The heuristic rules used for the selection are as follows:

1. Give higher priority to the stripes which have flows in at least one path.
2. Give higher priority to the stripes which occupy more paths.
3. Subscribe to as many stripes as possible.

The first two rules are quite straightforward, because subscribing to the stripes that already have flows in the paths can maximize the utilization of the bandwidth that is already used by the stripes. It is similar to grafting a new receiver to an existing multicast tree in multicast routing. The last rule means that if there is still residual bandwidth after subscribing the stripes based on the first two rules, subscribe to new stripes to fill up the residual bandwidth.

In LC, there are different priorities assigned to different layers as well. For example, the base layer is given the highest priority. However, the priority used here is different from that used in LC. In LC, the priority is mandatory such that the layer with lower priority is useless unless all the layers with higher priorities are received. The priority is used to differentiate different layers. In our scheme, the priority is not mandatory. Peers can choose stripes with lower priority instead of higher priority to encode and every stripe is used to decode at the receivers. Priority is used to improve the bandwidth utilization.

The pseudo-code for the stripe selection algorithm is listed in Table 4.3.

Peer Leaving

When a peer leaves the system, it needs to leave the stripes it subscribes to. It performs the leave procedure repeatedly for each stripe.

To leave a stripe, a peer first checks its position on the subgraph of the stripe. There are two cases:

Case 1: It has no outgoing links. In this case, the leaving peer sends PRUNE message to its parents (maybe more than one) such that they can delete the links from the subgraphs.

Table 4.3: Stripe Selection Algorithm

```
INPUT: path p_i, link e_ij, link capacity c_ij, stripes f_ij^k
       and bandwidth b_ij^k
//link e_ij is the j^th link on path p_i
OUTPUT: stripes to subscribe to
BEGIN
  foreach stripe k
    if bandwidth_is_enough(k)
      Put stripe k into a list l;
  Sort l in a descending order of the number of paths the
  stripe occupies;
  foreach stripe k in l
    if bandwidth_is_enough(k)
      foreach path p_i
        Allocate b * cap_i^k / sum_i cap_i^k bandwidth to stripe k;
END

bandwidth_is_enough(k)
BEGIN
  foreach path p_i
    foreach link e_ij
      cap_ij^k = c_ij - sum_k b_ij^k + b_ij^k;
    cap_i^k = min_j{cap_ij^k};
  if sum_i cap_i^k >= b
    return TRUE;
  else
    return FALSE;
END
```

Case 2: It has one or more outgoing links. In this case, the peer will notify its children to rejoin the system. Meanwhile, it will send PRUNE message to its parents.

Sometimes, peers may leave the system due to crash. In this case, it is impossible for the leaving peer to notify its neighbors. To solve this problem, peers send HELLO messages to its neighbors periodically and maintain a HELLO timer for each neighbor. Receiving a HELLO message triggers a reset of the corresponding HELLO timer. The neighbors detect the abruptly leaving by the timeout of the HELLO timer. After detecting the leaving of the peer, the parents and the children will perform the same operations discussed earlier.

Network Coding

We apply random network coding within each stripe. After encoding the media content into k stripes, the source node sends each stripe by dividing it into groups called $generations$. A generation is a unit for network coding. Only messages within the same generation can be encoded together by network coding. The source node first performs random coding for messages in the same generation. The encoded messages are sent to the outgoing links over which the corresponding stripe has flow.

The relay peers perform random coding when receiving the stripes of the media content. As there is no coding between different stripes, peers only mix the flows from the same stripe and the same generation. When a peer receives flows belonging to the same stripe and generation from its parents, it mixes them up with random coefficients generated from a large Galois field. The mixed flow is sent out only to those outgoing links over which the stripe has flow. By doing this, it is guaranteed that a peer will not receive the flow which is mixed with stripes it does not subscribe to.

The peers decode the messages in the same generation once they receive enough linearly independent messages from their parents. It will acknowledge the source node of its successful receiving of the generation.

4.2.3 Performance Evaluations

In this section, we study the performance of the proposed scheme through simulations. We have implemented the proposed scheme in NS-2 [33]. The network topologies used in the simulations are random transit-stub network topologies generated by GT-ITM software [27]. Peers are selected randomly from the stub networks. The links between the stub networks and the transit network have 8Mbps bandwidth on average. The bandwidth of the links in the transit network is set to be sufficiently high (10,000 Mbps in the simulations) to simulate the network model discussed in Section 6.2.

We compare our scheme with a recently proposed scheme in [21], called LION, as it is the closest work to ours. We use ANC (Adaptive Network Coding) to represent our scheme in the figures in the following.

The simulation adopts following three performance metrics:

Satisfaction: A peer may not receive the media content at its maximum rate (downlink bandwidth). We use *satisfaction* to evaluate the extent the peer is satisfied with its receiving rate. A peer's satisfaction is defined as the ratio of the received rate to its downlink bandwidth. In the simulation, we use the average satisfaction of all peers to represent the satisfaction of the system.

Resilience: Resilience is a performance metric used to evaluate the ability of the

system to handle peers' dynamic join/leave called churn. We let the peers join and leave the system during the time span. We assume that the uptime of a peer follows Poisson distribution [56]. We evaluate the resilience by examining the throughput under dynamic peer join/leave. Throughput is defined as the service the system provides in one time unit. Here the service is the media content received by the receivers. Due to the heterogeneity of the receivers, the receivers are receiving the media content at different rates. We evaluate the volume of the media content received by every receiver in a given time span. The throughput is equal to the summation of the volumes of received media content by all the receivers divided by the length of the time span.

Control Overhead: Control overhead is a performance metric used to evaluate the efficiency of the scheme. It is measured by the number of control packets during the time span. In particular, control overhead includes the packets generated by the push-relabel algorithm and PRUNE messages, etc.

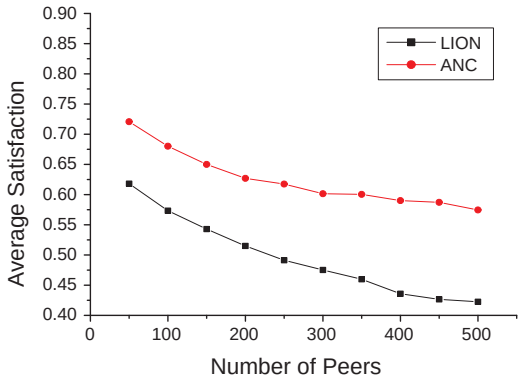
Satisfaction

We first compare the average satisfaction of peers under different sizes of the peer-to-peer streaming system. We use the number of peers in the system to represent the size of the system. To investigate the impact of the topology construction algorithm, we compare the satisfaction using two different topology construction algorithms: one is proposed in this section and the other is proposed in [12].

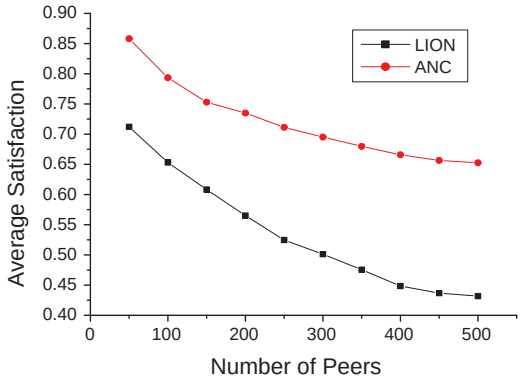
Fig. 4.10 shows the average satisfaction as we change the system size. We can observe that with the increase of the system size, the average satisfaction drops. This is because that when the number of peers increases, the competition of the bandwidth resource is more severe. Thus, it is more difficult to allocate the bandwidth to satisfy every peer's need. For both overlay topology construction algorithms, our proposed scheme ANC performs better than LION regardless of the system size. The advantage is greater when the system size is larger, which suggests that ANC is more scalable than LION. When using the overlay topology construction algorithm proposed in this section, the satisfaction is increased by $13\% - 20\%$.

Resilience

In this subsection, we compare the resilience of the system under different sizes of the system. Fig. 4.11(a) shows the throughput comparison of the two schemes with and without churn. We can see that the proposed scheme ANC achieves about 18% higher throughput than LION without churn and about 48% higher throughput than LION under churn. This can be explained by the fact that when churn occurs, the probability that a peer misses the base layer in LION increases. Without the base



(a)



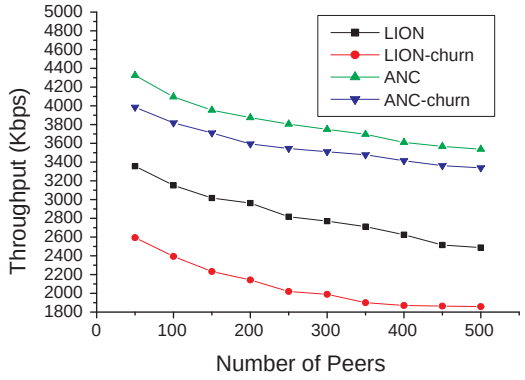
(b)

Figure 4.10: Average satisfaction evaluation. (a) Average satisfaction under system sizes when using the overlay topology construction algorithm in LION; (b) Average satisfaction under system sizes when using the overlay topology construction algorithm in this section.

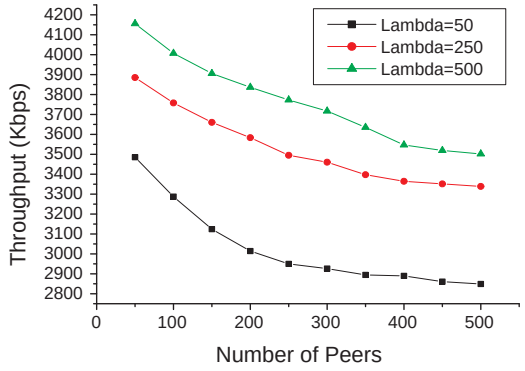
layer, it is impossible for a peer to perform decoding. The advantage of MDC over LC leads to the advantage of resilience of ANC over LION.

Since the uptime of the peers follows Poisson distribution, we use λ to denote the mean uptime of peers. Fig. 4.11(b) shows the simulation results when we set the mean uptime to 50, 250 and 500, respectively. The total simulation time is 1000 (in NS-2 time units). We can see that the shorter the mean uptime, the lower the throughput. This is obvious as a shorter mean uptime implies a higher rate at which the peers join or leave the system. When λ is smaller, the throughput is more sensitive to the system size. This is because that a small system size limits the

optional paths during churn.



(a)



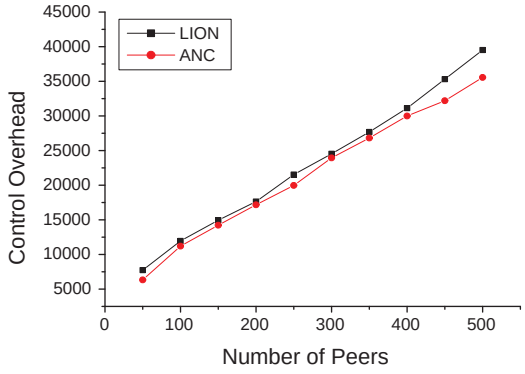
(b)

Figure 4.11: Throughput evaluation. (a) Throughput under different system sizes when peers join/leave dynamically; (b) Throughput under different system sizes and different mean uptimes of peers when peers join/leave dynamically.

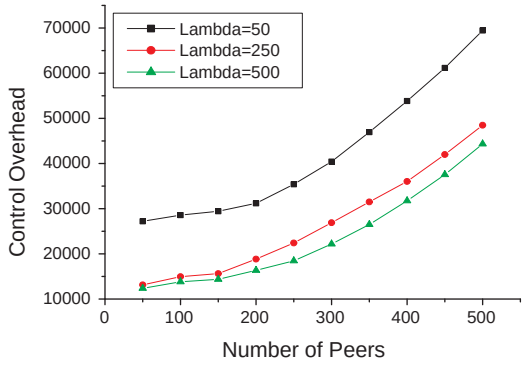
Control Overhead

In this subsection, we compare the control overhead of the system under different sizes of the system. Fig. 4.12(a) shows the curves of the control overhead of the two schemes. With the increase of the system size, the control overhead increases almost linearly for both schemes. The control overhead is mainly caused by the join procedure of peers. As the number of peers increases, the number of control packets increases proportionally as well. The control overhead of LION is slightly

more than that of ANC, because LION floods packets around the system to find a maximum number of link-disjoint paths when building the layered mesh. Fig. 4.12(b) shows the impact of λ on the control overhead. We can see that the control overhead when $\lambda = 50$ is much higher than that when $\lambda = 250$ or $\lambda = 500$, while the difference between $\lambda = 250$ and $\lambda = 500$ is small.



(a)



(b)

Figure 4.12: Control overhead evaluation. (a) Control overhead under different system sizes; (b) Control overhead under different sizes and different mean uptimes of peers.

4.3 Summary

In this chapter, we have proposed a scheme to optimize the overlay topology for live peer-to-peer streaming systems. One of the merits of the proposed scheme is that it

can make efficient use of the uplink bandwidth of the peers and satisfy the heterogeneous downloading rate requirements of the peers as much as possible. Compared to other overlay topology construction schemes, the proposed scheme can handle both heterogeneous uplink bandwidth and heterogeneous downlink bandwidth at the same time. Peers with different downloading bandwidths can receive the media content at different rates without bandwidth wasting. Besides, the distributed algorithm constructs an adaptive overlay topology which can adapt itself to the changing peers such that the end-to-end delay and link stress are minimized. Simulation results show that the proposed scheme outperforms MDM by about 30% with respect to the average peer satisfaction. In addition, the proposed scheme achieves less link stress than MDM.

Thereafter, we have presented an adaptive network coding scheme for peer-to-peer media streaming systems. Compared to other peer-to-peer media streaming schemes, our scheme has the following advantages. (a) Heterogeneity support. As most peers are individual computers connected to the Internet through heterogeneous access links, our scheme can maximize the bandwidth utilization of access links and therefore maximize the total throughput of the system. (b) Resilience. Churn is a common problem in peer-to-peer networks. With the help of MDC, peers can reconstruct the media content with a subset of the stripes. (c) Scalability. Media content is distributed through a peer-to-peer network. With the increase of the network size, the total available bandwidth also increases.

Chapter 5

A Linear Inter-Session Network Coding Scheme for Multicast

Most existing works on network coding in the literature focused on a single multicast session. For example, Li et al. [15] showed that linear network codes are sufficient to achieve the multicast capacity. Kotter et al. gave an algebraic characterization for a linear network coding scheme in [25]. They also gave an upper bound on the field size and a polynomial time algorithm to verify the validity of a network coding scheme. Ho et al. presented a random linear network coding approach in [18, 36] in which nodes generate edge vectors randomly. The linear network coding scheme generated by this approach is not always valid. They proved that the probability of failure is $O(1/q)$ where q is the size of the finite field. In contrast to the random network coding, Jaggi et al. proposed a polynomial deterministic algorithm in [16] which can construct deterministic linear network coding schemes for multicast networks.

An extension to the network coding for a single multicast session is to apply network coding to multiple concurrent multicast sessions, which is called *inter-session network coding*. The benefit of inter-session network coding can be demonstrated by the example shown in Fig. 5.1, where nodes s_1 and s_2 are the respective sources of the two multicast sessions and nodes r_1 and r_2 are the receivers of both multicast sessions. We can see that edge $c - d$ becomes a bottleneck if no inter-session network coding is employed. Inter-session network coding can eliminate the bottleneck by encoding two messages received at node c and send them along link $c - d$ together. Both r_1 and r_2 can recover the messages from s_1 and s_2.

However, although it is an extension of the single multicast session, inter-session network coding is much more complex. Dougherty *et al.* [57] showed that linear network coding is insufficient to achieve multicast capacity for multiple multicast sessions. Li *et al.* [58] showed that there is no coding gain for an undirected graph. Also, even we confine the encoding function to linear functions, it is a NP-hard

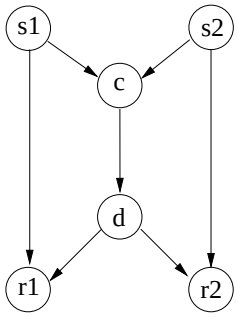


Figure 5.1: An example that inter-session network coding achieves higher throughput.

problem [25] to find such a linear network coding assignment. Wang *et al.* [51] gave some preliminary work on inter-session network coding for two simple multicast sessions from a graph theory point of view. They proved an equivalent condition under which there exists a linear network coding scheme for two multicast sessions. Wu [59] applied random network coding to all the sessions after transforming the network topology such that the source can only reach the receivers that are interested in the source.

To the best of our knowledge, this is the first work that provides a practical method to construct a linear network coding scheme for inter-session network coding and evaluate its performance. In this chapter, we will investigate inter-session network coding by providing heuristic algorithms and conducting extensive simulations. We will propose an approach to identifying the situations where it is the most profitable to do inter-session network coding and which sessions should be encoded together. Two metrics will be introduced to characterize the overlap among sessions. The sessions are divided into multiple groups based on the metrics such that the overlap among sessions in the same group is above a threshold. The inter-session network coding is constrained within the same group. We will also propose two heuristic algorithms, the deterministic algorithm and the random algorithm, to construct the linear coding scheme on the divided groups.

5.1 Preliminaries

We model the network as a directed acyclic graph (DAG), $G = (V, E)$, where V is the node set and E is the edge(link) set. An edge e can be represented by an ordered node pair (x, y) where $x, y \in V$. y is called the head of the edge and x is called the tail of the edge. The messages can only be transmitted from x to y.

Each node has one or more incoming edges and one or more outgoing edges except that source nodes have no incoming edges and receivers have no outgoing

edges. Each edge, also called link, has a link capacity of 1, which means that it can only transfer 1 unit of data at 1 time slot. For a network with link capacities larger than 1, we transform the network based on the following rule: for each link with link capacity lc ($lc > 1$), we replace the link with lc links such that each of them has the same head and tail as the original link and has a link capacity of 1. In the case that a receiver has one or more outdoing edges, we add a virtual receiver to replace it and multiple virtual links from the original receiver to the virtual receiver. The number of the virtual links equals to the number of incoming edges of the original receiver.

A multicast session is represented by a pair (s,M) where $s \in V$ represents the source of the session, and $M \subset V$ represents the set of receivers[1]. We assume that there are k concurrent multicast sessions represented by (s_1,M_1), (s_2,M_2), ..., (s_k,M_k). For multicast session (s_i, M_i), the multicast capacity is denoted by C_i, and the transmission rate is denoted by r_i. Clearly, $r_i \leq C_i$. We call vector $\pi = (r_1, r_2, \ldots, r_k)$ a *rate vector*. A rate vector is achievable if it is possible to transmit all the sessions at the respective rates in the rate vector. The *achievable rate region*, or *rate region* for short, is the set of all achievable rate vectors.

The inter-session network coding problem for multiple multicast sessions can be described as follow:

Given a DAG, $G = (V, E)$, and k multicast sessions, (s_1,M_1), (s_2,M_2), ..., (s_k,M_k), find the rate region for the k multicast sessions and a method to achieve the rate region.

A special case of inter-session network coding is that each multicast session constructs a network coding scheme for its own session. As the network coding is only within the same session, we call it intra-session network coding. Intra-session network coding is easy to implement. However, it is not the optimal solution in most cases, because it can achieve the optimal rate region only if we can find a subgraph for each session such that these subgraphs are edge-disjoint and the multicast capacity of the subgraph is no less than the multicast capacity of the corresponding session. Apparently, this is difficult to achieve in general. In fact, the butterfly network in Fig. 5.1 is an example that intra-session network coding is inferior to inter-session network coding.

In this chapter, we are interested in solving the inter-session network coding problem through linear network coding due to its simplicity and easy to implement in hardware. As discussed earlier, linear network coding alone can not achieve the optimal rate region. Thus, we focus on maximizing the rate region with linear inter-session network coding. In particular, we are interested in answering following questions:

- *Question 1*: Under what condition inter-session network coding outperforms

[1]We assume $s \notin M$

intra-session network coding? We are especially interested in throughput improvement. As intra-session network coding is a special case of inter-session network coding, the maximum rate region of inter-session network coding is no less than that of intra-session. Therefore, the maximum throughput of inter-session network coding is no less than that of intra-session.

- *Question 2*: How much can inter-session network coding improve the performance compared to intra-session network coding? We are interested in quantifying the benefit brought by applying the inter-session network coding. We believe the benefit is a function of some factors which leads to the next question.

- *Question 3*: What are the main factors that affect the performance? We are interested in finding the factors which dominate the benefit of inter-session network coding. These factors are then taken into consideration when constructing a practical inter-session network coding scheme.

5.2 Heuristic Algorithms for Linear Inter-Session Coding for Multicast

In this section, we propose heuristic algorithms for constructing a linear network coding scheme to achieve a near optimal rate region. To fully examine the behaviors and properties of inter-session network coding, we propose two heuristic algorithms: one is deterministic and the other is random.

A naive way to apply inter-session network coding to multiple sessions is to combine all the sessions into one "big" session. The source node of the big session is an artificial node connecting to the original source nodes of the multiple sessions, and the receivers of the big session are the union of the receivers of the multiple sessions. Then an intra-session linear network coding is applied to this big session. However, this naive solution can hardly improve the performance, or even worsen the performance. This is because different sessions have messages destined to different sets of receivers. When considered as one big session, the probability a receiver receives a message it is not interested increases, which causes more wasting of bandwidth. To avoid such situation, we divide the sessions into groups and perform intra-session network coding within each group. The selection of group is performed carefully such that the benefit of inter-session network coding overwhelms the overhead. We adopt two metrics for the group division. We first describe the two metrics.

5.2.1 Two Metrics for Session Division

The goal of mixing different sessions is to increase the throughput by eliminating the bottlenecks caused by shared links of different sessions. If there are no shared links between two sessions, inter-session network coding is not necessary or even impossible. Therefore, we have the following heuristic rule for the algorithm: sessions that overlap more will benefit more from inter-session network coding.

Now the problem becomes how to characterize the overlap among sessions. We expect to find a method to quantify the shared links among sessions in order to determine which group of sessions should be considered together. Before we dive into the details, we introduce a notion called $field$ to facilitate our presentation. A $field$ is a function which maps a session to a subgraph of the network topology. Recall that given a multicast session (s_i,M_i), it is always possible to find C_i edge-disjoint paths from the source to any of the receivers. Thus, there are a total of $C_i|M_i|$ such paths. A $field$ is formally defined as follows:

$$\begin{aligned} &field(s_i) = G'(V', E') \text{ where} \\ &V' = \{v' | v' \in \text{one of the } C_i|M_i| \text{ paths}\}, \\ &E' = \{e | e \in E, head(e) \in V' \text{ and } tail(e) \in V'\} \end{aligned} \tag{5.1}$$

Each session has its own field. It is one of the subgraphs over which the session can achieve the multicast capacity through network coding. Fields may overlap, that is, a link may belong to multiple fields. Now we can quantify the overlap among sessions by adopting the following two metrics:

- *Overlap Ratio (OR)*: the overlap ratio measures the overlap by the percentage of the overlapped links between two sessions. Suppose the two sessions are s_i and s_j. The overlap ratio of the two sessions can be calculated by the following function:

$$OR(s_i, s_j) = \frac{CL(s_i, s_j)}{|E'(s_i)| + |E'(s_j)| - CL(s_i, s_j)} \tag{5.2}$$

  where $CL(s_i, s_j)$ represents the number of common links of $field(s_i)$ and $field(s_j)$, and $|E'(s)|$ represents the number of links in $field(s)$. From the definition, we can see that this metric gives a higher priority to the sessions that have the most common links.

- *Overlap Width (OW)*: the overlap width measures the overlap by the percentage of the overlapped paths between two sessions. Here the paths refer to the edge-disjoint paths in the field. The overlap width of the two sessions can be

calculated by the following function:

$$OW(s_i, s_j) = \frac{\sum_{m=1}^{C_i} \sum_{n=1}^{C_j} P_{mn}}{C_i C_j} \tag{5.3}$$

where $P_{mn} = 1$ if the m^{th} path in $field(s_i)$ shares one or more links with the n^{th} path in $field(s_j)$ or $P_{mn} = 0$ otherwise. From the definition, we can see that this metric gives a higher priority to the sessions that have the most number of paths crossed.

We use a tunable parameter δ ($0 \leq \delta \leq 1$) as a threshold. If either $OR(s_i, s_j)$ or $OW(s_i, s_j)$ is greater than δ, we put session (s_i, M_i) and (s_j, M_j) into the same group. After checking all the sessions, we can divide the multiple sessions into several groups with each group composed of one or more sessions. The groups are disjoint with each other. Now we can apply network coding to each group respectively.

5.2.2 The Deterministic Algorithm

The deterministic algorithm constructs the linear network coding scheme by assigning a fixed edge vector to each edge. The edge vectors are designed such that the receiver can recover the original messages based on its received messages. Without inter-session network coding, nodes can only mix the messages generated by the same source. If inter-session network coding is permitted, nodes can mix the messages from different sources. We introduce a parameter called $mixability$ to describe the maximum number of sessions that are permitted to be mixed together. If $mixability = 1$, inter-session network coding degenerates to intra-session network coding. If $mixability = k$, all the sessions are considered as one unified session and the network coding scheme is constructed on the unified session. We will evaluate the effect of $mixability$ on the performance in Section 5.3.

As discussed earlier, the multicast sessions are divided into different groups such that groups are disjoint with each other. All the sessions within the same group are considered as one unified session. The union of the sources forms the sources of the unified session. The union of the receiver sets of the sessions forms the receiver set of the unified session. We add one artificial source node connected to all the sources to simplify the linear network coding construction. Now the inter-session network coding for multiple sessions is transformed to an intra-session network coding for the unified session. There are several existing methods to construct a deterministic linear network coding scheme for a single multicast session. Here we adopt the method proposed in [60] for each group of sessions respectively. Given a group, we first preprocess the graph to find a minimum subgraph which has the

same multicast capacity as the original graph. This preprocessing can greatly reduce the graph size to be processed. The subgraph can be looked as the union of the fields of the sessions in the group. Usually a field of a session is a subgraph which is much smaller than the whole graph. Then we can apply the hypergraph based approach in [60] to the subgraph to find a valid linear network coding scheme.

5.2.3 The Random Algorithm

The linear network coding scheme can be constructed not only in a deterministic way, but also in a random way. In random linear network coding, nodes mix the received messages and assign random coordinates to the edge vectors. Receivers keep receiving the encoded messages until they have enough independent messages to decode. In a random linear network coding without inter-session network coding, nodes can only mix messages generated by the same source. With inter-session network coding, nodes can mix the messages from different sources.

If there is no constraint for a node to mix and forward messages, that is, nodes always encode all the messages they receive and send to all the outgoing edges, the receivers will eventually be able to decode the original messages, since the sources will keep sending the messages until all the receivers decode the messages successfully. However, this method is inefficient as it mixes all the sessions and treats them as one session. To avoid this problem, we divide the sessions into groups in a similar way to that used in the deterministic algorithm. Only messages within the same group can be encoded together, and nodes should obey the following rules:

1. If a node receives a message which contains the information generated by a source that does not belong to the group and the sessions whose fields include the node, the message should be discarded;

2. Encode the received message with other messages in the same group and send to all the outgoing edges.

Based on the above two rules, we constrain the messages generated from one source within its corresponding group. The reason is two folds. First, the receivers whose corresponding session is outside the group will never receive messages from the group. Thus the receivers can collect a sufficient number of independent messages to decode in a shorter time. Second, the messages will not flow aimlessly and the network bandwidth is saved.

5.3 Performance Evaluations

We have conducted extensive simulations to evaluate the performance of the proposed algorithms when the inter-session network coding is employed. In this sec-

tion, we present our simulation results and compare different approaches.

5.3.1 Simulation Setups

We use NS-2 [33] as the network simulator. The network topologies are generated by GT-ITM software [27] which is a degree-based Internet structural topology generator. Each topology consists of 1000 nodes with an average node degree of 2 to 10 depending on the simulation scenario. Both the source nodes and the receiver nodes are selected randomly. Any node can not be a source node and a receiver node at the same time.

The simulation includes two parts. First, we implement the intra-session network coding and use it as a benchmark. We compare the two heuristic algorithms with the benchmark under different situations. Second, we study the behaviors of the inter-session network coding by tuning the parameters: $mixability$ and δ.

The simulation adopts the following two performance metrics:

- *Throughput*: Throughput is defined as the service the system provides in one time unit. Each node maintains an incoming buffer for each incoming edge and an outgoing buffer for each outgoing edge. The size of the buffer is 1. All the source nodes keep sending messages to the next node as long as the corresponding incoming buffer is not full. Each message contains a sequence number which indicates its position in the message stream. After a certain period of time, we stop all the message streams. The number of delivered messages is represented by the largest sequence number of the message that is received by all the receivers. The throughput of the system is the average of these numbers.

- *Bandwidth consumption*: We defined the network bandwidth used to deliver the messages as bandwidth consumption. As there are no control messages involved in the delivery, the bandwidth consumption is only caused by data messages. A data message going through one link contributes 1 unit to the bandwidth consumption.

5.3.2 Performance of Inter-Session Network Coding

We first compare the heuristic algorithms with the intra-session network coding in some general scenarios.

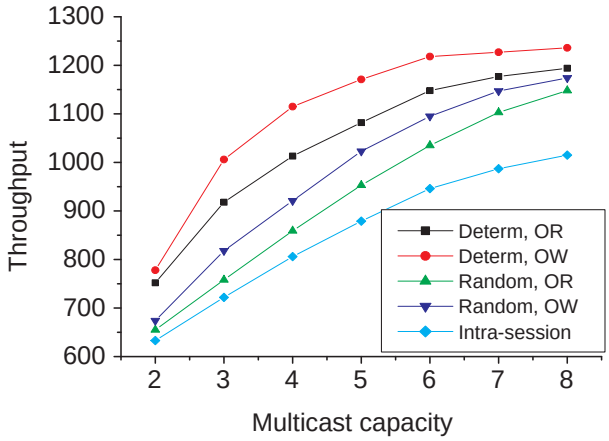
Multicast Capacity

Figure 5.2(a) shows the throughput evaluation under different multicast capacities (average multicast capacity of all sessions). There are five curves which represent

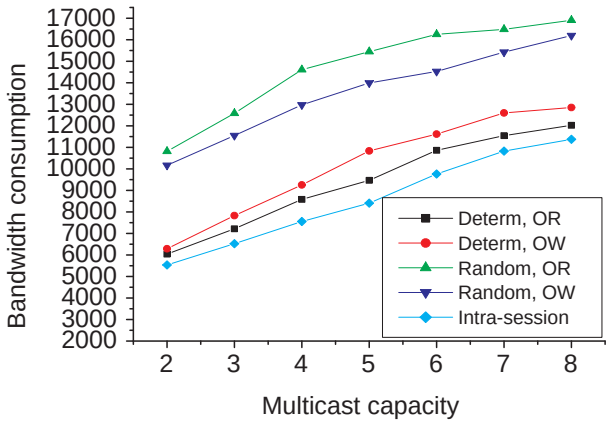
the deterministic algorithm using OR metric (denoted by "Determ, OR"), deterministic algorithm using OW metric (denoted by "Determ, OW"), random algorithm using OR metric (denoted by "Random, OR"), random algorithm using OW metric (denoted by "Random, OW"), and intra-session network coding (denoted by "Intra-session"), respectively. From the figure, we can see that both the deterministic algorithm and the random algorithm achieve higher throughput than the intra-session coding. It indicates that inter-session network coding can achieve better throughput than intra-session network coding. If we adopt OW metric, the throughput of the deterministic algorithm is about 30% higher than that of the intra-session coding. Also, the throughput of the deterministic algorithm is higher than that of the random algorithm. This is due to the possibility of failing to decode with random coding. We can see that with the increase of the multicast capacity, the throughput increases as well. The algorithm with metric OW performs better than that with metric OR with respect to the throughput. This can be explained from the definitions of the two metrics. OR emphasizes the shared links between sessions, in which the sessions with most common links are encoded together. OW emphasizes the crossed paths, in which the sessions with most crossed paths are encoded together. The throughput is determined by the bottleneck of the system. Encoding the sessions with most crossed paths together implies that the bottleneck is relaxed to the maximum extent. The gap between the heuristic algorithms and the intra-session coding becomes larger with the increase of the multicast capacity. It indicates that the system benefits more from the inter-session coding when the multicast capacity is larger due to the higher possibility to encode different sessions together.

Figure 5.2(b) shows the bandwidth consumption evaluation under different multicast capacities. As the figure shows, the bandwidth consumption increases with the increase of the multicast capacity. The deterministic algorithm achieves lower bandwidth consumption than the random algorithm. We observe that the bandwidth consumption of the deterministic algorithm is slightly higher than that of the intra-session coding. This can be explained by the mechanism of inter-session network coding. For the sessions in the same group, the messages are encoded and transmitted together on bottleneck links which are shared by sessions. To this end, the traffic is reduced. On the other hand, for receivers to recover the original messages, it is inevitable to generate more messages for the receivers to decode the messages, which increases the traffic. However, these messages are usually transmitted through some relatively lightly loaded links. Based on the simulation, the increased traffic is slightly higher than the saved traffic. Given that inter-session network coding can increase the throughput by about 30% compared to the intra-session network coding, the minor increase in bandwidth consumption on some lightly loaded links is quite acceptable. We can also see that the algorithm with metric OR saves more network bandwidth than the algorithm with the metric OW.

Since OR maximizes the shared links between the sessions encoded together, data packets are delivered only once on the shared links, which reduces the bandwidth usage. It indicates that metric OR is more suitable for applications where network bandwidth is scarce and expensive.



(a)



(b)

Figure 5.2: The performance comparison under different multicast capacities. (a) throughput; (b) bandwidth consumption.

From the above discussion, we can see that if we adopt metric OR instead of metric OW, the bandwidth consumption can be saved. On the other hand, adopting OW can achieve higher throughput than OR. This is a general observation

throughout the entire simulation. Since throughput is generally a critical performance metric in most networks and the difference of bandwidth consumption between metric OR and metric OW is small, in the rest of the simulation figures, we will draw the curve of metric OW only for clarity. Thus, by the deterministic (random) algorithm, we refer to the deterministic (random) algorithm adopting metric OW.

Session Size

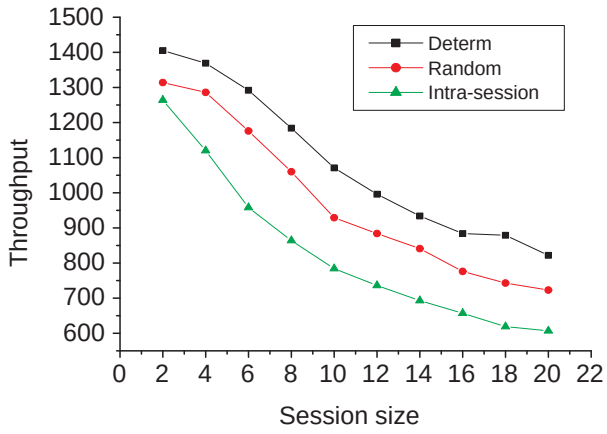
Figure 5.3 shows the throughput and bandwidth consumption under different session sizes (the session size is the number of receivers in the session). We let all the sessions have the same group size. As the figure shows, with the increase of the session size, the throughput drops sharply. This is because that a larger session size increases the possibility of overlap between sessions. The interference caused by overlap will drag the throughput down. However, the drop rate of the heuristic algorithms is lower than that of the intra-session due to the inter-session network coding.

We can also see that the bandwidth consumption increases when the session size increases. The bandwidth consumption of the random algorithm is highest and it increases dramatically when the session size is greater than 8. This is due to the increased possibility of failing to decode the messages at the receivers.

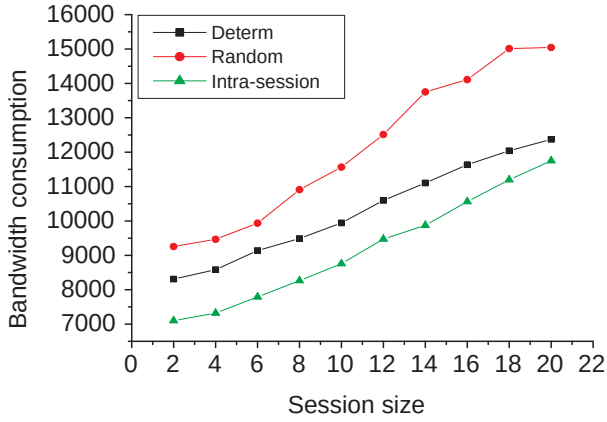
From the above simulation results, we can draw the following conclusions: the deterministic algorithm achieves higher throughput than the random algorithm and the intra-session network coding (specifically, about 13% higher than the random algorithm and 30% higher than the intra-session network coding). The random algorithm has the highest bandwidth consumption (specifically, about 35% higher than the deterministic algorithm and 45% higher than the intra-session network coding).

5.3.3 Inter-Session Network Coding Parameters

There are two important tunable parameters in the simulation of the inter-session network coding: $mixability$ and the threshold δ. $Mixability$ is used to limit the maximum number of sessions which are encoded together. When $mixability = 1$, the system degenerates to the intra-session network coding. The threshold δ is used to control the condition under which the sessions can be encoded together. When the threshold is high, the possibility to encode different sessions is low. In this subsection, we will examine how these two parameters affect the system performance through simulations.



(a)



(b)

Figure 5.3: The performance comparison under different session sizes. (a) throughput; (b) bandwidth consumption.

Mixability

Figure 5.4(a) shows the evaluation for the throughput and bandwidth consumption under different $mixability$ values. We can see that the throughput of both the deterministic algorithm and the random algorithm experiences a rise followed by a drop. During the rise period, the deterministic algorithm has a steeper slope which indicates that the deterministic algorithm exploits the inter-session coding better than the random algorithm when $mixability$ is small. However, when $mixability$

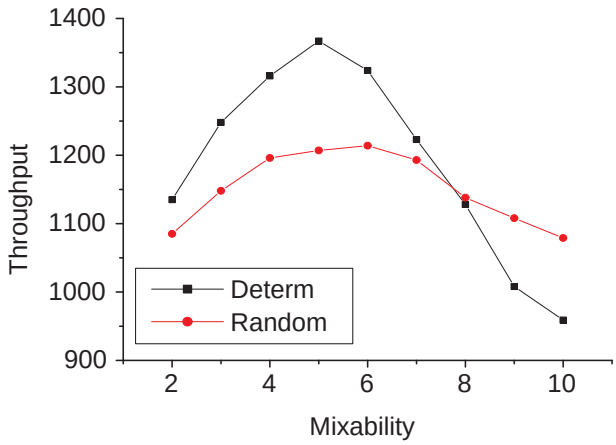
is greater than 6, the throughput of the deterministic algorithm drops dramatically with the increase of $mixability$. The throughput of the random algorithm achieves its maximum when $mixability$ is around 9 after which it drops slowly. When the $mixability$ is large, the throughput of the random algorithm is higher than that of the deterministic algorithm.

At first glance, this performance degradation of heuristic algorithms may be surprising, since one may expect that the throughput of the inter-session network coding should not lower than that of the intra-session network coding. However, this is due to the fact that when two sessions with a large difference in their multicast capacities are encoded together, the average throughput of these two sessions is lower than that when only the intra-session network coding is used. This is because that if the two sessions have a large difference in multicast capacities, the session with a greater multicast capacity is dragged down by the other session as the capacity of the shared link is divided into half in inter-session coding instead of being allocated according to the different rate requirements by sources.

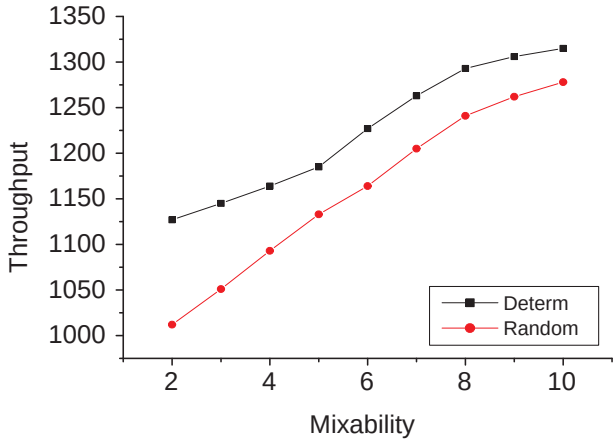
Based on this observation, we revise the heuristic algorithms by limiting the difference between the greatest multicast capacity and the smallest one in the same group. To do this, we compare their multicast capacities before we calculate the OR or OW metric for two sessions. If the difference is large than a specific value (denoted by ρ), no overlap metric is calculated, and these two sessions will not be put into one group. Otherwise, whether the two sessions are put into one group is based on the overlap metric. The determination of the optimal value of ρ depends on the average session size of the two sessions. The larger the session size, the smaller the optimal value of ρ. It indicates that if two sessions have a large average session size, it is more critical to have a small multicast capacity difference. The reason is that the throughput degradation due to the multicast capacity difference is more severe in this case as it involves more receivers when the session size is large.

Figure 5.4(b) shows the simulation results based on the revised algorithms. As an example, we plot the figure for the case when the average session size is 16 and $\rho = 4$. We can see that now the throughput always increases as $mixability$ increases although the increase rate is slower than the previous simulation. It indicates that the improved algorithm can eliminate the throughput degradation due to the large multicast capacity difference.

Figure 5.5 shows the bandwidth consumption under different mixability values. With the increase of the mixability, the bandwidth consumption increases slightly. When the mixability is large, bandwidth consumption stays at the same level. This indicates that when $mixability$ is greater than 8, the group division remains the same.



(a)



(b)

Figure 5.4: The performance comparison under different $mixability$ values. (a) throughput without considering the multicast capacity difference; (b) throughput considering the multicast capacity difference.

Threshold δ

Figure 5.6(a) shows the throughput when we tune the threshold δ from 0.1 to 0.9. As can be seen, the throughput drops as the threshold increases. The throughput drops faster when δ is greater than 0.4. This is because that when δ is large, the possibility that overlapped sessions are encoded together becomes smaller. When δ

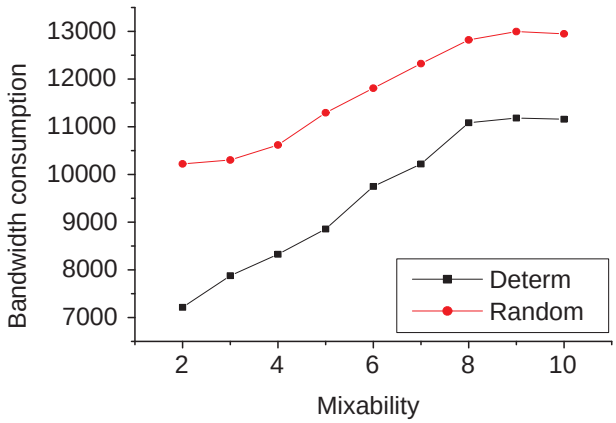


Figure 5.5: The bandwidth consumption comparison under different $mixability$ values.

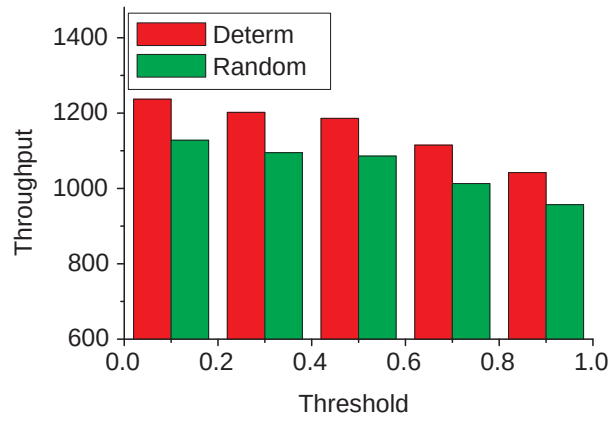
is small, the throughput drops slightly, which indicates that the number of sessions affected by the threshold is small. It implies that for most of the sessions, the value of overlap metric is greater than 0.4.

Figure 5.6(b) shows the bandwidth consumption under different threshold values. The bandwidth consumption decreases with the increase of δ. This is another evidence that the traffic becomes less when the inter-session coding possibility is smaller.

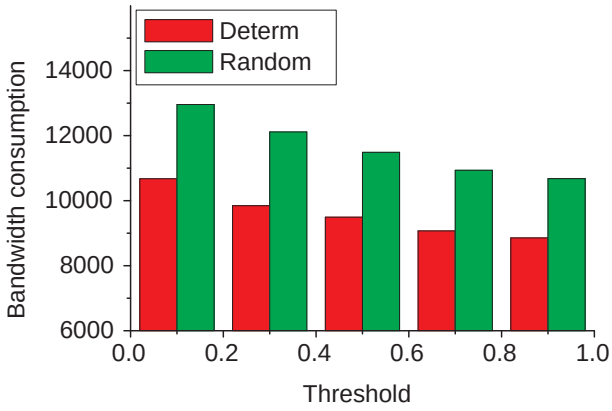
From the above simulation results, we can draw the following conclusions: With the increase of $mixability$, both the throughput and the bandwidth consumption become higher; With the increase of δ, both the throughput and the bandwidth consumption become lower. When designing an inter-session network coding scheme, it is necessary to consider all the influential parameters: $mixability$, δ and ρ.

5.4 Summary

Network coding is a promising technique to improve the resource efficiency for multicast networks. In this chapter, we have investigated the linear inter-session network coding for multicast. The contribution of this chapter is three folds. First, we proposed a practical inter-session network coding scheme for multicast and implemented in NS-2. Second, we introduced two different metrics to characterize the benefit of inter-session network coding with each metric having its own application targets. Third, we studied the performance of inter-session network coding from both the deterministic coding and random coding perspectives. Our simula-



(a)



(b)

Figure 5.6: The performance comparison under different δ values. (a) throughput; (b) bandwidth consumption.

tion results show that the inter-session network coding outperforms the intra-session network coding by about 30% in terms of throughput in most cases. In addition, the deterministic algorithm achieves higher throughput and less bandwidth consumption than the random algorithm.

Chapter 6

A Service-Centric Multicast Architecture and Routing Protocol

As we mentioned in 1, traditional multicast is implemented in network layer through multicast routing protocols running on routers. Network layer multicast has not been deployed in reality due to some technical and non-technical reasons. As a result, scalable and efficient support for multicast communication remains to be a critical and challenging issue in networking research. In this chapter, we reexamine the technical issues in existing multicast protocols and present a new angle to solve these problems.

The main concern of a multicast routing protocol is how to efficiently and effectively construct multicast trees and how to manage multicast sessions. For network-wide multicasting, a network can be modeled as a graph with the nodes representing the routers and the edges representing links between routers as shown in Fig. 6.1(a). The internal structure of a generic router is showed in Fig. 6.1(b). In the rest of the chapter, we will use "router" and "node" interchangeably.

Traditional multicast protocols construct and update the multicast tree in a distributed manner, which causes two problems: first, since each node has only local or partial information on the network topology and group membership, it is difficult to build an efficient multicast tree; second, due to the lack of complete information, broadcast is often used when transmitting control packets or data packets, which consumes a great deal of network bandwidth.

Quality-of-Service (QoS) is another important issue in constructing multicast trees, as many multicast applications are QoS-sensitive. For example, real-time audio/video stream requires bandwidth/delay guarantees. Constructing a multicast tree that can satisfy multiple QoS constraints is difficult and in fact is NP-hard. Thus, heuristic algorithms are usually developed to find an approximation solution. A heuristic algorithm should achieve a good balance between the optimality of the multicast tree and the time complexity of the algorithm.

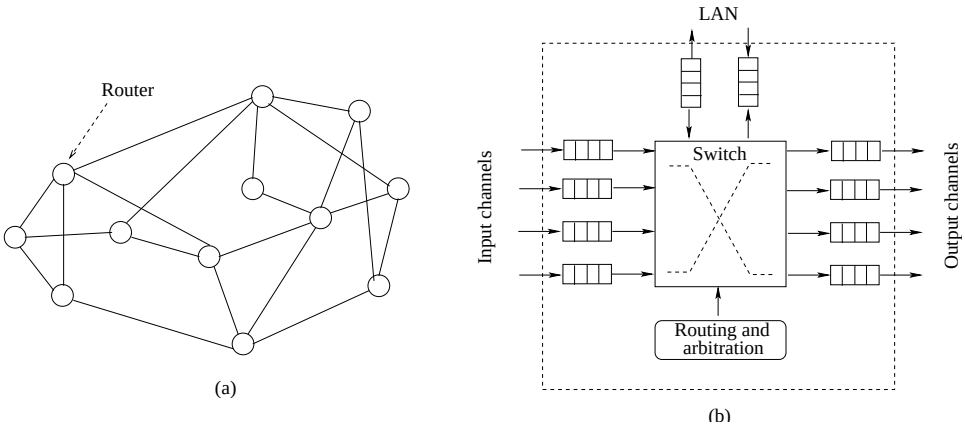


Figure 6.1: (a) An example of a WAN; (b) Internal structure of a generic router.

In this chapter, we address the problem of providing efficient and flexible multicast services and constructing a delay constrained minimum cost multicast tree for dynamic multicast groups.

6.1 Preliminaries

We first introduce some existing multicast routing protocols and multicast tree construction algorithms. Then we analyze the drawbacks and present the motivation of our work.

6.1.1 Existing Multicast Routing Protocols

There have been extensive research and development activities in the area of multicast routing in recent years, see, for example, [2, 3, 4, 5, 6, 7, 8, 28, 61, 62, 63, 64, 65, 66]. Multicast routing protocols can be categorized into two types: Shortest Path Tree (SPT) based multicast routing protocols and Shared Tree (ST) based multicast routing protocols.

SPT-based protocols build a separate multicast tree for each (*source*, *group*) pair rooted at the source. DVMRP (Distance-Vector Multicast Routing Protocol) [3] and MOSPF (Multicast Extensions to Open Shortest Path First Protocol) [**?**] are SPT-based protocols. In DVMRP, the multicast tree is built by "flooding-and-pruning." When a source router has multicast packets for a specified group, it first floods the packets throughout the network; then the routers that do not belong to the group will respond a prune message back to the source router. After this "flooding-and-pruning" process, a SPT multicast tree is constructed which connects the source

router to each destination router by the shortest delay path. When there is a newly joining/leaving group member or the topology changes, DVMRP relies on the instant membership joining information and the periodical membership updating information to maintain a dynamic multicast tree. MOSPF makes use of the feature of OSPF (Open Shortest Path First Protocol) [63] that each router keeps the network topology and link state information to construct the SPT multicast tree. MOSPF extends OSPF by adding a new type of packet, group-membership-LSA packet which pinpoints the locations of all group members. Whenever a host joins/leaves a group, this information is distributed by flooding a group-membership-LSA packet throughout the network. Based on the information of the network topology and the group membership, each router can build an identical multicast tree for a (*source, group*) pair.

On the other hand, ST-based protocols create one tree for the entire group, that is shared by all the sources. The shared tree is rooted at a core router that is publicized to all sources by some mechanism. Core-Based Tree (CBT) [5], Protocol-Independent Multicast Sparse Mode (PIM-SM) [6], and Simple Multicast (SM) [64] are ST-based protocols. In CBT, each group has a corresponding core which is a router chosen by some election mechanism or hash function. The multicast tree is rooted at the core which is shared by all the sources. The shared multicast tree is constructed as follows: Each host that wants to join the group sends a join message along the shortest path to the core; the join message stops at the core or a router which is already on the tree; then the core or the router sends an acknowledgement to the joining host along the same path that becomes a part of the tree after the joining host receives the acknowledgement. Once the multicast tree is established, a source can transfer the packet by first sending the packet to the core and then the core sends the packet to group members along the multicast tree. PIM-SM protocol is quite similar to CBT, but it also allows to create a source-based shortest-path tree on behalf of their attached group members. Thus, PIM-SM is a hybrid routing protocol in a strict sense. SM protocol extends CBT with a major difference that SM identifies a multicast group by the combination of the core node address and the multicast address, and thus eliminates the need for the uniqueness of multicast address across the Internet.

6.1.2 Existing Multicast Tree Construction Algorithms

A multicast tree spans the source and all the group members involved in the multicast communication. It plays a crucial role in multicast. Since data is delivered along the multicast tree, the end-to-end delay, network bandwidth usage and delay jitter are mainly determined by the multicast tree. Though any tree that spans the source and all the group members could be a multicast tree, it is always preferred to

find a multicast tree with minimum cost. In this chapter, tree cost is defined as the sum of link cost of all the links in the tree.

Finding a multicast tree with minimum cost is called Steiner tree problem and it is a well-known NP-hard problem [67, 68]. Many heuristic algorithms have been proposed. Kou, Markowsky and Berman proposed the KMB algorithm in [69], which achieves the best approximation ratio among all proposed algorithms. In KMB algorithm, the network is abstracted as a complete graph in which each node represents the source or one of the group members and the cost of the edge equals the lowest cost among all paths connecting the two nodes. KMB finds the minimum spanning tree of the complete graph by using Prim's algorithm [24]. Then each edge in the spanning tree is replaced by the original path in the network. If the replacement causes a loop, KMB constructs the minimum spanning tree for the resulting graph. Finally, the leaf nodes that are neither source nor group member are deleted.

Kompella, Pasquale and Polyzos proposed the KPP algorithm [70] that extends KMB to support delay constraint. KPP takes two link parameters into account: link cost and link delay. The delay constraint requires that the end-to-end delay should be less than a threshold Δ. KPP constructs a near minimum cost tree in which the delay between every two nodes (source or group member) is less than the threshold.

In [71], Sriram, Manimaran and Murthy proposed the Controlled Rearrangement for Constrained Dynamic Multicasting (CRCDM) algorithm to construct a delay constrained minimum cost multicast tree. The CRCDM algorithm is based on a concept called Quality Factor (QF) that represents the usefulness of a portion of the multicast tree to the overall multicast session. It first constructs a delay constrained multicast tree. When after some joining/leaving operations the QF is below a threshold, the algorithm rearranges the multicast tree to minimize the tree cost without violating the delay constraint.

6.1.3 Problems in Existing Approaches and Our Contributions

In general, SPT-based multicast routing protocols have three problems. First, SPT-based multicast routing introduces the scalability problem for a large network in terms of routing table storage since routers need to store routing information for each (*source*, *group*) pair. Second, adopting DVMRP or MOSPF wastes a large portion of the network bandwidth due to flooding, although in different ways. Third, the multicast trees generated in DVMRP or MOSPF are shortest path trees, which may not be the lowest cost multicast trees. ST-based multicast routing protocols have been proposed to overcome the scalability and bandwidth wasting problems. However, the ST-based multicast routing protocols introduce new problems. First, the multicast communication from a source to a multicast group may not be very

efficient in most cases in terms of multicast tree cost and communication delay due to the shared multicast tree. Secondly, the elected core has the same architecture as any other routers in the network, thus has limited computing and packet forwarding capability. Moreover, the ST-based approach may cause traffic jam around the core, since the packets from multiple sources may reach the core simultaneously. The traffic concentration will further cause the problems of packet loss and longer communication delay. Finally, it cannot tolerate any failure of the core.

Among the existing multicast tree construction algorithms, although KMB algorithm constructs a minimum cost multicast tree which achieves the best approximation ratio, it does not consider any delay constraint. KPP takes both tree cost and delay constraint into consideration, but the delay constraint in KPP is unnecessarily tight that it requires the delay between every two nodes (source or group member) is within the threshold. Moreover, both algorithms can only construct the multicast tree for static multicast groups. CRCDM can construct a dynamic minimum cost multicast tree which satisfies the delay constraint. However, since the CRCDM algorithm relies on a delay constrained minimum cost unicast routing algorithm, its time complexity is relatively high.

Our work in this chapter is motivated by the need of an efficient, flexible multicast architecture that can optimize the multicast tree while maintaining a relatively low overhead including both network bandwidth and the computing and storage resources in routers. In this chapter, we propose a service-centric approach for multicast communication, in which there are one or more powerful routers in each domain which handle most of multicast-related tasks. This approach can provide better efficiency and flexibility for two reasons: first, the powerful routers possess all the information on the network, such as network topology, link delay, link cost and group membership, thus it can adopt a more sophisticated algorithm to build an efficient multicast tree without increasing the network bandwidth usage; second, since the multicast tree constructing algorithm is run only on the powerful routers, it is convenient to modify the algorithm if the requirements of the multicast applications change while other routers in the domain do not need to know the change.

We also design a multicast routing protocol (SCMP) corresponding to the proposed multicast architecture. SCMP is an intra-domain multicast routing protocol which takes advantage of both the shared tree approach and the centralized processing fashion. A powerful router acts as the control unit and the root of the multicast tree for all the groups in the domain. A potential problem is that the traffic around the m-router is much more congested than the rest of the network. However, there is no traffic jam around the m-router for the following three reasons: first, the m-router is designed to process simultaneous many-to-many communication requests efficiently; second, the links connected to the m-router have more capacity and higher speed than the other links; third, SCMP can build a SPT tree for a specific

source on demand. The multicast tree is computed in the powerful router and distributed around the domain with as little overhead as possible by the self-routing packets. SCMP adopts the DCDM algorithm to construct a delay constrained dynamic multicast tree with minimum cost. The end-to-end delay between the source and any group member is bounded by a threshold. Group members can join or leave the group at any time during the multicast session. By taking advantage of two precalculated optimal unicast paths between any two nodes, it can achieve a good tradeoff between tree optimality and time complexity of the tree construction.

Although both DCDM and CRCDM construct delay constrained minimum cost multicast trees, they are different in many aspects. First, CRCDM is focused on the multicast tree rearrangement after nodes leave, while DCDM is focused on the non-rearrangeable multicast tree construction. Second, CRCDM relies on an existing delay constrained minimum delay unicast routing algorithm to find the graftpath, while DCDM finds the graftpath by itself. Third, the unicast routing algorithm used in CRCDM is a complex distributed algorithm which incurs long join latency and high computational load, while DCDM uses a simple centralized heuristic rule to find a sub-optimal graftpath in linear time complexity. Due to introducing the m-router, a centralized algorithm with less complexity is feasible and preferred.

6.2 The New Multicast Architecture

The proposed multicast architecture consists of three components: the specially designed powerful routers, the group and session management protocol and the multicast routing protocol. In this section, we first give an overview of the new multicast architecture, and then describe the three components separately.

6.2.1 Overview of the New Multicast Architecture

The proposed multicast architecture and protocol are based on the concept of *service-centric multicast routing*. Instead of treating each router equally as in existing multicast architectures, the new multicast architecture has two different types of multicast routers, which we call *master multicast router* (or *m-router*) and *intermediate multicast router* (or *i-router*). An i-router functions as an ordinary multicast router for forwarding multicast packets; while an m-router is responsible for more complex service-related tasks such as multicast session and group membership management, routing scheme control, transmission bandwidth management, and traffic scheduling and performs some service specific functions. The m-router integrates multiple routers, each of which can serve more than one multicast groups. Each m-router should be owned by an ISP (Internet Service Provider), who provides multicast services, so that the ISP centralizes most of service-related tasks on the m-router to

alleviate the burden on the Internet. An ISP may own more than one m-router in the Internet for serving its customers in different geographic regions. Fig. 6.2 shows the m-routers and i-routers in the Internet. For simplicity, we assume that one domain owns only one m-router in this chapter although our approach can be easily extended to multiple m-routers per domain. An i-router can adopt any multicast-capable switching fabric. For an m-router, we need to develop a special type of switching fabric. In general, we can consider an m-router has multiple input ports and multiple outputs ports and each of its input/output links has sufficiently high bandwidth.

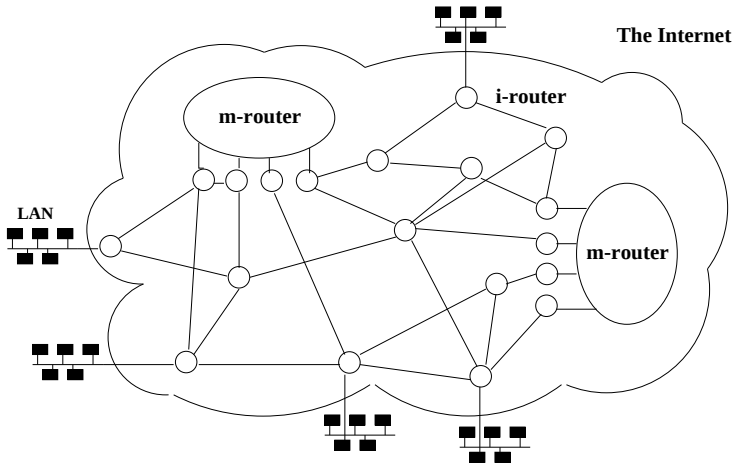


Figure 6.2: Illustration of m-routers and i-routers in the Internet.

A typical multicast communication is realized as follows. For each multicast group, the m-router dynamically assigns one of its output ports to the group, and a multicast tree rooted at the output port is built to reach members of the group via some i-routers which are non-root nodes of the multicast tree. The multicast tree is generated in the m-router in response to the JOIN/LEAVE message and then distributed around the domain. When transmitting a multicast packet, if the source router is the m-router itself, or it is on the multicast tree already, the packet is sent along the multicast tree; if the source router is a router which is not on the multicast tree, the source router first sends its packet to the m-router, then the m-router forwards the data along the multicast tree.

6.2.2 Design of the m-Router

The m-router plays a very important role in the multicast architecture because it handles most multicast related tasks and it is the root of the multicast trees. To avoid traffic jam around the m-router, it is required that the m-router is capable of

handling multiple multicast tasks simultaneously and forwarding heavy multicast traffic efficiently. Fig. 6.3 shows the sketch of the internal structure of an m-router, which has an $n \times n$ switching fabric.

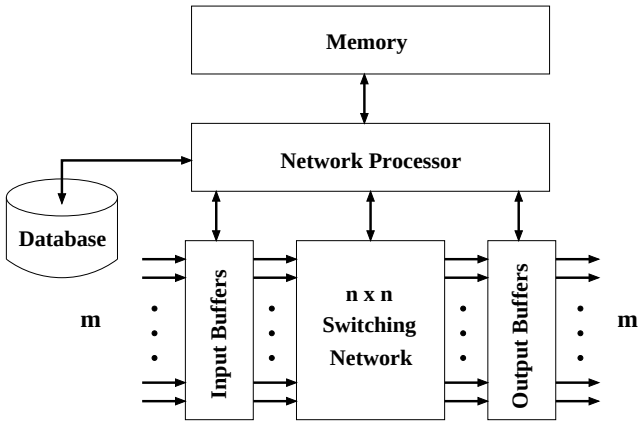


Figure 6.3: A sketch of the internal structure of an m-router.

Many tasks in the m-router, such as managing multicast group membership, generating multicast trees, scheduling, routing and transmission, are relatively independent, which can be performed in parallel. Thus, the m-router can adopt a multiprocessor or a cluster computer architecture. MMC Networks' NP3400 processor [72] and Motorola's C-port network processor [**?**] are examples of the routers with multiple processors.

Efficient hardware support from the underneath switching fabric with multicast capability is the key for the m-router to provide various multicast services and handle heavy multicast traffic. There has been a lot of work concerning multicast switching fabric designs in the literature [73]-[74]. Recall that in various multicast applications, for a multicast connection, a source may or may not belong to the multicast group. Also, there may be several multicast connections from different sources to the same multicast group, which can be referred to as *many-to-many communication*. We denote a many-to-many communication as (S, M), where S is the set of sources and M is the set of multicast group members. To support multiple such many-to-many communications in the Internet, the multicast switching fabric can be designed by adopting the concepts from the conference switching networks [75], but the switching fabric of the m-router is required to support more general communication patterns and make fully use of the multicast trees built in the Internet.

An illustration of the m-router switching fabric interconnecting with the Internet is shown in Fig. 6.4. By adopting the sandwich network structure [75, 76, 77, 78],

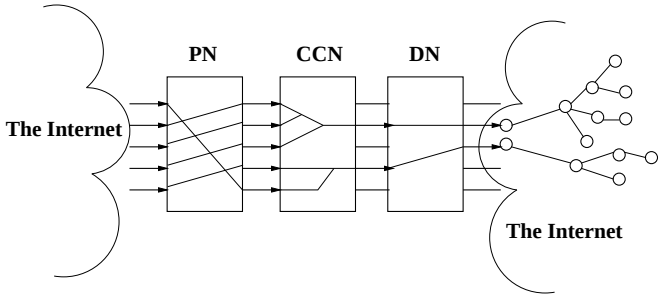


Figure 6.4: Illustration of an m-router switching fabric interconnected with the Internet.

the $n \times n$ switching fabric consists of three $n \times n$ subnetworks, permutation network (PN), connection component network (CCN) and distribution network (DN). Among them, the PN and the DN are permutation networks, which have the functions of keeping inputs/outputs in some order for the CCN and performing load-balance for the m-router in the Internet. The CCN realizes the connections of multiple sources by "merging" them in a reversed tree rooted at an output, which is then linked through the DN to the root of the corresponding multicast tree in the Internet. This way, the multiple sources can share one multicast tree via the connections in the CCN. However, as it should be, sources to different multicast groups are never be connected in the switching fabric. The CCN in the switching fabric of the m-router functions as connecting links coming from different sources to a link leading to the root of the multicast tree.

The network topology for the CCN is a modified version of a self-routing multistage network with low hardware cost, such as an omega network or an indirect binary cube network [74]. It has fan-in capability so that reversed trees can be established for different sources to share a corresponding multicast tree in the Internet. The fan-in capability can also be used for synthesizing multiple simultaneously arrived input data in a conference service. The data synthesis varies in conference applications with different service requirements. It is the responsibility of an ISP to define and implement the data synthesis features for its offered conference services. The CCN can provide an architectural support for the connectivity within a many-to-many communication and for the privacy among different many-to-many communications, using the lowest hardware cost and the minimum routing time in a switching network. That is, for any mutually disjoint many-to-many communications (S_1, M_1), (S_2, M_2), ..., (S_k, M_k) with $\sum_{i=1}^{k} |S_i| \leq n$ it is required to find k mutually disjoint reverse trees with the numbers of leaves being $|S_1|, |S_2|, \ldots, |S_k|$, respectively, in a self-routing multistage network.

6.2.3 Multicast Group and Session Management Protocol

We expect the new architecture would still maintain the user-transparency property for multicast group information, and be compatible to existing protocols. We use an existing protocol, Internet Group Management Protocol (IGMP) [2], to manage the process of a host computer joining or leaving a multicast group in the subnet. IGMP is used by hosts to register their dynamic multicast group membership. It is also used by routers to discover these group members. In IGMP, one of the routers connected to the same subnet is called the designated router (DR) which is selected among the routers in the same subnet to do some jobs on behalf of the subnet. In IGMP, the DR is responsible for sending Host Membership Query messages to discover which groups have members on their subnet. Hosts respond to a Query by generating Host Membership Reports, reporting each group they belong to on the network interface from which the Query was received. Thus, the group membership is transparent to end-users regardless of they are in the group or outside the group. However, unlike DVMRP and CBT where no routers have the complete group membership information and MOSPF where every router knows all the membership information, in the new architecture only the m-router knows all the group membership information. This feature of the new architecture makes more sense, because the group membership information should be easily accessible by the ISP for possible accounting and billing purposes, and the information should not be accessed by other parties.

The m-router acquires group membership information from i-routers as follows. Whenever an i-router finds out that a host in its resided subnet joins a new group, or all members in its subnetwork quit from an existing group, the i-router sends a unicast message to the m-router to inform the group membership change. Thus, the m-router collects group membership information passively and dynamically. Since the m-router is responsible for managing the multicast groups, it should be able to issue a multicast address for a new multicast group, revoke a multicast address from an abandoned multicast group, and publish the multicast addresses for existing multicast groups.

For managing multicast sessions, the m-router is responsible to start a new multicast session, to tear down an expired multicast session, and to check, track and record the multicast traffic in the corresponding multicast session. Since the lifetime of a multicast session depends on its multicast service requirements, multicast session management follows the service-related requirements and policies. The m-router also keeps track of all the membership on-off information for multicast scheduling/routing and for accounting/billing purposes.

Because the m-router is the sole entity for managing the multicast groups and multicast sessions, it should have abilities for outsiders to query proper information about multicast groups and sessions in the m-router. All the service-related infor-

mation will be kept in a database on the m-router. As SCMP is an intra-domain multicast routing protocol, which means the number of hosts is limited by the domain size, the information maintained in the m-router will not increase with the expanding Internet.

6.2.4 Multicast Routing Protocol (SCMP) - An Overview

Having described the multicast architecture, in this subsection, we give a brief overview of its corresponding multicast protocol SCMP. The details of this protocol will be presented in the next section.

We assume that the Internet consists of a number of autonomous systems or domains, where each domain is under the administrative control of a single entity. Besides a multicast routing protocol, each domain also runs a unicast routing protocol. SCMP is an intra-domain multicast protocol that constructs the multicast tree within the domain in which a link state unicast routing protocol is running.

The multicast routing protocol for the proposed architecture is expected to satisfy the following requirements:

- Allow any sophisticated network-wide routing algorithms to be used for constructing multicast trees;
- The computing effort for multicast trees is centralized at the m-router, saving the computing resource of other routers;
- Multicast routing information is transmitted only through the i-routers on the multicast tree, and does not affect the rest of the Internet.

In SCMP, the m-router's IP address is known to all the routers in the domain in advance. This can be realized by putting the IP address of the m-router in every router's configuration file. After a router is notified by one host in its subnet that it wants to join or leave a group, the router sends a JOIN/LEAVE request message to the m-router indicating the group ID and the IP address of the router. The m-router keeps track of all the group members in the group. When the m-router receives such JOIN/LEAVE request message, it updates the multicast tree according to the change of the group membership. A network-wide routing algorithm is run on the m-router to generate a multicast tree for a given multicast group. This is achievable because the m-router has all the group membership and global network topology information.

The multicast tree is generated in the m-router based on the collected topology and membership information. After the multicast tree is generated in the m-router, it should be physically formed in the domain. The routers on the tree should update the routing table according to the generated multicast tree. SCMP uses a special type of packet, TREE packet, to accomplish this. Each TREE packet contains the

complete information about a subtree of the multicast tree. The m-router is responsible to generate the original TREE packets. The i-router receives a TREE packet which represents a subtree rooted at the i-router itself. After receiving the TREE packet, the i-router sets the routing table according to the information in the TREE packet, and sends a new TREE packet to each i-router which is the child of the i-router in the multicast tree. The procedure is performed recursively on the multicast tree until it reaches the leaf routers. The resulting multicast tree is a shared, bi-directional tree rooted at the m-router.

The reasons for running a network-wide routing algorithm at the m-router and passing the multicast tree information along the tree are the following. A routing algorithm that constructs a near optimal multicast tree usually has higher time complexity. Thus, it should be run only once network-wide, and should be run by the router with more computing power. It is not necessary to run the same algorithm for the same multicast tree repeatedly on many i-routers with less computing power. On the other hand, the packet size of the TREE packet is comparable to that of the packet containing the multicast membership information, thus it will not increase the message payload much. Besides, the information is transmitted only through the i-routers on the multicast tree, and all the routing-related operations are performed only once during a multicast session.

When a source sends a packet to a group, if the source is not on the tree, the multicast packet is encapsulated in a unicast packet and sent to the m-router first (Note that the IP address of the m-router is known to all the routers in advance). The m-router decapsulates the packet and forwards it along the tree as a multicast packet. If the source is on the tree, the packet can be forwarded along the tree directly because the tree is bi-directional. Fig. 6.5 shows an overview of the SCMP.

6.3 Multicast Routing Protocol (SCMP)

In this section, we give the detailed descriptions of the new multicast routing protocol. We first provide some terminologies.

6.3.1 Terminologies

Each router on the multicast tree has an *upstream* which is the parent router of the router. The root of the tree (m-router) has no *upstream*. Each router on the tree has a *downstream* which is a set of routers and interfaces. The routers in the *downstream* are the child routers of the router. The interfaces in the *downstream* are the interfaces of the router that are connected to the subnets in which there is at least one host belonging to the group. A multicast routing entry is a triple with

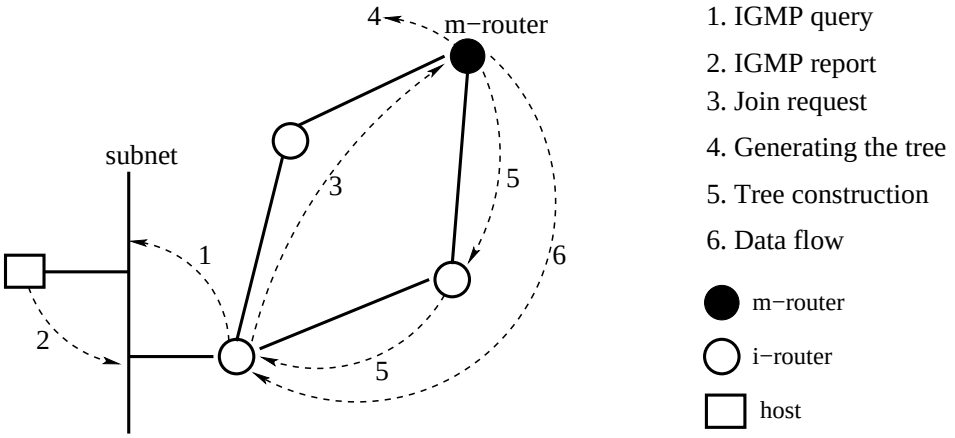


Figure 6.5: Overview of the SCMP protocol.

three fields: (group id, upstream, downstream). If a router is on the multicast tree of a group, "group id" is the identification of the group, "upstream" is the *upstream* of the router, and "downstream" is the *downstream* of the router. The multicast routing table is composed of one or more multicast routing entries.

6.3.2 Member Joining

When a host wants to join a group G, it sends an IGMP report message identifying the group id, gid, in response to the designated router DR's IGMP query message. When the DR receives an IGMP report for group gid, it checks whether it is on the multicast tree of group gid first. This is done by checking whether there exists a multicast routing entry whose "group id" is gid. If there exists such a multicast tree, it checks whether the interface connected to the host is included in the "downstream" of the multicast routing entry. If not, add it to the "downstream." If the interface is the first interface added to the "downstream," the DR will send a JOIN message to the m-router. Although the multicast tree does not need to be updated, the m-router needs this information for possible accounting and billing purposes.

If the DR is not on the multicast tree, it sends a JOIN message to the m-router indicating the gid and the IP address of the DR. At the same time, the interface from which the IGMP report message is received is marked so that it will be added to the "downstream" of the multicast routing entry which will be set up when the DR receives the TREE packet later. A timer is set and the DR will resend the JOIN message if the TREE packet is not received before the timer expires.

The pseudo-code of the member joining procedure is shown in Table 6.1.

Table 6.1: Member Joining Procedure

input: group id gid, interface inf.
output: JOIN message.
algorithm:
if there exists a multicast routing entry whose group id field is gid
  if inf is not in the multicast routing entry
    add inf to downstream of the routing entry;
  if inf is the only interface element in downstream
    send JOIN message to m-router;
else
  store (gid, inf) for creating the multicast routing entry in the future;
  send JOIN message to m-router;
end

6.3.3 Member Leaving

When the last group member of a subnet sends an IGMP leave report to the DR, the DR removes the interface from the "downstream" of the routing entry first. After that, the DR checks whether it becomes the leaf node of the multicast tree. A router is a leaf node of the multicast tree when the *downstream* of the router is null. If the DR is not a leaf node, there are two cases: (1) There is at least one interface element in the *downstream*. In this case, no action is needed; (2) All the elements in the *downstream* are routers. In this case, although the multicast tree remains the same, the DR should send a LEAVE message to the m-router for possible accounting and billing purposes.

If the DR is a leaf node after receiving the IGMP leave report, in addition to sending a LEAVE message to the m-router, it also sends a PRUNE message to the *upstream* router so that the *upstream* router will no longer forward the multicast packet to it. Similarly, if the *upstream* router finds that itself is a leaf node, it triggers another PRUNE message to its *upstream* router. This PRUNE message will continue until it reaches a non-leaf router. If the DR receives data packet after it leaves the group, it triggers a retransmission of the LEAVE message and the PRUNE message.

The pseudo-code of the member leaving procedure is shown in Table 6.2.

6.3.4 Constructing the Multicast Tree at the m-Router

The multicast tree is constructed in the m-router before it is physically formed in the domain. As discussed earlier, whenever a host wants to join or leave a group, a JOIN or LEAVE message will be sent to the m-router and then the m-router updates

Table 6.2: Member Leaving Procedure

input: group id gid, interface inf.
output: LEAVE message and PRUNE message.
algorithm:
remove inf from downstream of the multicast routing entry;
if downstream becomes NULL
  send a PRUNE message to upstream router;
  send a LEAVE message to m-router;
else
  if all the elements in downstream are routers
    send a LEAVE message to m-router;
end

the multicast tree. SCMP adopts the DCDM algorithm to update the multicast tree. Next we describe the DCDM algorithm. We first give some terms used in the description.

For each pair of nodes in the network, there exists a path that has the least cost among all the paths connecting these two nodes, and we use P_{lc} to denote this path. Similarly, for each pair of nodes in the network, there exists a path that has the shortest delay among all the paths connecting these two nodes, and we use P_{sd} to denote this path. Once the network is given, we can precalculate these two paths for every pair of nodes and store them for future routing. The *unicast delay* between two nodes is the delay of path P_{sd}. The unicast delay of a group member is the unicast delay between the group member and the m-router which is denoted as ul. For any group member, there is a unique path on the tree connecting the group member to the m-router. The *multicast delay* of the group member is the delay of the unique path and is denoted as ml. The longest multicast delay of all group members is the *tree delay*. The upstream router is denotes as US and the downstream set is denoted as DS.

Problem formalization

We first formalize the problem considered in the DCDM algorithm. The problem is called Delay Constrained Dynamic Steiner Tree-Nonrearrangeable (DCDST-N) problem which adds delay constraint to the DST-N problem in [79].

A point-to-point communication network can be represented by an undirected graph $G(V, E)$, where V is the set of nodes and E is the set of edges. The nodes in V represent the routers in the network. The edges in E represent the communication links connecting the routers. Two positive real functions are defined on E:

- *link cost function* ($LC : E \rightarrow \Re^+$): link cost is determined by the utilization of the link. It denotes the cost to use the link. The higher the utilization, the higher the link cost.

- *link delay function* ($LD : E \rightarrow \Re^+$): link delay is defined as the sum of the perceived queueing delay, transmission delay and propagation delay over the link.

The two functions will not change with time. Suppose P is a path in G which is composed of links $e_i, i = 1, 2, \ldots, p$. The cost and the delay of the path $Cost(P)$ and $Delay(P)$ are defined as follows:

$$Cost(P) = \sum_{i=1}^{p} LC(e_i), \text{ and } Delay(P) = \sum_{i=1}^{p} LD(e_i)$$

In a multicast communication session, there is a source node $s \in V$ and a group member set $M \subseteq V$. s is the node corresponding to the *source* that generates the data. The nodes in M, called group member nodes, correspond to the *group member*s that receive the data. s may or may not be in M. A multicast tree T is a tree in graph G such that T spans $s \cup M$. We use T^v and T^e to represent the node set and edge set of T, respectively. The tree cost is defined by the sum of the link cost in the tree.

$$Cost(T) = \sum_{e \in T^e} LC(e)$$

The nodes in set $(T^v - s \cup M)$ are called relay nodes. They are used to relay data from the source node to group member nodes.

Let v be a node in set $(T^v - s)$, there must exist only one simple path connecting s to v such that all the links in the path are in T^e. We use $P^t(v)$ to represent this path.

A multicast request r_i is a pair $(v_i, \rho_i), v_i \in V, \rho_i \in \{add, remove\}$. Request r_i can be viewed as node v_i requesting to join ($\rho_i = add$) or leave ($\rho_i = remove$) the multicast session. R is a sequence of multicast requests $R = \{r_1, r_2, \ldots, r_i, \ldots, r_n\}$. Corresponding to R, there are a sequence of member sets M^r and a sequence of multicast trees T^r. $M^r = \{M_1, M_2, \ldots, M_i, \ldots, M_n\}$, where $M_i = \{v_k | \exists k, 1 \leq k \leq i, r_k = (v_k, add)$ and $\forall j, k < j \leq i, v_j \neq v_k\}$ and $T^r = \{T_1, T_2, \ldots, T_i, \ldots, T_n\}$, such that T_i spans $s \cup M_i$.

The DCDST-N problem can be described as follows: Given a graph G, a source node s, two link functions LC and LD, a threshold Δ and a multicast request sequence R, construct a sequence of multicast trees T^r such that
1. $Delay(P^t(v)) \leq \Delta, \forall v \in M_i$;
2. The tree cost is minimum among all the possible trees;
3. if $\rho_i = add, T_{i-1} \subseteq T_i$; if $\rho_i = remove, T_i \subseteq T_{i-1}$.

Adding a group member node to the multicast tree

A host can join the multicast session anytime when it wants to receive data from the source by issuing a multicast request r_i in which $\rho_i = add$. The initial multicast tree includes only the source node s, i.e. $T_0^v = \{s\}$ and $T_0^e = \phi$. Suppose a multicast tree T_k is formed after k multicast requests have been processed, where T_k^v is the node set and T_k^e is the edge set. Now node v wants to join the multicast session. T_{k+1} is the multicast tree after node v joins. There are two cases:

Case 1: $v \in T_k^v$. In this case, node v is already on tree T_k. The multicast tree remains the same, that is, $T_{k+1} = T_k$ and $M_{k+1} = M_k \cup \{v\}$.

Case 2: $v \notin T_k^v$. In this case, node v is not on tree T_k. Suppose the size of T_k^v is l. For each node in T_k^v, there are two precalculated paths connecting v to this node: P_{lc} and P_{sd}. Thus there are totally $2(|V| - 1)$ paths connecting v to other nodes. These paths are sorted in an ascending order of their cost. The path list is checked from the beginning one by one until a path which connects v to the current tree and satisfies the delay constraint is found. Suppose the path found is P, which connects node v to node g on the tree. g is called the $graftnode$, and P is called the $graftpath$. P is then added to the tree, that is, $T_{k+1} = T_k \cup P$.

The pseudo-code of the procedure for adding a member node to the tree is given in Table 6.3.

Fig. 6.6 gives an example of the multicast tree construction by using the DCDM algorithm. Fig. 6.6(a) is the network topology. The numbers on the link represent (*link delay*, *link cost*). Node 0 is the m-router. Nodes 4, 3 and 5 represent three group member nodes g_1, g_2 and g_3, respectively. Suppose g_1 is the first group member to be added. The algorithm finds a shortest delay path connecting the m-router and g_1, which is $0 \rightarrow 1 \rightarrow 4$, and the tree delay is $3 + 9 = 12$. Now g_2 wants to join the group. The unicast delay of g_2 is 2 which is less than the current tree delay 12. Thus, there are two nodes for g_2 to graft on, node 0 and node 1. If node 0 is chosen, the multicast delay of g_2 is 2 and the tree cost is increased by 6. If node 1 is chosen, the multicast delay of g_2 is $3 + 3 + 4 = 10$, the tree cost is increased by 3. Thus, choosing node 1 will not increase the current tree delay while the tree cost is minimized at the same time. The final tree is $0 \rightarrow 1 \rightarrow 4$ and $1 \rightarrow 2 \rightarrow 3$, as shown by the solid lines in Fig. 6.6(b).

One issue we need to deal with in the multicast tree construction is to eliminate loops. Suppose g_3 wants to join the group after g_1 and g_2 join the group. The unicast delay of g_3 is $4 + 7 = 11$, which is less than the current tree delay 12. If node 2 is chosen as the graft node, the multicast delay will be $3 + 3 + 7 = 13$, which is greater than 12. Thus, the graft node should be node 0. As shown in Fig. 6.6(c), after path $0 \rightarrow 2 \rightarrow 5$ is added to the tree, a loop $0 \rightarrow 1 \rightarrow 2 \rightarrow 0$ is formed.

In order to remove the loop, before the $graftpath$ is added to the tree, every node in the $graftpath$ must be checked whether it is already in T_k^v. If it is in the

Table 6.3: Procedure of Adding a Member Node

input: G, v and T_k.
output: T_{k+1}.
algorithm:
  if $v \in T_k^v$
    $M_{i+1} = M_i + \{v\}$
  else
    $mincost = MAX$;
    //MAX is the largest number in the system
    for each node t in T_k^v
      if $Delay(P_{lc}(t,v)) + Delay(P^t(t)) \leq \Delta$
        if $Cost(P_{lc}(t,v)) < mincost$
          $mincost = Cost(P_{lc}(t,v))$;
          $graftpath = P_{lc}(t,v)$;
          $graftnode = t$;
      else if $Delay(P_{sd}(t,v)) + Delay(P^t(t)) \leq \Delta$
        if $Cost(P_{sd}(t,v)) < mincost$
          $mincost = Cost(P_{sd}(t,v))$;
          $graftpath = P_{sd}(t,v)$;
          $graftnode = t$;
    $T_{k+1} = T_k \cup graftpath$
end

tree, prune the tree as if it is a leaf node and update the P^t path for all the nodes in the downstream set of the node. The only exception is that the upstream of the node is in the $graftpath$. The pseudo-code of the procedure is given in Table 6.4. In the example, the algorithm prunes the tree upstream from node 2 until it reaches node 1. The final tree is $0 \rightarrow 1 \rightarrow 4$, $0 \rightarrow 2 \rightarrow 5$ and $2 \rightarrow 3$, as shown by the solid lines in Fig. 6.6(d).

We can also see that the loop removing procedure will not break the delay constraint. Consider the example in Fig. 6.6(c). Since node 0 is the $graftnode$, we have

$$Delay(0,2) + Delay(2,5) \leq \Delta \tag{6.1}$$

$$Delay(0,1) + Delay(1,2) + Delay(2,5) > \Delta \tag{6.2}$$

If (6.2) is false, the algorithm will choose node 2 as the $graftnode$. From (6.1) and (6.2), we have

$$Delay(0,2) < Delay(0,1) + Delay(1,2) \tag{6.3}$$

Since links $(0,1)$, $(1,2)$ and $(2,3)$ belong to T_k, we have

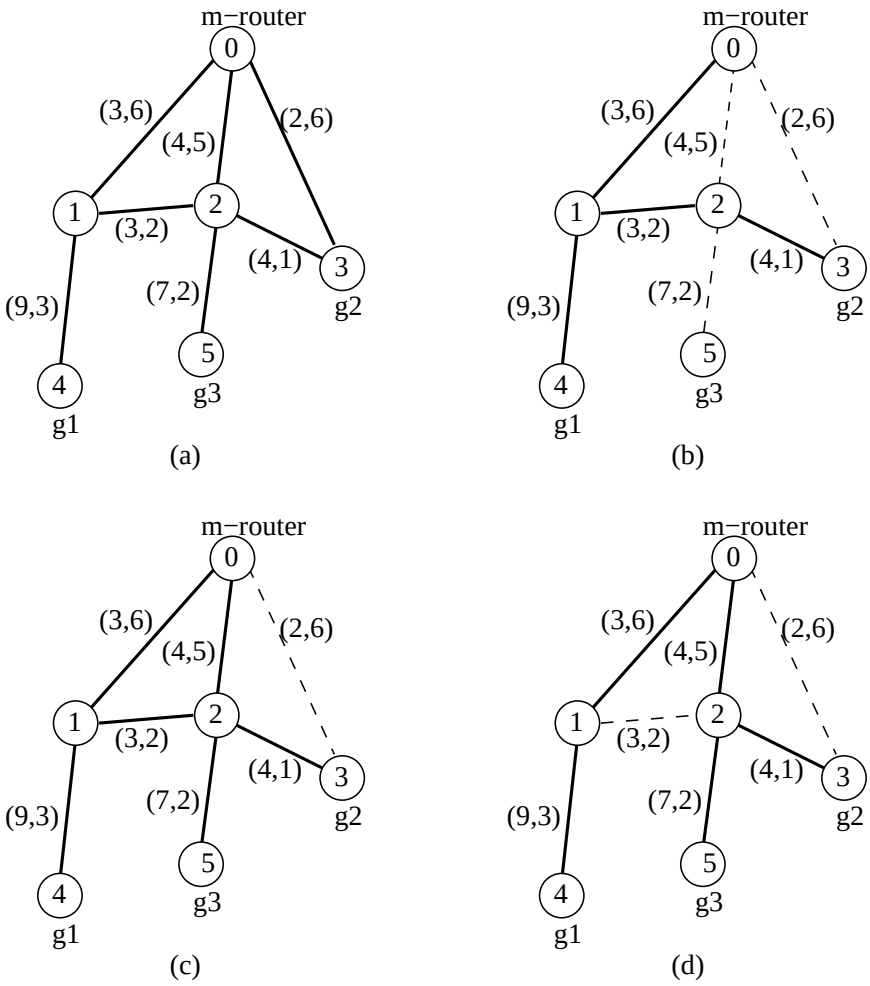


Figure 6.6: Example of using the DCDM algorithm. (a) Network topology; (b) Multicast tree after g_1 and g_2 are added; (c) A loop is formed after g_3 is added; (d) Multicast tree after g_3 is added.

$$Delay(0,1) + Delay(1,2) + Delay(2,3) \leq \Delta \tag{6.4}$$

From (6.3) and (6.4), we have

$$Delay(0,2) + Delay(2,3) \leq \Delta$$

Deleting a group member node from the multicast tree

When a host quits the multicast session, there are also two cases:

Table 6.4: Loop Removing Procedure

input: G, v, T_k, $graftpath$ and $graftnode$.
output: T_{k+1}.
algorithm:
  //Assume $graftpath = (g_0, g_1, \ldots, g_j)$
  //where $g_0 = graftnode, g_j = v$
  $T_{k+1} = T_k$;
  for each node $g_i (i > 0)$ on path $graftpath$
    if $g_i \in T_k^v$
      $\delta = Delay(P^t(g_i)) - Delay(P^t(g_{i-1})) - Delay(g_{i-1}, g_i)$;
      for each node d in $DS(g_i)$
        $Delay(P^t(d)) = Delay(P^t(d)) + \delta$;
      if $US(g_i) \neq g_{i-1}$
        Prune(g_i);
        //Function Prune() is used to prune the tree
        //which will be discussed in the next section.
    $T_{k+1} = T_{k+1} + link(g_{i-1}, g_i)$;
end

Case 1: The host is a leaf node of the multicast tree. In this case, prune the tree towards the upstream of the host recursively until it meets a branch or a node which is in M_k.

Case 2: The host is not a leaf node of the multicast tree. Then it is simply deleted from M_k, and $T_{k+1} = T_k$.

Table 6.5 gives the pseudo-code for the member deleting procedure.

Table 6.5: Procedure of Deleting a Member Node

input: G, v and T_k.
output: T_{k+1}.
algorithm:
  $T_{k+1} = T_k$;
  if v has no child
    $m = v$;
    while $US(m) \notin M_k \cap US(m)$ has the only child m
      $T_{k+1} = T_{k+1} - link(m, US(m))$;
      $m = US(m)$;
    $T_{k+1} = T_{k+1} - link(m, US(m))$;
  $M_{k+1} = M_k - v$;
end

Correctness and performance of the algorithm

We have the following lemma concerning the existence of the multicast tree.

Lemma 1 *A delay constrained multicast tree exists if and only if for each node* $m \in M$, $Delay(P_{sd}(s,m)) \leq \Delta$.

Proof: If a delay constrained multicast tree exists, there must be at least one path from source node s to each group member node m whose delay is less than Δ. Since the delay of $P_{sd}(s,m)$ has the shortest delay among all the paths connecting s to m, it must be less than Δ. On the other hand, if for all $m \in M$, $P_{sd}(s,m)$ is less than Δ, then the tree consisting of these shortest delay paths is a delay constrained multicast tree. ■

Theorem 6 *The DCDM algorithm finds a delay constrained multicast tree if and only if a delay constrained multicast tree exists.*

Proof: First, if the algorithm finds a delay constrained multicast tree, clearly, a delay constrained multicast tree exists. On the other hand, if a delay constrained multicast tree exists, from Lemma 1 we have $Delay(P_{sd}(s,m)) \leq \Delta, \forall m \in M$. When a new group member node v joins the group, it will check $2l$ paths which include $P_{sd}(s,m)$. Therefore, it can always find a path satisfying the delay constraint, that is, the algorithm can always find a delay constrained multicast tree. ■

The DCDM algorithm is easy to implement and has low time complexity. Its most time-consuming part is to compute the two paths P_{lc} and P_{sl} between each pair of nodes. This can be preprocessed off-line after the topology is given. For each newly joining node, the on-line time complexity is $O(|V| + |T^v|)$ where $|V|$ is the number of nodes in the network and $|T^v|$ is the number of nodes in the old tree. The following theorem summarizes this result.

Theorem 7 *The time complexity of the DCDM algorithm is* $O(|V| + |T^v|)$*, where* $|V|$ *is the number of nodes in graph* G *and* $|T^v|$ *is the number of nodes in* T^v.

Proof: The two paths, P_{lc} and P_{sd}, between every pair of nodes are calculated by Dijkstra algorithm in advance. So there are totally $(|V|-1)^2$ paths calculated off-line. For each node, the $2(|V|-1)$ paths are sorted based on their link cost. DCDM spends most of the time processing multicast requests. For a multicast request where $\rho_i = remove$, the time to prune the multicast tree is $O(|T^v|)$. For a multicast request where $\rho_i = add$, DCDM needs to do three things: finding the $graftnode$; adding the $graftpath$; and removing the loop if necessary. The time to find the $graftnode$ is $O(|V|)$ as $2(|V|-1)$ paths are checked in order. Since the longest length of the $graftpath$ is less than $|V|$, the time to add the $graftpath$ to the multicast tree is $O(|V|)$. Removing the loop includes updating path P^t for

the nodes in the downstream set and pruning the upstream node recursively. Since each node is either updated or pruned at most once, the time to remove the possible loop is $O(|T^v|)$. Thus, the total time of the DCDM algorithm is $O(|V|+|T^v|)$. ■

When the m-router receives a JOIN message, it takes one of the following two actions.

1. If the source node is on the multicast tree already. The m-router simply puts a tag on that source node indicating that the node is a group member.

2. If the source node is not on the multicast tree, the m-router adds an appropriate $graftpath$ to the tree which connects the source node to the old tree as shown in the DCDM algorithm above.

When the m-router receives a LEAVE message, it removes the node from the group. If the node becomes a leaf node, the tree is pruned towards the upstream router until it reaches a group member or a node that has more than one downstream routers. This guarantees that the tree in the m-router is consistent with the actual tree in the network. The actual prune operation is accomplished by the leaving member sending the PRUNE message upstream hop by hop.

Until now we assume SCMP builds a tree rooted at the m-router. Actually, to alleviate the burden of the m-router, SCMP can build an SPT tree upon the request of the source. In this case, the m-router is still responsible for collecting the JOIN/LEAVE messages and computing the multicast tree. But the tree is distributed from the source node instead of the m-router which will be discussed in the following subsection.

6.3.5 Forming the Multicast Tree in the Network

After the m-router constructs the multicast tree, it should distribute the tree in the domain so that the i-routers on the tree can update their routing tables and forward the multicast data packets correctly. This is completed by the multicast tree forming process.

In order to minimize the protocol overhead, we adopt the self-routing scheme proposed in [80] for a self-routing multicast network in which multicast routing is realized by the tag attached to the packet. For a random topology network, we can similarly use such self-routing packets to construct the multicast tree. We call this type of packet TREE packet.

A TREE packet includes all the information about a tree. The length of the TREE packet is a variable which depends on the size of the tree. The format of the TREE packet is described in Table 6.6.

Table 6.6: Format of TREE Packet

Number of the downstream routers
IP address of the downstream router 1
Length of subpacket 1
Subpacket 1
IP address of the downstream router 2
Length of subpacket 2
Subpacket 2
...

The format of the subpacket is the same as the format of TREE packet. This recursive packet structure reflects the recursive structure of the tree. "Subpacket i" includes all the information about the tree rooted at "downstream router i."

The TREE packet is a self-routing packet, which means that the routers forward the TREE packet according to the information in the TREE packet itself. When a router receives a TREE packet from its upstream router, it updates its routing table based on the information in the packet and sends new TREE packets to its downstream routers if any. The first TREE packet is generated in the m-router based on the multicast tree. Since the m-router is the root of the multicast tree, each downstream router of the m-router is the root of a subtree. The m-router builds a TREE packet for each subtree. Then the m-router sends these TREE packets to the corresponding downstream routers and the downstream routers are added to the "downstream" of the routing entries. When an i-router receives a TREE packet, the TREE packet should include the information of the subtree which is rooted at the i-router itself. The "upstream" of the route entry is set to be the router from which the TREE packet is received. The TREE packet is split into several smaller TREE packets each of which represents a subtree rooted at one of the downstream routers. Then each of the smaller TREE packets is sent to the corresponding downstream router after the downstream router is added to "downstream" of the route entry. After all the TREE packets reach the leaf routers, the tree is formed in the network.

The pseudo-code of the TREE packet processing algorithm in i-routers is showed in Table 6.7.

Now let's look at an example. Suppose the m-router has three downstream routers. Fig. 6.7 shows the multicast subtree rooted at node 2.

Here we use node id to represent the address of the router. The m-router generates three TREE packets for its three downstream routers respectively. The TREE packet for node 2 is $(3; 4, 1, 0; 5, 7, 2, 7, 1, 0, 8, 1, 0; 6, 4, 1, 9, 1, 0)$, where 3 means node 2 has three downstream routers; $(4, 1, 0)$ means the first downstream router is node 4, the length of subpacket representing the subtree rooted at node

Table 6.7: TREE Packet Processing Algorithm

input: TREE packet.
output: updated routing entry and new TREE packets.
algorithm:
$upstream$ = IP address of the router from which the TREE packet is received;
for each downstream router ds in the TREE packet
  add ds to the $downstream$ of the routing entry;
  extract the subpacket corresponding to ds from the TREE packet;
  send the subpacket as a new TREE packet to the ds;
end

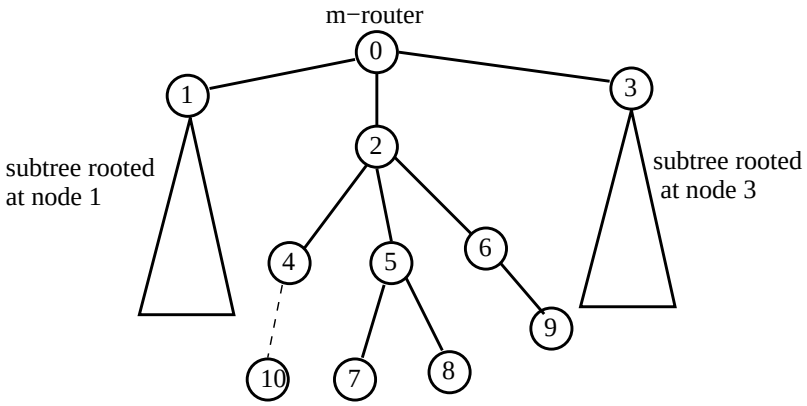


Figure 6.7: Forming the multicast tree using TREE and BRANCH packets.

4 is 1, the subpacket is (0); Similarly, the second downstream router is 5, the length of subpacket representing the subtree rooted at node 5 is 7, the subpacket is $(2, 7, 1, 0, 8, 1, 0)$, and so on. When node 2 receives this TREE packet, it sets the "upstream" to the m-router and splits this TREE packet into three TREE packets: $(0), (2, 7, 1, 0, 8, 1, 0)$ and $(1, 9, 1, 0)$. Then sends them to nodes 4, 5 and 6 and adds nodes 4, 5 and 6 into the "downstream" of the routing entry. The multicast routing entry in node 2 after processing the TREE packet is

gid	1	4, 5, 6

. When receiving the TREE packets, node 4 will add the interface marked after receiving the IGMP report message into the "downstream" of the routing entry, then it sets "upstream" to node 2; node 5 will continue to split the TREE packet into two TREE packets which are both (0) after updating the routing entry; node 6 will send TREE packet (0) to node 9 after updating the routing entry. Finally, after nodes 7, 8 and 9 receive the TREE packets and update the routing entries, the tree forming process is completed.

Clearly, whenever a new router joins a group, a TREE packet will be triggered in the m-router if the tree is changed. If the change is small, using a TREE packet containing the whole tree structure is too expensive. Thus, we use a new type of packet, called BRANCH packet, to update a minor change. A BRANCH packet consists of a branch of the tree from the m-router to the new group member.

The format of the BRANCH packet is simple. It is composed of a sequence of routers that are on the path from the current router to the new group member in order. When an i-router receives a BRANCH packet, it deletes itself from the head of the path, sets the "upstream" as the router from which the packet is received, adds the next router on the path to the "downstream" of the routing entry and forwards it to the downstream router. For example, in Fig. 6.7, if node 10 wants to join the group, the m-router generates a BRANCH packet $(2, 4, 10)$ and sends it to node 2. Node 2 then updates the routing entry, deletes itself from the packet and sends a BRANCH packet $(4, 10)$ to node 4, ..., until node 10 receives the BRANCH packet (10), it adds the marked interface to the "downstream" and sets the "upstream" to node 4.

In the case of building an SPT tree, the original TREE packet is generated by the m-router. Then it will be sent to the source node where it is distributed around the domain by a similar way discussed above.

Handling unstableness of the network

When a link fails or a router crashes, the tree should be able to recover in time. In some existing multicast protocols, a downstream router sends a JOIN message to its upstream router periodically. Once the upstream router does not receive the message before timeout, it deletes the downstream router from the downstream of the routing entry. Then the leaf router will try to find another path to graft on the tree. This method may generate a lot of JOIN messages. In our protocol, the routing entry is soft-state in the sense that it should be refreshed periodically or be removed due to timeout. There is a timer attached to each routing entry. Only receiving a packet can refresh the timer. When it times out, the corresponding routing entry is deleted automatically. When an i-router on the multicast tree crashes, all the downstream routers will not receive packets for a while. After the m-router detects the crash, it determines whether a TREE packet or a BRANCH packet is needed and recovers the tree. The routing entry of the old tree will be deleted due to timeout. One possible problem is that what we should do if there is no packet for a long time. In this case, the m-router issues a dummy packet to refresh the forwarding state in the i-routers on the tree.

6.3.6 Forwarding Multicast Packets

The multicast tree constructed is a bi-directional tree. It means that the multicast packet can be transmitted in both directions on the tree. The multicast packet is not only forwarded to the downstream routers, but also forwarded to the upstream router when necessary. Since we assume the link is symmetric, the delay of a path in both directions is the same, thus the optimality of the multicast tree will not be impaired.

All the i-routers are configured to know the IP address of the m-router. When a source router has a multicast packet to send, it first checks whether it is on the multicast tree. If it is on the multicast tree, it sends the packet to the upstream router and all the downstream routers. If the source is not on the tree, it encapsulates the packet into a unicast packet and sends it to the m-router. The m-router decapsulates the data, puts it in a multicast packet and forwards the packet according to the routing entry.

Table 6.8: Multicast Packet Forwarding Algorithm

input: multicast packet.
output: forwarded packet.
algorithm:
$incoming$ = the router from which the multicast packet is received;
if $incoming$ is the upstream router
  forward the packet to all the downstream routers;
else
  if $incoming$ is one of the downstream routers
    forward the packet to upstream router;
    forward the packet to all the downstream routers
    except $incoming$;
  else
    drop the packet;
end

When forwarding the multicast packet, the i-router considers both the upstream router and the set of downstream routers as a single set, say, F. If an i-router receives a packet, it checks whether the packet comes from a router in set F. If yes, then forwards the packet to other routes in set F. If no, it drops the packet. The packet forwarding algorithm is described in Table 6.8.

Let's look at the example in Fig. 6.8, where the solid lines represent the multicast tree and the green arrows represent the data flow. Suppose node 1 has a packet to send as shown in Fig. 6.8(a). Since node 1 is not on the tree, it will encapsulate the data into a unicast packet and send it to node 0. Node 0 then decapsulates the packet and forwards it to nodes 2 and 3. Node 3 forwards it to nodes 4 and 5. On

the other hand, in Fig. 6.8(b), if node 3 has a packet to send, since it is on the tree already, it will send the packet to nodes 0, 4 and 5 at the same time. Then node 0 forwards it to node 2 and all the group members receive the packet.

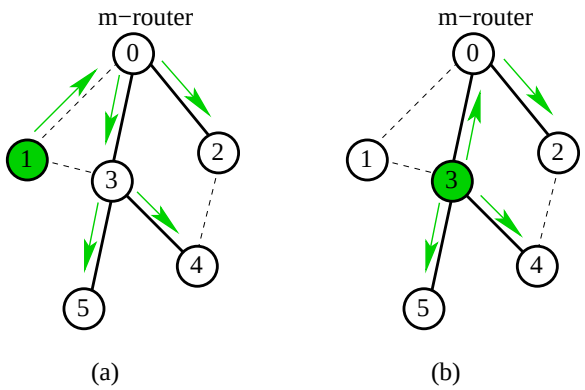


Figure 6.8: Forwarding the multicast packet along a bi-directional multicast tree.

6.4 Performance Evaluations

We have implemented the newly proposed multicast protocol along with three existing protocols on the NS-2 [33] simulator and conducted extensive simulations to evaluate the protocol. In this section, we compare the multicast trees constructed by our protocol and other protocols, and the maximum end-to-end delay, protocol overhead and data overhead of the protocols.

6.4.1 Multicast Trees

A random network topology is generated with each link assigned a random link delay and a random link cost. A set of group members are randomly picked from the nodes in the topology. We use different algorithms to construct the multicast tree for the same set of group members and compare the multicast tree afterwards. The comparison metrics are *tree delay* and *tree cost*.

We compare the DCDM algorithm with other four existing algorithms, KMB, DVMRP, MOSPF and CBT. CRCDM was not implemented due to its dependence on a high complexity unicast routing protocol. Since the multicast tree constructed by DVMRP or MOSPF is determined not only by the set of group members but also by the source node, we assume that the source node is the same node as the core in CBT. Under this assumption, the multicast trees constructed by these three algorithms (DVMRP, MOSPF and CBT) are identical because all of the trees are

composed of the shortest delay paths between the core/source and the group members. Therefore, in the simulation we only implemented CBT, KMB and DCDM.

The simulation model we used is similar to that used in [81]. Nodes in the graph are placed randomly in a rectangular coordinate grid by generating uniformly distributed values for their x and y coordinates. The size of the rectangular is $32,767$ by $32,767$. x and y are random integers between 0 and $32,767$. For every pair of nodes u and v, the probability that there exists an edge connecting u and v is $P(u,v) = \beta * e^{\frac{-d(u,v)}{\alpha L}}$, where $d(u,v)$ is the Manhattan distance between u and v. Let (x_u, y_u) be the x and y coordinates of node u and (x_v, y_v) be the x and y coordinates of node v, then $d(u,v) = |x_u - x_v| + |y_u - y_v|$. L is the maximum Manhattan distance between any two nodes, which is $2 * 32,767$. α and β are two tunable parameters. Increasing α increases the number of edges between far away nodes and increasing β increases the degree of each node. The link cost of an edge is equal to the Manhattan distance between the two nodes, and the link delay of an edge is equal to an uniformly distributed random variable between 0 and the link cost of the edge.

In our simulations, the total number of the nodes in the topology is 100, the group size increases from 10 to 90 by 10 at each step, $\alpha = 0.25$, and $\beta = 0.2$. Each simulation was run 10 times with different random generator seeds. We plot the figure based on the average of the 10 values obtained from the 10 simulations.

Fig. **??** shows the simulation results. We set the delay constraint into three levels: tightest, moderate and loosest. The tightest level means that the delay constraint cannot be tighter, or there is no multicast tree satisfying the delay constraint. The loosest level means that all possible multicast trees can satisfy the delay constraint. Fig. 6.9(a), (b) and (c) show the tree delay comparison under the three levels respectively. Fig. 6.10(a), (b) and (c) show the tree cost comparison under the three levels respectively.

We first compare the tree delay. From the figures we can see that no matter what level the delay constraint is, the tree delay of DCDM is much shorter than that of KMB. As the group size increases, the tree delay of DCDM is relatively stable and increases a little, while the tree delay of KMB oscillates intensely. This is because KMB tries to minimize only the cost and does not consider the delay at all. CBT always achieves the shortest tree delay. When the delay constraint is in the tightest level, DCDM can achieve the same tree delay as CBT. When the delay constraint relaxes, DCDM will try to minimize the tree cost without violating the delay constraint. Therefore, the tree delay of DCDM is a little longer than that of CBT, but as will be seen shortly, its tree cost is much lower.

Now we compare the tree cost. We can see that as the group size increases, the tree cost of the three algorithms increases too. The tree cost of CBT is the highest, while the tree cost of KMB is the lowest. DCDM achieves the tree cost between

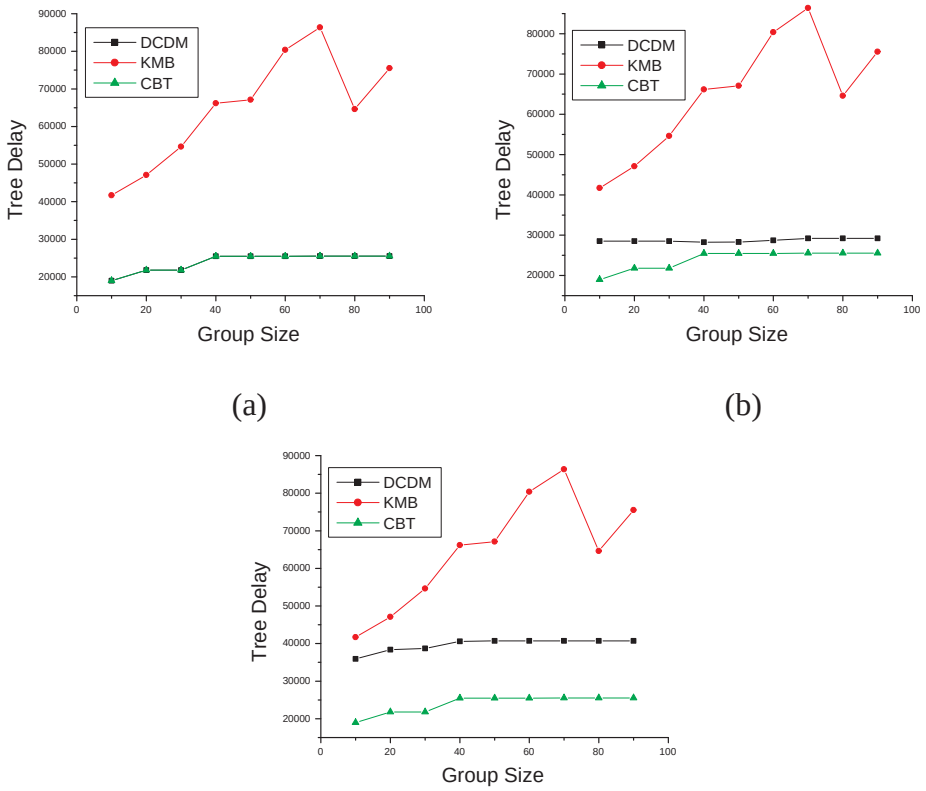


Figure 6.9: Tree delay comparison. (a), (b) and (c): delay constraint is tightest, moderate and loosest, respectively.

CBT and KMB, and it is closer to that of KMB. The differences between the tree cost of any two algorithms increases when the group size increases. When the delay constraint is looser, the gap between DCDM and KMB is obviously smaller than that when the delay constraint is tighter. When the delay constraint is in the loosest level and the group size is small, the tree costs of DCDM and KMB are almost the same.

In our simulations, we also change the location of the m-router to see how it affects the tree cost. We observe that since the set of group members and the join order of group members are changing, there is no such location of the m-router that it has the best performance under all conditions. However, we find that there are some heuristics for placing the m-router to achieve good performance in most cases:

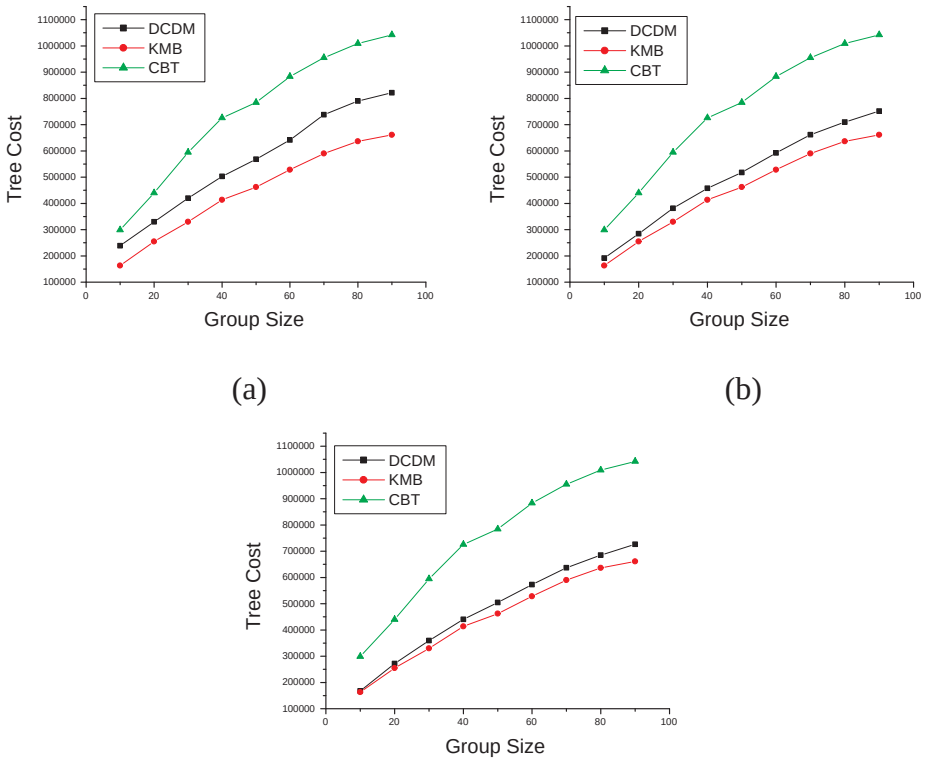


Figure 6.10: Tree cost comparison. (a), (b) and (c): delay constraint is tightest, moderate and loosest, respectively.

- *Rule 1:* For each node, calculate the average delay between the node and all the other nodes. Choose the node with less average delay.
- *Rule 2:* Choose the node with a larger node degree.
- *Rule 3:* Choose the node lying on the path whose delay is equal to the delay diameter of the graph.

6.4.2 Network-Wide Performance

Besides comparing the multicast trees, we implemented SCMP and other three protocols (DVMRP, MOSPF and CBT) on the NS-2 simulator to compare their network-wide performance. The following metrics are compared.

Data overhead: The network bandwidth used by data packets. A data packet going through one link contributes lc units to the data overhead, where lc is the link cost of the link.

Protocol overhead: The network bandwidth used by protocol packets. A protocol packet going through one link contributes lc units to the protocol overhead, where lc is the link cost of the link.

Maximum end-to-end delay: The maximum delay experienced by the packets from the source to the group members.

Three different network topologies are used for performance comparison. One is the ARPANET, and the other two are random topologies generated by GT-ITM software [27]. The network size of each random topology is 50 and the average node degrees of the two random topologies are 3 and 5 respectively. There is a source node sending one multicast packet per second. The group size varies from 8 to 40 and the group members are picked randomly. The total simulation time is 30 seconds.

Data overhead

We compare the data overhead of the four protocols in Fig. 6.11. In the figure, we can see that SCMP always has the lowest data overhead among all four protocols, and CBT and MOSPF have higher but relatively closer overhead to SCMP, while DVMRP has much higher data overhead. This is caused by the fact that DVMRP floods the packets frequently when it starts to construct the tree or the timer in a leaf router is expired. The data overhead of the other three protocols is close to each other. Of the three protocol, MOSPF has the most data overhead, and SCMP has the least data overhead. As the group size increases, the difference between SCMP and MOSPF and CBT increases too. This superiority is more obvious when the average node degree is around 3. As can be seen, the data overhead is strongly correlated to the multicast tree cost. The trends of the curves of the two metrics are very similar.

Protocol overhead

As Fig. 6.11 shows, the protocol overhead increases as the group size increases except for DVMRP. DVMRP shows an inverse ratio when the group size increases. If the X axis is extended longer enough, DVMRP would be the one with the least protocol overhead. Since DVMRP adopts “flooding and pruning” algorithm, the more the group members, the less the prune messages. It is a dense mode multicast protocol, which means that it is only suitable for the domain in which most nodes are group members. MOSPF has the steepest curve. In MOSPF, whenever a group

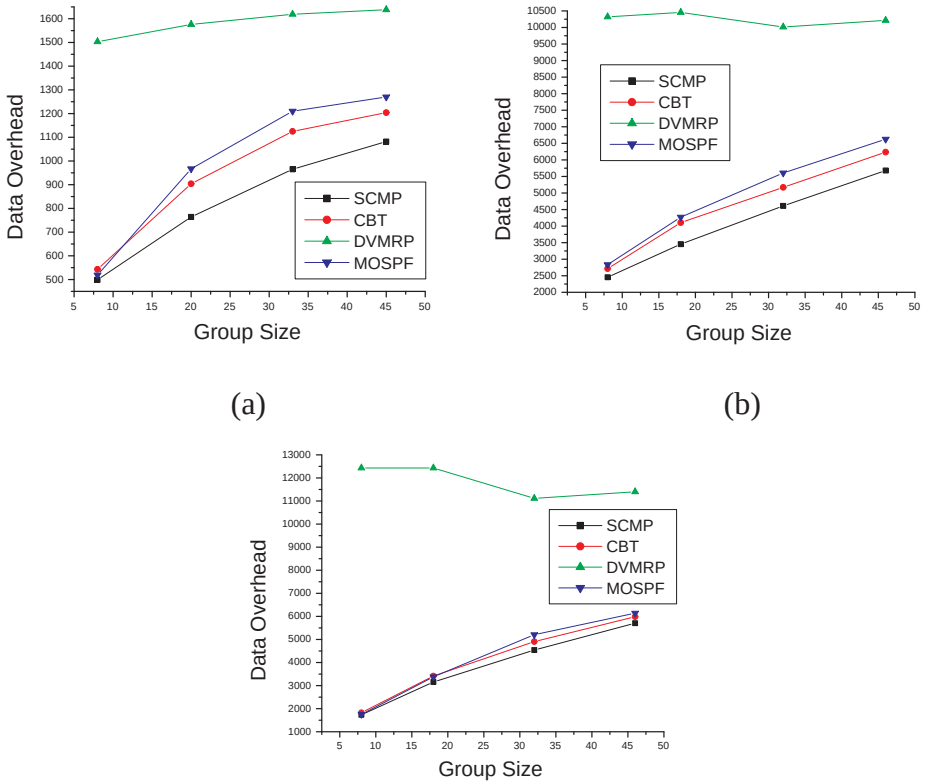


Figure 6.11: Group size versus data overhead. (a) ARPANET; (b) Random topology network with average node degree 3; (c) Random topology network with average node degree 5.

member wants to join or leave the group, the DR will flood a group-membership-LSA packet throughout the domain to make all the routers updated, which generates a great deal of protocol packets. SCMP and CBT have the lease protocol overhead. The difference between the two protocols is so small that we have to replace the Y axis with $\log Y$ to separate the two curves, which is shown in Fig. 6.12(b) and (c).

When the group size increases, the protocol overhead of SCMP is a little bit more than that of CBT. In our simulation, we did not simulate the core selection process of CBT which is a sophisticated and relatively open problem. However, whatever the selection mechanism CBT chooses, it will certainly induce some protocol overhead. When we do not take the core selection into consideration, CBT has a little less protocol overhead than SCMP. This is because CBT only needs to send an acknowledgement packet from the graft node to the newly joining node

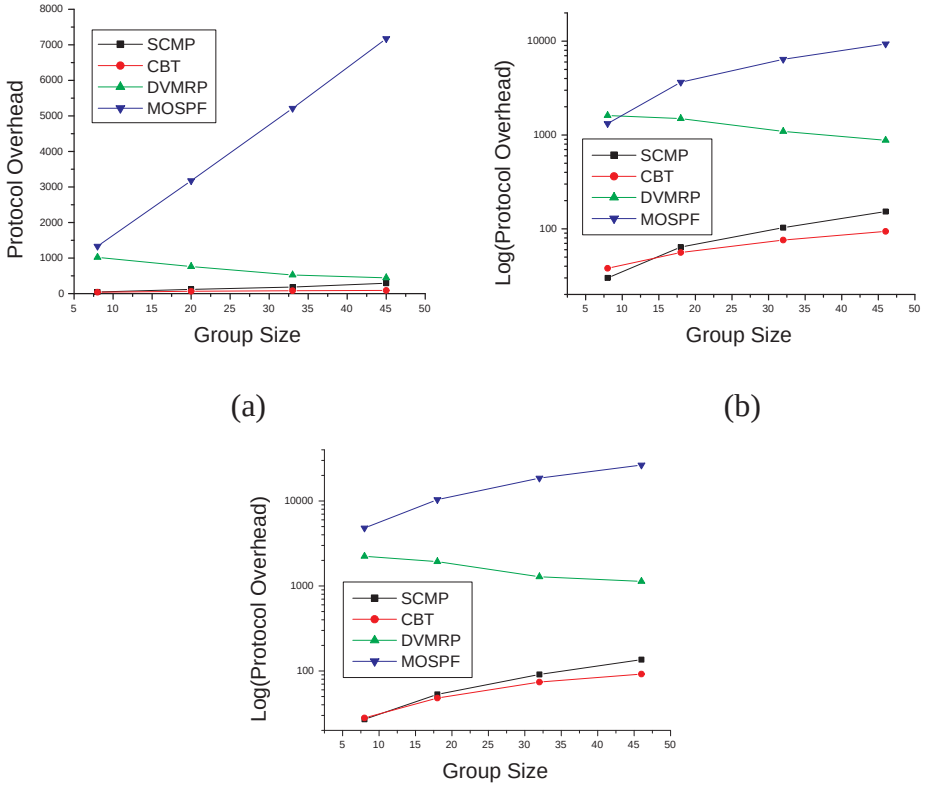


Figure 6.12: Group size versus protocol overhead. (a) ARPANET; (b) Random topology network with average node degree 3; (c) Random topology network with average node degree 5.

when performing the join operation, while SCMP always needs to send a BRANCH packet from the m-router all the way to the newly joining node.

Maximum end-to-end delay

We now consider the maximum end-to-end delay with different group sizes. Fig. 6.13 shows the maximum end-to-end delay of the four protocols for the three network topologies. We can see that the delay of SCMP and CBT is very close and is slightly longer than the SPT-based protocols. This is because that in SPT-based protocols, data packets are delivered from the source node to group members directly, while in SCMP or CBT, packets are sent to the core router first if the source node is not on the tree. We can also observe that the difference in delay becomes smaller

when the group size or the node degree increases. However, as we pointed out earlier, SPT-based protocols have the scalability and bandwidth wasting problems, which have been seen clearly in their data overhead and protocol overhead.

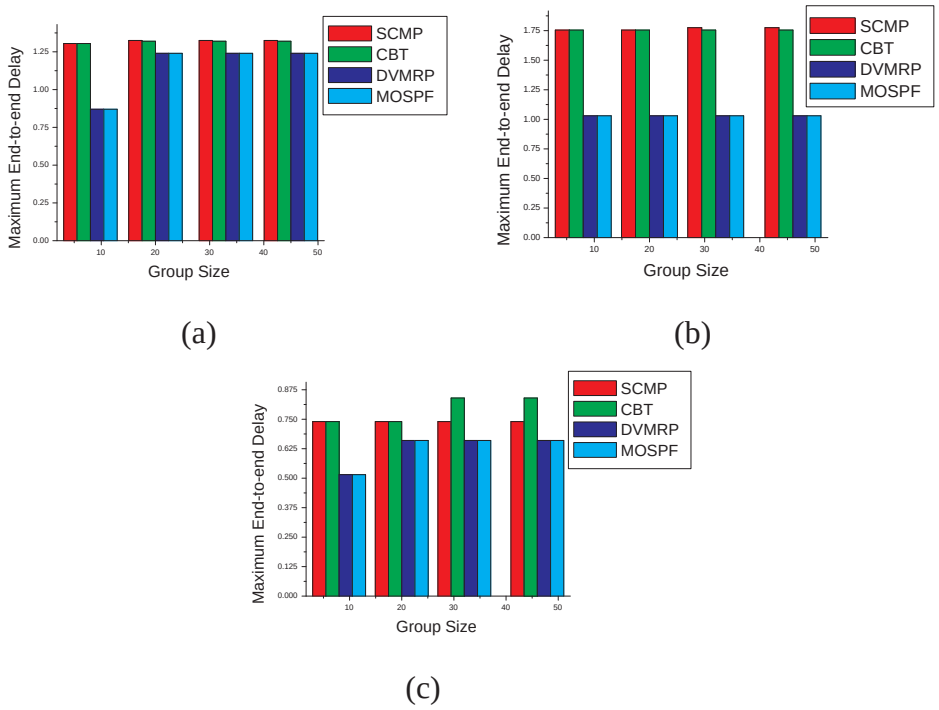


Figure 6.13: Group size versus maximum end-to-end delay in seconds. (a) ARPANET; (b) Random topology with average node degree 3; (c) Random topology network with average node degree 5.

6.5 Summary

In this chapter, we have proposed a service-centric multicast architecture and an efficient and flexible multicast routing protocol SCMP. Compared to existing multicast architectures, the new architecture has following advantages: (1) By concentrating most multicast routing and service-related tasks in the m-router, routing efforts on other routers can be greatly reduced and the bandwidth wasting in the rest of the Internet can be avoided; (2) The new architecture can adopt any sophisticated network-wide routing algorithms in the m-router to construct a near optimal multicast tree, which makes it easier to be adapted to various applications with different QoS requirements; (3) ST-based protocols suffer the traffic concentration problem

around the core, while the m-routers in the new architecture are specially designed powerful routers to efficiently handle heavy network traffic, which can greatly alleviate the problem; (4) Another common problem of ST-based protocols is the single core failure problem. In the new architecture, since the m-router is owned and administrated by the ISP, it is easy to install a hot standby system, in which there is a secondary m-router besides the primary m-router. The primary and the secondary m-routers run simultaneously. The data is mirrored to the secondary m-router in real time so that both m-routers have identical information. When the primary m-router fails, the secondary m-router will take over the job automatically. Our simulation results demonstrate that the new SCMP protocol outperforms other existing protocols. The DCDM algorithm can dynamically adjust the multicast tree such that the tree delay and tree cost are minimized at the same time. In particular, SCMP has the least amount of data overhead among the four protocols. The protocol overhead of DVMRP and MOSPF is much higher than SCMP and CBT. Although the protocol overhead of CBT is a little bit less than SCMP, this is partially because of the simplification of the core selection process of CBT in the simulation. As for tree delay, we can see that SCMP and CBT are very close and their delay is slightly longer than the SPT-based protocols due to the fact that they use shared multicast trees. However, as we pointed out earlier, SPT-based protocols have the scalability and bandwidth wasting problems, which can be clearly seen in their data and protocol overhead. Therefore, overall we believe the newly proposed SCMP protocol is a promising alternative for providing efficient and flexible multicast services over the Internet.

One potential problem is the feasibility of the new architecture. Generally speaking, little work is needed for the ISPs to upgrade their network to support SCMP, the process is reliable and the billing system is easy to manage. To deploy SCMP within a domain, for the hardware part, the ISPs need to connect an m-router into the network; for the software part, the software in the other routers should be upgraded. As there is little work done in the i-routers, a small software patch is sufficient for the software upgrade. This software upgrade can be done at once or gradually depending on the network size or the reliability requirements. In the case of gradually upgrade, only part of the i-routers run SCMP and they communicate through tunnels. The multicast tree is built on these i-routers only. By running BGMP [82] in the border routers, SCMP can cooperate with other multicast routing protocols in other domains to build a multicast tree across multiple domains. As the m-router contains all the information of multicast sessions and group members, it is easy for the ISPs to provide services with different requirements and charge them accordingly.

Chapter 7

A Peer-to-Peer Tree Based Reliable Multicast Protocol

Because the Internet provides best effort service only and the receivers may have different bandwidths and processing abilities, it is necessary to design a multicast transport layer to provide reliable multicast over the Internet. Like its unicast counterpart, Transmission Control Protocol (TCP), a reliable multicast protocol should possess the functions such as packet loss detection and recovery, ordering and flow control. However, having multiple receivers makes the reliable multicast vastly different and much more complex than TCP. First, in unicast, the receiver uses positive acknowledgements (ACK) or negative acknowledgements (NACK) to inform the sender its current status, while in multicast, many ACK/NACK messages produced by all the receivers will overwhelm the sender and congest the links around the sender, which is called "acknowledge implosion" [83]. Second, it is difficult to adapt the data transmission rate of the sender to the different data reception rates of the heterogenous receivers. Third, the wide range of requirements of the multicast applications makes it impossible to design a one-fit-all reliable multicast protocol [84]. On the other hand, most multicast applications are real-time applications, such as media streaming and audio/video conferencing, which have strict requirements on QoS, especially on delay jitter [85], and it is essential to minimize the retransmission delay when the data is lost in such applications.

There has been a lot of work on reliable multicast in the literature, see, for example, [86, 87, 88, 89]. In general, reliable multicast protocols are classified into sender-initiated, receiver-initiated and tree based protocols. In sender-initiated protocols, the sender maintains the states of all the receivers. The receivers send ACK messages to the sender indicating the correct reception of packets. A missing ACK message is caused by either a lost data packet, a lost ACK or a crash of the receiver. The sender retransmits the lost packet by multicast when it does not receive the ACK messages before timeout. Sometimes NACK messages are used to speed up

the retransmission. An example for a sender-initiated protocol is Xpress Transport Protocol (XTP) [90]. In receiver-initiated protocols, it is the receivers' responsibility to detect packet loss. When a receiver detect a wrong checksum, a skip in the sequence number or a timeout while waiting for the packet, it sends a NACK message to the sender and the sender multicasts the lost packets in response to the NACK. It is non-deterministic since the sender is unable to decide when all the receivers have received the packet successfully. Some receiver-initiated protocols multicast NACK messages to all receivers to avoid redundant NACK messages. An example for a receiver-initiated protocol is PGM [91] and an example for a NACK-avoidance receiver-initiated protocol is [84]. Both of the sender-initiated protocols and the receiver-initiated protocols suffer the acknowledge implosion problem. Tree based protocols arrange all the receivers into a hierarchy called the ACK tree which is responsible to collect acknowledgements and retransmit lost data packets. Each receiver sends ACK/NACKs to its parent node only, collects the ACK/NACKs from its child nodes and is responsible for retransmitting lost data packets to its child nodes. By limiting the degree of the ACK tree, we can make it possible that no node is overwhelmed by ACK/NACKs.

RMTP (Reliable Multicast Transport Protocol) [92] and TMTP (Tree-based Multicast Transport Protocol) [88] are two typical tree based reliable multicast protocols. RMTP provides sequenced, lossless delivery of bulk data from a sender to a group of receivers. In RMTP, receivers are grouped into a hierarchy of local regions, with a Designated Receiver (DR) in each local region. Receivers send ACK messages to the DR of its local region, then DRs send ACK messages to the DRs in the upper level. The sender is the DR on the top level. DRs cache the data and respond to retransmission requests in the local region. For example, Fig. 7.1 shows a possible ACK tree built on the multicast tree in RMTP. The rectangles represent the receivers and the filled ones are the designated receivers each of which is responsible to retransmit packets to one or several other receivers. In TMTP, the ACK tree is called *control tree*. Typically, in TMTP the receivers in the same subnet belong to a domain and a single domain manager acts as the parent of other receivers in the domain.

However, since both protocols build the ACK tree based on the multicast tree, this causes the following problems. First, the eligible ACK tree may not exist. In the ACK tree of RMTP, the parent of a receiver must be the ancestor of the receiver in the multicast tree. In an extreme case, all the receivers have to choose the sender as their parent. Thus, the protocol degenerates to a receiver-initiated protocol. It is even worse when there are some constraints, as it may be impossible to find an eligible ACK tree under the constraints. Second, since the parent and child in the ACK tree is the ancestor and descendant in the multicast tree, there must be some correlation among their data loss probabilities. For example, if the data is lost

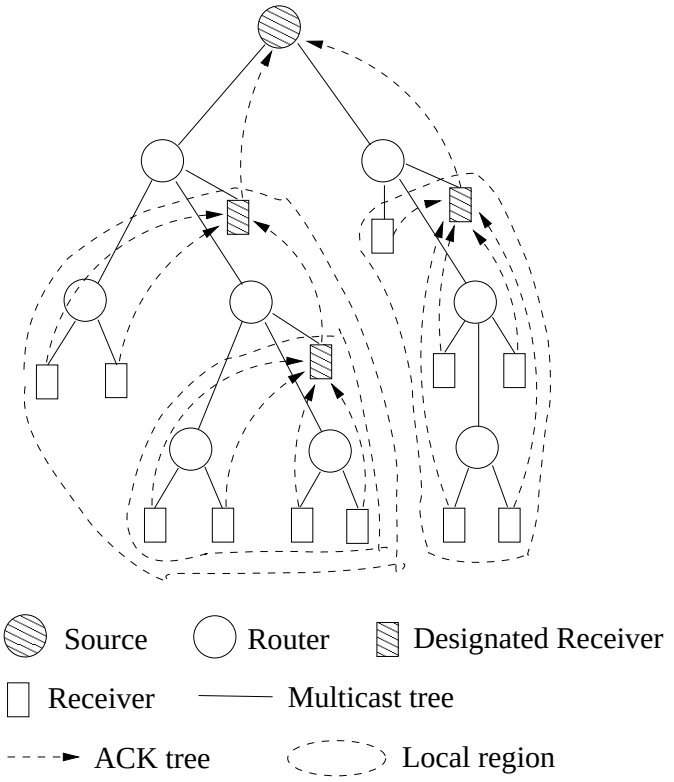


Figure 7.1: ACK tree in RMTP.

before reaching the parent, the child will not receive the data definitely. If the child sends the NACK message to the parent, it will not receive the retransmitted data because the parent is also requesting the data. Third, the superposition of the ACK tree and the multicast tree increases the traffic on the same links, thereby increases the probability of congestion.

In this chapter, we propose a new reliable multicast protocol which can minimize retransmission delay and overcome the problems discussed above. The proposed protocol takes advantage of both the tree based approach and the peer-to-peer technique. Our protocol constructs a logical ACK tree independent of the multicast tree. In our protocol, any two receivers can be the parent node and child node in the ACK tree. Fig. 7.2(b) shows a possible ACK tree given the receiver set is the same as in the Fig. 7.2(a). Intuitively, the average retransmission delay can be decreased greatly. To take advantage of the flexibility of the ACK tree construction, a receiver uses a heuristic function to choose another receiver as its parent. The heuristic function is designed to minimize the retransmission delay. The child node sends ACK/NACK messages to the parent node and receives retransmitted packets

from the parent node. The price for the improved retransmission delay is the increased complexity of window management in the window based flow control. We use a state transition diagram to describe the operations of the sender and the receivers. The sender can only send data packets whose sequence numbers are in the window, and receivers can only accept data packets whose sequence numbers are in the window. The window in the parent node will not advance unless the parent node receives all the ACK messages from its child nodes. Under the window based flow control mechanism, the receivers can maximize their receiving rate satisfying the retransmission requirements of its children.

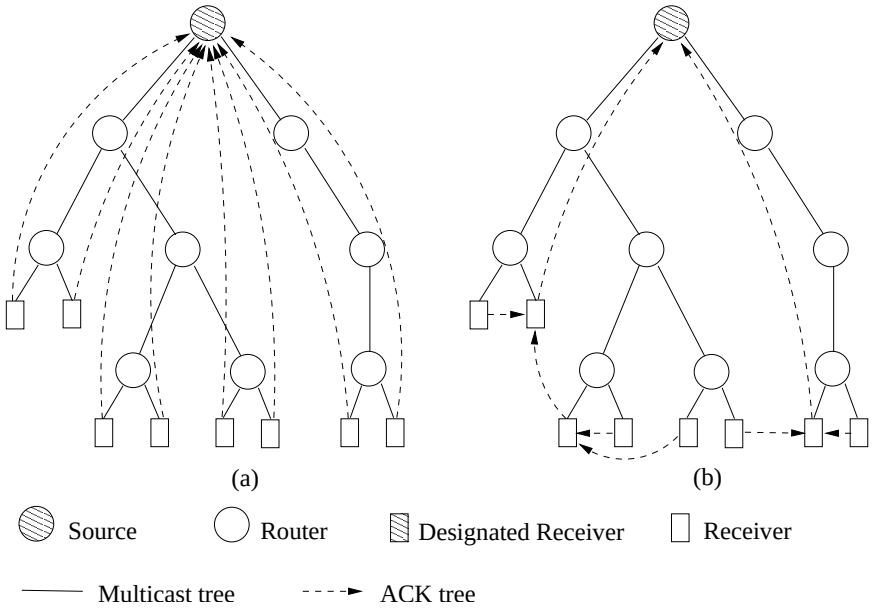


Figure 7.2: Illustration of ACK tree in reliable multicast protocol. (a) ACK tree in RMTP; (b) ACK tree in our protocol.

7.1 Preliminaries

We assume that our reliable multicast protocol is deployed in the service-centric multicast architecture proposed in [93] along with the multicast routing protocol SCMP. The service-centric multicast architecture provides flexible and efficient multicast service by processing most multicast related tasks in a powerful router, which is called *m-router*. The m-router collects JOIN requests, builds the multicast tree and distributes the tree information around the network, while other routers in the network only need to perform minimum functions for routing. One reason for

adopting this multicast architecture is that the architecture provides the centralized service which our protocol can use to construct the ACK tree with little protocol overhead. Another reason is that the ACK tree information can be distributed along with the multicast tree information which saves the network bandwidth. The sender in our protocol is always the m-router in the architecture. However, our reliable multicast protocol is not restricted to the multicast architecture in [93]. It can be used in any network running a link state unicast routing protocol.

We also assume that each data packet has a sequence number which represents its order in the data stream. The receivers detect data loss or out of order based on the sequence number of the received data packets. To simplify the presentation, we use "packet n" to represent "the data packet whose sequence number is n" and "packet less than n" to represent "the data packet whose sequence number is less than n."

We use $parent_{ACK}$ to represent the parent of a receiver in the ACK tree. Similarly, we use $child_{ACK}$ to represent a child of a receiver in the ACK tree. For any receiver, there is a unique path on the multicast tree connecting the receiver to the m-router. The delay of this path determines how soon the receiver will receive the data packet after the m-router sends it. We denote this unique path by $path_m$, the length of the path by $pathlen_m$, and the delay of the path by $delay_m$.

7.2 Peer-to-Peer Tree Based Reliable Multicast Protocol

In this section, we will first give an overview of the proposed protocol. Afterwards, we will describe the operations for both the source and the receivers in details.

7.2.1 Protocol Overview

Under the service-centric multicast architecture, the m-router has the full knowledge of the network topology and the receivers. The ACK tree is constructed dynamically in a similar way the multicast tree is constructed. However, the multicast tree is a physical tree while the ACK tree is a logical tree. Each receiver keeps the IP address of its $parent_{ACK}$. The physical links of the ACK tree are determined by unicast routing. Upon receiving a JOIN request from a new receiver, the m-router will graft it to both the multicast tree and the ACK tree. An existing receiver on the ACK tree is selected as the $parent_{ACK}$ of the new receiver. The selection is based on a heuristic function to be described in the next section. To save bandwidth, the m-router encapsulates the $parent_{ACK}$ information in the packet used for constructing the multicast tree. After receiving the $parent_{ACK}$ information, the new receiver

sends ACK_JOIN to the $parent_{ACK}$ and the $parent_{ACK}$ adds the new receiver as a $child_{ACK}$.

The data packets flow along the multicast tree first. Once a receiver detects data loss, it will use the ACK tree to recover. The receiver experiencing data loss sends the NACK message which includes a sequence number n and a bitmap. All the packets less than n are received correctly up to now. The length of the bitmap is a variable and the bitmap indicates whether the packets following packet n are correctly received or not. The parent retransmits the lost packets by unicasting according to the bitmap if the required packets are in its buffer.

When the physical links are unstable or the sending rate of the sender exceeds the processing ability of the receiver, our protocol uses window based flow control to inform the sender to lower its sending rate. Ideally, all the windows should advance at different but close speeds. When some receiver is experiencing data loss, its window will stop advancing which triggers the stop of the window in its $parent_{ACK}$. This chain effect finally stops the window in the sender and then the sender stops sending data packets.

7.2.2 Constructing the ACK Tree

Before we give the details of the ACK tree construction, we first introduce a new term. For any two receivers u and v, the two paths $path_m(u)$ and $path_m(v)$ may have some common links. Since there is no loop in the multicast tree, the number of the common links can be calculated by tracing the two paths from the m-router until they ramify. We use $commonlink(u, v)$ to represent this number. Generally, the larger the $commonlink(u, v)$, the more correlation between the data loss probabilities of the receivers u and v.

When a new receiver joins the group, any node on the ACK tree is a candidate for the $parent_{ACK}$ of the receiver. The following is some heuristics for selecting $parent_{ACK}$:

- Rule 1: the delay between the receiver and its $parent_{ACK}$ should be minimized.
- Rule 2: the $delay_m(parent_{ACK})$ should be less than the sum of the $delay_m$ (receiver) plus the delay between $parent_{ACK}$ and the receiver.
- Rule 3: the $commonlink(parent_{ACK}, receiver)$ should be minimized.

Rule 1 is intuitive as the delay between the receiver and its $parent_{ACK}$ determines the delay of the ACK/NACK messages and the retransmitted data packets between them. Minimizing the delay helps to minimize the retransmission delay. However, the retransmission delay is affected not only by this delay, but also by the

probability that the $parent_{ACK}$ has the required data packets in its buffer. Rules 2 and 3 are trying to maximize this probability. Rule 2 guarantees that the data packets will reach $parent_{ACK}$ earlier than the NACK from the receiver. Rule 3 is to minimize the correlation of the data loss probabilities between the receiver and its $parent_{ACK}$. If such correlation is strong, that is to say a data packet which is lost before arriving at the receiver is very likely lost before arriving at its $parent_{ACK}$, then it is very likely that $parent_{ACK}$ does not have the required packets when receiving the NACK from the receiver.

It is difficult to find a perfect $parent_{ACK}$ satisfying all the three rules. In fact, these three rules contradict each other in some cases. Here we give a preference heuristic function which takes all the three rules into consideration.

$$P(u,v) = \begin{cases} delay(u,v) * e^{delay_m(v)-delay_m(v)-delay(u,v)} * \\ commonlink(u,v) * (1/pathlen_m(u) + 1/pathlen_m(v)) \\ \qquad\qquad \text{if } delay_m(v) + delay(u,v) - delay_m(v) \geq 0 \\ \\ MAX \qquad\qquad \text{if } delay_m(v) + delay(u,v) - delay_m(v) < 0 \end{cases}$$

where u is a candidate for $parent_{ACK}$ and v is the new receiver, $delay(u,v)$ is the delay between u and v, and MAX is a very large number. The m-router calculates the preference values for each candidate and choose the candidate with the lowest preference value as the $parent_{ACK}$.

7.2.3 Loss Recovery and Flow Control

Loss recovery and flow control both strongly depend on a data structure - window. Suppose data packets consist of a stream in an ascending order of sequence numbers. A window is a span of continuous sequence numbers. The length of the window is $window\_size$. The first sequence number determines the position of the window.

A receiver uses both ACK and NACK to inform its $parent_{ACK}$ its current status. Each ACK includes a sequence number $current\_seqno$, which indicates that all the packets less than $current\_seqno$ are received correctly and the receiver is waiting for the packet $current\_seqno$. The $current\_seqno$ is always in the window. Each NACK includes a sequence number $current\_seqno$ and a bitmap. The sequence number has the same meaning as in ACK. The bitmap is stored in the window and indicates whether the packets after packet $current\_seqno$ are correctly received or not. The $parent_{ACK}$ retransmits the lost data packets according to the bitmap.

All the windows advance at their own speeds until they reach the end of the data stream. A window will not advance until some conditions are satisfied. Based on these conditions, we divide the nodes into three types: *sender* which is the root node of the ACK tree, *i-receiver* which is the inner node of the ACK tree and *l-receiver*

which is the leaf node of the ACK tree. Each type of nodes take different actions when receiving ACK/NACKs. We describe each of them next.

Operation of the Sender

The window in the sender [1] is shown in Fig.7.3. The packets colored in red are sent already. $current\_seqno$ is the sequence number of the packet the sender is about to send next. $last\_acked$ is the least sequence number among all the ACK/NACKs collected from the $child_{ACK}$ of the sender. After the sender sends

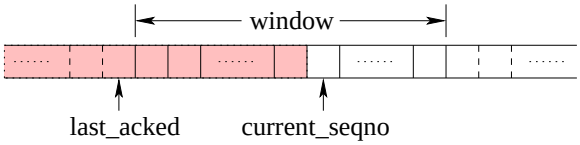


Figure 7.3: The window in the sender.

a data packet, it increases the $current\_seqno$ by 1. The sender will keep sending data packets until the window is full, and at this time, $last\_acked + window\_size = current\_seqno$. When the sender receives ACK/NACKs from a $child_{ACK}$, it will update the $last\_acked$ if necessary. The increase of the $last\_acked$ results in the advance of the window. For example, if the $last\_acked$ is increased by m, the window advances m packets as well, where m is an integer between 0 and $window\_size$. When the window is full, the sender will stop sending data packets. It checks the window periodically and resumes to send data packets after the window advances.

Operation of the I-Receiver

The i-receivers are the most complicated nodes among the three types of nodes because they have to do three things simultaneously: receiving the multicast data through the multicast tree, sending ACK/NACKs to their $parent_{ACK}$ and collecting the ACK/NACKs from their $child_{ACK}$. To simplify the presentation, we use a state transition diagram to describe the operation of the i-receiver. We use four states to represent the status of an i-receiver. Each i-receiver must be in one of the states.

- S_1: the i-receiver does not detect any packet loss or out of order, and the window is not full.
- S_2: the i-receiver detects packet loss or out of order, and the window is not full.

[1] Here the sender is the source node.

- S_3: the i-receiver does not detect any packet loss or out of order, and the window is full.
- S_4: the i-receiver detects packet loss or out of order, and the window is full.

The state transition diagram is shown in Fig.7.4.

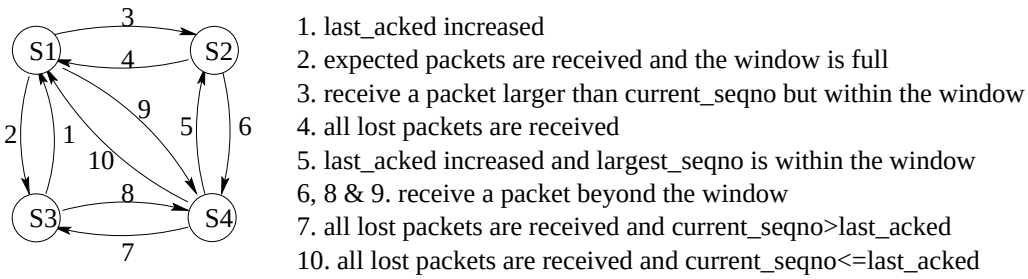


Figure 7.4: State transition diagram of i-receiver.

Ideally, if there is no packet loss or out of order, all the receivers will stay in S_1. The window is as shown in Fig.7.5, where $last\_acked$ is the same as in the sender, $current\_seqno$ is the sequence number of the packet that the i-receiver is expecting to receive, the packets colored in red are received correctly already, and $largest\_seqno$ is the largest sequence number of the received packets. When the i-receiver receives packet $current\_seqno$, both $current\_seqno$ and $largest\_seqno$ increase by 1. The window advances when the $last\_acked$ increases after receiving ACK/NACK from a $child_{ACK}$.

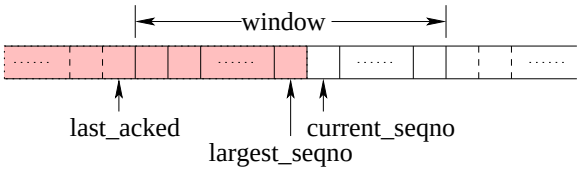


Figure 7.5: The window of i-receiver in state S_1 when $last\_acked$ is less than $current\_seqno$.

In Section 7.2.2, we used rule 2 to ensure that the data packets reach the $parent_{ACK}$ before the NACK from a $child_{ACK}$. But when the $parent_{ACK}$ is experiencing a much heavier congestion than the $child_{ACK}$, the window in the $parent_{ACK}$ will lag behind the window in the $child_{ACK}$. As a result, the $last\_acked$ in $parent_{ACK}$ will become larger than $current\_seqno$. This case is shown in Fig.7.6.

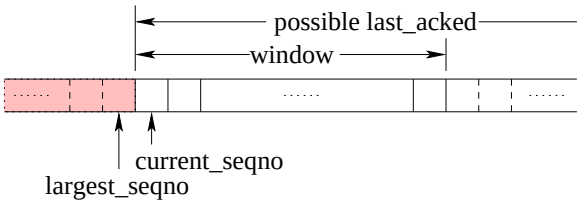


Figure 7.6: The window of i-receiver in state S_1 when $last\_acked$ is equal to or greater than $current\_seqno$.

In this case, the window is empty. The only condition that the window will advance is the i-receiver receives packet $current\_seqno$. After $current\_seqno$ catches up with $last\_acked$, the $last\_acked$ will dominate the window advancement again.

If one receiver receives a packet whose sequence number is larger than $current\_seqno$, two cases are possible.

Case 1: If the packet's sequence number is within the window, the receiver's state transits from S_1 to S_2. The window is as shown in Fig.7.7. $largest\_seqno$ is updated to the received packet's sequence number. The bitmap is a bit string with length $largest\_seqno - current\_seqno + 1$, and it indicates whether the packets between $current\_seqno$ and $largest\_seqno$ are received correctly or not, where "1" means the packet is received correctly and "0" means the packet is lost or damaged. The i-receiver sends NACK containing $current\_seqno$ and the bitmap to its $parent_{ACK}$.

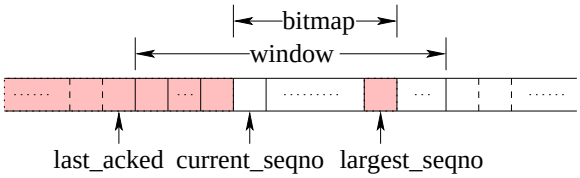


Figure 7.7: The window of i-receiver in state S_2 when $last\_acked$ is less than $current\_seqno$.

Similarly, if $last\_acked$ is larger than $current\_seqno$, the window is as shown in Fig.7.8.

Case 2: If the packet's sequence number is beyond the window, the receiver's state transits from S_1 to S_4. The window of S_4 is shown in Fig.7.9. $largest\_seqno$ is still updated to the received packet's sequence number. But this time the bitmap only includes the packets between $current\_seqno$ and the right end of the window due to the limitation of the window size. If the window advances m packets after updating $last\_acked$, the length of the bitmap increases by m and the values of the

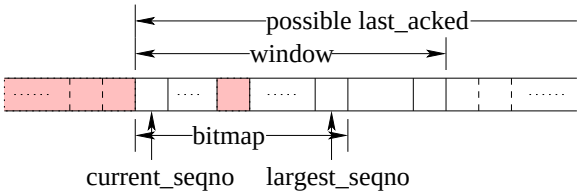


Figure 7.8: The window of i-receiver in state S_2 when $last\_acked$ is equal to or greater than $current\_seqno$.

new m bits are set to 0.

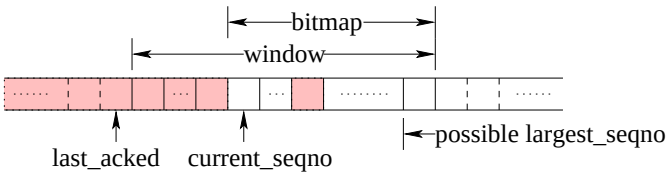


Figure 7.9: The window of i-receiver in state S_4 when $last\_acked$ is less than $current\_seqno$.

Similarly, when $last\_acked$ is larger than $current\_seqno$, the window is as shown in Fig.7.10. Now the bitmap includes the whole window of packets.

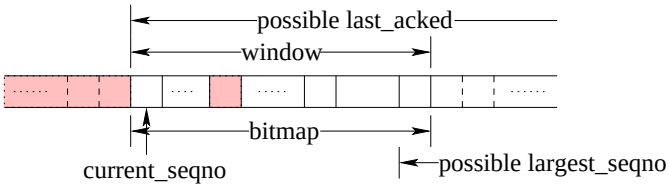


Figure 7.10: The window of i-receiver in state S_4 when $last\_acked$ is equal to or greater than $current\_seqno$.

If there is no data loss or out of order, but the $last\_acked$ is not updated in time due to that a $child_{ACK}$ is experiencing data loss, the window in the $parent_{ACK}$ will become full shortly and the state will transit from S_1 to S_3. The window in S_3 is shown in Fig.7.11. In this case, the i-receiver will stop increasing $current\_seqno$ even it receives packet $current\_seqno$. However, $largest\_seqno$ will be updated in a similar way to other three states.

When the i-receiver is in S_2, there are two situations resulting in a state transition: if the i-receiver receives a data packet beyond the window, the i-receiver transits from S_2 to S_4; or if the i-receiver receives all the lost packets recorded

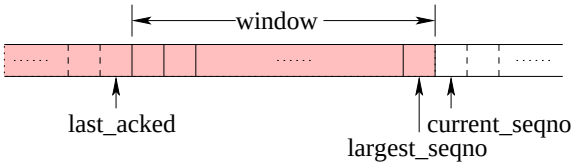


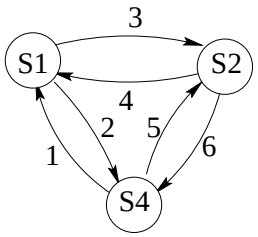
Figure 7.11: The window of i-receiver in state S_3.

in the bitmap, the i-receiver transits from S_2 to S_1. When the i-receiver is in S_3, it transits to S_1 if $last\_acked$ is increased due to receiving ACK/NACK, or to S_4 if it receives a data packet beyond the window. When the i-receiver is in S_4, its $largest\_seqno$ may be much larger beyond the window. The i-receiver transits to S_2 only if the $last\_acked$ is increased and the window advances further enough that $largest\_seqno$ lies in the window again. If $largest\_seqno$ is equal to $last\_acked + window\_size$ (it should be that $largest\_seqno$ is equal to $current\_seqno + window\_size - 1$ in the case that $last\_acked$ is no less than $current\_seqno$) and all the lost packets recorded in the bitmap are received correctly, the i-receiver transits from S_4 to S_3 (it should be S_1 in the case that $last\_acked$ is no less than $current\_seqno$).

We summarize the packet processing algorithm for i-receiver in pseudo-code in Table 7.1.

Operation of the L-Receiver

The l-receivers are similar to the i-receivers except that the l-receivers have no $child_{ACK}$. As they do not need to collect ACK/NACKs, $last\_acked$ is always equal to $current\_seqno - 1$. There are only three states for an l-receiver: S_1, S_2 and S_4. The state transition diagram is shown in Fig.7.12.



1. all lost packets are received
2. receive a packet beyond the window
3-6. same as i-receiver

Figure 7.12: State transition diagram of l-receiver.

Table 7.1: Packet Processing Algorithm for i-receiver (Part I)

Input: $packet$.
Output: State transition, ACK/NACK.
Packet Processing Algorithm:
//We omit the "break" in the switch statements for easy reading.
switch (type of the $packet$)
  case data packet:
  //suppose the sequence number of the data packet is $seqno$
    switch (current state)
      case S_1:
        if $packet$ is the expected packet
          increase $current\_seqno$ and $largest\_seqno$ by 1;
          if window is 3/5 full
            send ACK;
          if window is full
            state changes to S_3;
        else if $seqno > current\_seqno$ and $packet$ is within the window
          update $largest\_seqno$ and bitmap;
          state changes to S_2;
          send NACK;
        else if $packet$ is beyond the window
          update $largest\_seqno$ and bitmap;
          state changes to S_4;
          send NACK;
      case S_2:
        if $packet$ is beyond the window
          update $largest\_seqno$ and bitmap;
          state changes to S_4;
          send NACK;
        else if $seqno > current\_seqno$ and $packet$ is within the window
          update $largest\_seqno$ and bitmap;
        else if $packet$ is between $current\_seqno$ and $largest\_seqno$
          update bitmap;
        else if $packet$ is the expected packet
          update $current\_seqno$ and bitmap;
          if all the lost packets are received
            state changes to S_1;
            send ACK;
      case S_3:
        if $packet$ is beyond the window
          update $largest\_seqno$;
          state changes to S_4;

Table 7.2: Packet Processing Algorithm for i-receiver (Part II)

```
    case S4:
      if packet is beyond the window
        update largest_seqno if necessary;
      else if seqno > current_seqno and packet is within the window
        update bitmap;
      else if packet is the expected packet
        if last_acked < current_seqno
          update current_seqno and bitmap;
          if all the lost packets are received
            state changes to S3;
        else
          update current_seqno and bitmap;
          if all the lost packets are received
            state changes to S1;
            send ACK;
          else
            state changes to S2;
            send NACK;
  case ACK packet:
    update last_acked if necessary;
    if window advances
      if current state is S3
        state changes to S1;
      if current state is S4 and largest_seqno is within the window
        state changes to S2;
  case NACK packet:
    update last_acked if necessary;
    if window advances
      if current state is S3
        state changes to S1;
      if current state is S4 and largest_seqno is within the window
        state changes to S2;
    retransmit the lost data packets if necessary;
End
```

7.2.4 Timers

Our protocol uses different timers to improve robustness and performance. There are three types of timers: poll timer, NACK timer and status timer.

Poll timer is scheduled in the sender when the window is full. When the timer is timeout, the sender checks whether the window advances. If the window advances, the sender resets the timer and resumes sending data packets. If the window is still full, the sender reschedules the timer.

When a receiver detects data loss, it sends an NACK to its $parent_{ACK}$ immediately. Although this helps to decrease the retransmission delay, it may waste a lot of bandwidth. For example, the "lost" packet is not really lost, it simply experiences a longer delay than the packet after it. In this case, the NACK is redundant and unnecessary. We use the NACK timer to avoid such situation. Every time the receiver detects data loss, it schedules an NACK timer. If the packet is still missing when the timer is timeout, the receiver then sends out the NACK.

Status timer is used by a $child_{ACK}$ to periodically send status information to its $parent_{ACK}$. When the timer is timeout, the receiver will send an ACK or NACK based on its current status. If the receiver is in states S_1 or S_3, it will send an ACK; if it is in states S_2 or S_4, it will send an NACK.

7.3 Theoretical Analysis

7.3.1 Protocol Correctness

A protocol is considered *correct* if it is shown to be both *safe* and *live* [94]. For a reliable multicast protocol to be live, no deadlock should occur at any receiver or the sender. In reliable multicast protocols, deadlock happens when a receiver requests a retransmission of a packet which is deleted due to the buffer size limit. For a reliable multicast protocol to be safe, all the data packets should be delivered to the higher layer at any receiver in a finite time. To address the correctness of our protocol, we assume the nodes never fail during the multicast session and both ACK and NACK messages are transmitted without loss or error.

Theorem 8 *Peer-to-peer tree-based reliable multicast protocol is safe and live.*

Proof: Suppose the height of the ACK tree is h. The proof proceeds by induction on h.

For the case in which $h = 1$, the protocol reduces to a non-hierarchical receiver-initiated reliable multicast protocol. For each (sender, receiver) pair, according to the protocol description, the sender will not advance its window until it receives ACK from the receiver, i.e. the packet is stored in the sender's window until the

receiver notify the sender about its successful reception. This guarantees that the receiver will receive all the missing packets in a finite time(we ignore the probability that the packet is lost indefinite times). As the sender will not delete the packet until the receiver responds ACK, no deadlock is possible.

For $h > 1$, assume the theorem holds for any $t < h$. We want to prove it holds for $t = h$. Consider a subtree of the ACK tree which includes all the nodes whose height is at most $t - 1$. By the inductive hypothesis, the protocol is live and safe in the subtree, i.e. the nodes in the subtree receive the packet in a finite time and no deadlock occurs at any node of the subtree. For the rest nodes of the ACK tree, each of them has a node in the subtree as its $parent_{ACK}$. The $parent_{ACK}$ will not advance its window until it receives all ACKs from its $child_{ACK}$s which are the leaf nodes of the ACK tree. Therefore, the leaf nodes are guaranteed to receive the packet in a finite time without deadlock. In conclusion, the protocol is live and safe in the ACK tree. ■

7.3.2 Maximum Throughput Analysis

To analyze the maximum throughput of the protocol, we adopt the same model as used in [95] which focuses on the processing requirement of each packet rather than the communication bandwidth requirement. Accordingly, the maximum throughput is the inverse of the average per-packet processing time. We analyze the average per-packet processing time at the sender, the i-receiver and the l-receiver respectively and deduce the maximum throughput by locating the bottleneck. Before we delve into the analysis, there are three assumptions to make. First, similar to the previous section, we assume both ACK and NACK messages are transmitted without loss or error. Second, the out degree of the ACK tree is bounded by a constant B. Third, each data packet is transmitted from the sender to the receiver with loss probability η and the loss probabilities between different receivers are independent. The assumption of probability independence is also used in analysis of other tree-based reliable multicast protocols in [89]. However, the peer-to-peer tree based reliable multicast protocol is the most consistent with this assumption as the ACK tree is independent from the multicast tree.

Throughput of the Sender

We first consider X, the per-packet processing time required to successfully transmit a packet to all receivers at the sender. It can be expressed as following:

$$\begin{aligned} X &= \text{(first transmission) + (multiple retransmission) + (receiving ACK/NACK)} \\ &= \Sigma_{m=1}^{M}(X_p(m)) + \Sigma_{i=1}^{L_1}(X_h(i)) + \Sigma_{i=1}^{L_2}(X_k(i)) \end{aligned} \tag{7.1}$$

where $X_p(i)$ is the time taken to transmit the packet for the i^{th} time, $X_h(i)$ is the time taken to process the i^{th} ACK message and $X_k(i)$ is the time taken to process the i^{th} NACK message. M is the number of transmissions that the sender has to send before all the receivers receives the packet successfully. L_1 is the number of the ACK messages received and L_2 is the number of the NACK messages received. Taking expectation, we have

$$E(X) = E(M)E(X_p) + E(L_1)E(X_h) + E(L_2)E(X_k) \tag{7.2}$$

After the sender sends the packet for the first time, $B\eta$ of its children will not receive it successfully. The sender has to retransmit the packet to these $B\eta$ children. After the retransmission $B\eta^2$ of its children will not receive it successfully, and so on. Thus the expectation of M is

$$E(M) = B + B\eta + B\eta^2 + \ldots = B/(1-\eta) \tag{7.3}$$

The number of children that receive the packet successfully after the first transmission is $B(1-\eta)$. Thus the expectation of L_1 is

$$E(L_1) = B(1-\eta) \tag{7.4}$$

After the first transmission, $B\eta$ of the children will respond NACK messages due to the unsuccessful reception of the packet. After the first retransmission, $B\eta^2$ of the children will respond NACK messages due to the unsuccessful reception of the packet and so on. Thus the expectation of L_2 is

$$E(L_2) = B\eta + B\eta^2 + \ldots = B\eta/(1-\eta) \tag{7.5}$$

Substituting equation (3), (4) and (5) into equation (2), we have

$$\begin{aligned} E(X) &= B/(1-\eta) * E(X_p) + B(1-\eta) * E(X_h) + B\eta/(1-\eta) * E(X_k) \\ &= O(B/(1-\eta)) \end{aligned} \tag{7.6}$$

Throughput of the L-receiver

We use Y^l to denote the per-packet processing time required for an l-receiver. We have

$$\begin{aligned} Y^l &= \text{(packet reception) + (sending ACK/NACK)} \\ &= Y_p + Y_h + \Sigma_{i=1}^{L_3} Y_k(i) \end{aligned} \tag{7.7}$$

where Y_p is the time taken to process packet reception. Y_h is the time taken to send ACK message and $Y_k(i)$ is the time taken to send the i^{th} NACK message. L_3 is the number of NACK messages the l-receiver send before successful reception of the packet. Take expectation, we have

$$E(Y^l) = E(Y_p) + E(Y_h) + E(L_3)E(Y_k) \tag{7.8}$$

The l-receiver receives the packet successfully after one retransmission with probability $\eta(1-\eta)$, i.e. the l-receiver sends one NACK message with probability $\eta(1-\eta)$. Similarly, the l-receiver sends two NACK messages with probability $\eta^2(1-\eta)$ and so on. Thus we have

$$E(L_3) = \eta(1-\eta) + 2\eta^2(1-\eta) + 3\eta^3(1-\eta) + \ldots = \eta/(1-\eta) \tag{7.9}$$

Substituting equation (9) into equation (8), we have

$$\begin{aligned} E(Y^l) &= \eta/(1-\eta) * E(Y_k) + E(Y_p) + E(Y_h) \\ &= O(\eta/(1-\eta)) \end{aligned} \tag{7.10}$$

Throughput of the I-receiver

We use Y^i to denote the per-packet processing time required for an i-receiver. It can be expressed as following equation

$$\begin{aligned} Y^i &= \text{(packet reception) + (sending ACK/NACK) +} \\ &\quad \text{(receiving ACK/NACK) + (multiple retransmission)} \\ &= Y_p + Y_h + \Sigma_{i=1}^{L_3}(Y_k(i)) + \\ &\quad \Sigma_{m=2}^{M}(X_p(m)) + \Sigma_{i=1}^{L_1}(X_h(i)) + \Sigma_{i=1}^{L_2}(X_k(i)) \end{aligned} \tag{7.11}$$

Take the expectation and substitute the previous results, we have

$$\begin{aligned} E(Y^i) &= E(Y_p) + E(Y_h) + E(L_3)E(Y_k) + \\ &\quad (E(M)-1)E(X_p) + E(L_1)E(X_h) + E(L_2)E(X_k) \\ &= O(\eta/(1-\eta) + B/(1-\eta)) = O((B+\eta)/(1-\eta))) \end{aligned} \tag{7.12}$$

Throughput of the System

Let the throughput at the sender, the l-receiver and the i-receiver be Λ_s, Λ_l and Λ_i respectively. The throughput of the system Λ is

$$\Lambda = min\{\Lambda_s, \Lambda_l, \Lambda_i\} \tag{7.13}$$

From equation (6), (10) and (12) it follows that

$$1/\Lambda = O((B+\eta)/(1-\eta)) \tag{7.14}$$

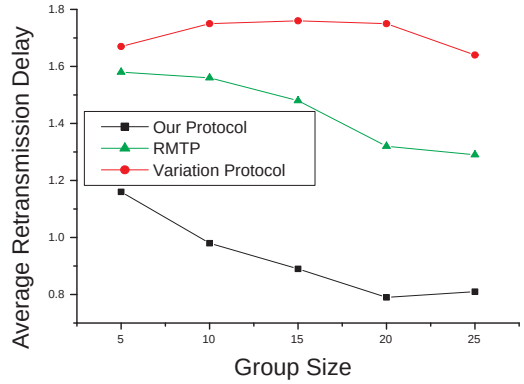
Therefore, the throughput of the system decreases when the data loss probability increases or the out degree of the ACK tree increases. If η is a constant, the throughput of the system is of the order of a constant B and independent of the size of group members.

7.4 Performance Evaluations

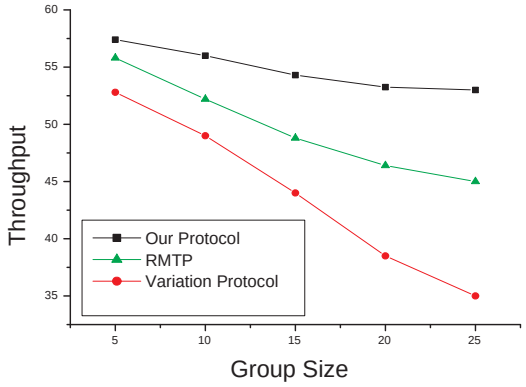
We have implemented our protocol on the NS2 simulator and evaluated the performance through simulations. For comparison purpose, we also implemented RMTP [92] and another protocol, which can be considered as a simplified version of our protocol. Considering [92] does not specify how to find the DR in each local region, we implement RMTP by assigning an existing group member to be the DR for a new group member. Specifically, in our simulation, after a new group member joins the multicast session, the group member nearest to the new group member along the multicast tree is assigned to be the DR. In the variation protocol, all the receivers send ACK/NACKs directly to the sender. Compared to the original protocol, the variation protocol does not have a hierarchical structure for loss recovery and flow control, and the height of the ACK tree in the variation protocol is always 2. We focus on two metrics: average retransmission delay and throughput. The retransmission delay is the time experienced by a receiver from it sends out an NACK message until it receives all the required data packets. The throughput is defined as the number of packets the sender has sent before the simulation completes, which is equal to the $current\_seqno$ of the sender.

The network topologies are generated by GT-ITM [27]. Nodes are picked randomly to join a multicast group. The sender sends data packets at a constant rate as long as the window is not full. The simulation time is 50 seconds. There are two tunable parameters: packet drop probability p and the window size $window\_size$. Each link drops the data packets independently and randomly with probability p.

Fig.7.13(a) shows the average retransmission delay under different group sizes. We can see that the average retransmission delay of our protocol is much shorter than that of RMTP and the variation protocol no matter what the group size is. The average retransmission delay of the variation protocol increases slightly as the group size increases. On the other hand, as the increase of the group size, both the average retransmission delay of our protocol and that of RMTP decrease. This is because that when the group size increases, the number of the $parent_{ACK}$ candidates increases too. It is more likely to find an eligible $parent_{ACK}$ close to the



(a)



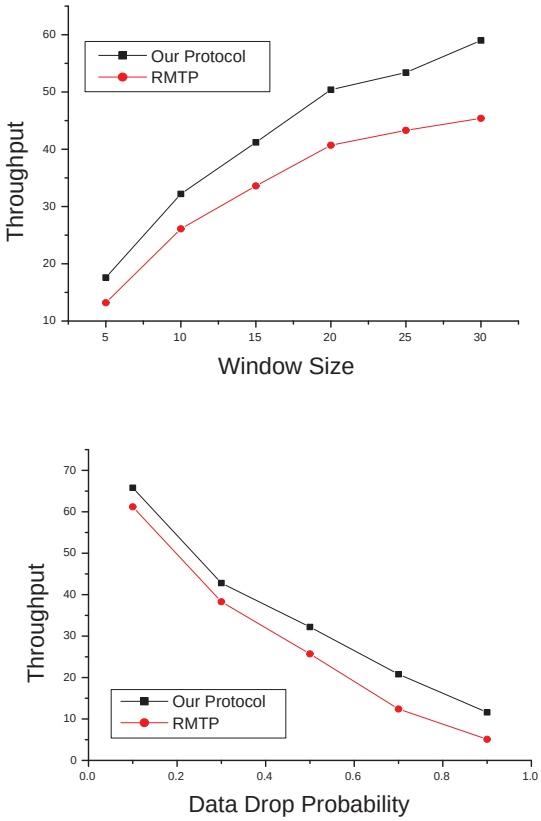
(b)

Figure 7.13: Performance comparison under different group sizes. (a) Average retransmission delay vs. group size; (b) Throughput vs. group size.

receiver. When the group size becomes large enough, the average retransmission delay almost remains constant.

Fig.7.13(b) shows the throughput under different group sizes. As can be seen, the throughput of our protocol is greater than that of RMTP or the variation protocol. When the group size increases, the gap between our protocol and the other two protocols becomes larger. The throughput of all the three protocols decreases as the group size increases. But the throughput of our protocol decreases at a much lower pace. When the group size becomes large enough, the decrease of the throughput is slim and the throughput of our protocol is around 20% higher than that of RMTP. From these observations, we can see that our protocol scales well when the group

size increases.



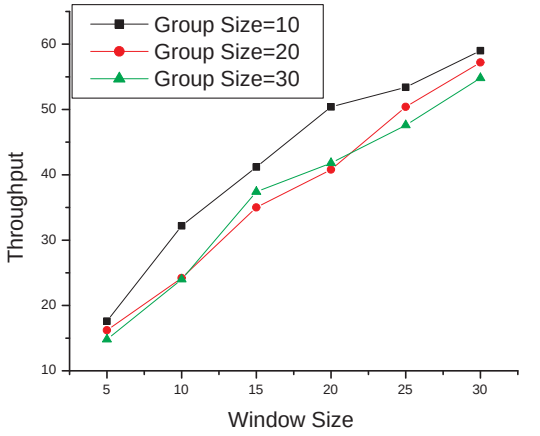
(a)

(b)

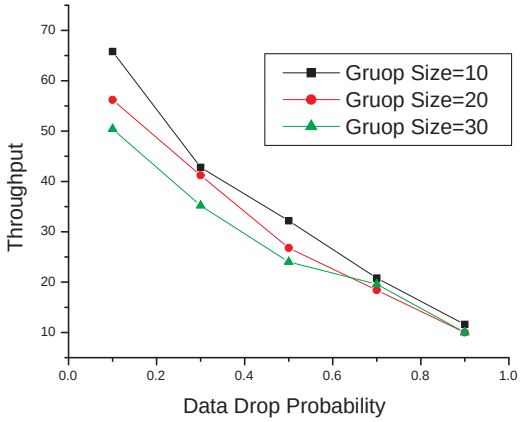
Figure 7.14: Performance comparison of tree-based protocols(1). (a) Throughput vs. window size; (b) Throughput vs. packet drop probabilities.

Previous simulation demonstrates that tree based reliable multicast protocols outperform non-hierarchical protocols in terms of average retransmission delay and throughput. Here we compare the two tree based protocols by tuning the two parameters: window size and data drop probability. Fig.7.14(a) and (b) shows the throughput under different window sizes and data drop probabilities respectively. Our protocol always achieves higher throughput than RMTP. The advantage remains constant when the window size is larger while it is more obvious when the data drop probability is higher. This is because the correlation of the ACK tree and the multicast tree in RMTP make it more vulnerable to data drop probability.

Now we variate the group size while tuning the parameters to observe the throughput of our protocol. Fig.7.15(a) shows the throughput under different window sizes.



(a)



(b)

Figure 7.15: Performance comparison of tree-based protocols(2). (a) Throughput vs. window size; (b) Throughput vs. packet drop probabilities.

We can observe that as the increase of the window size, the throughput increases as well. The throughput when the group size is 10 is better than that when the group size is 20 or 30. The two curves of group size is 20 and 30 interweave with the increase of the window size. It means that the group size has little impact on throughput when it is large enough.

Fig.7.15(b) shows the throughput under different packet drop probabilities. We can see that as the increase of the packet drop probability, the throughput decreases. Still, the throughput when the group size is smaller is better than that when the group size is larger. However, this advantage diminishes when the packet drop

probability is high enough.

7.5 Summary

we have proposed a peer-to-peer tree based reliable multicast protocol. Compared to existing reliable multicast protocols, our protocol can avoid the acknowledge implosion and minimize the retransmission delay therefore increase the system's throughput. Constructing the ACK tree in a peer-to-peer fashion makes our protocol transparent to routers and easy to deploy. In the proposed window based loss recovery and flow control scheme, the window in the child node can advance faster than the window in the parent node, which can greatly increase the throughput. Our simulation results show that the new protocol can achieve good scalability. Its average retransmission delay and throughput are much better than another tree based reliable multicast protocol RMTP and a variation non-hierarchical reliable multicast protocol. In the new protocol, the throughput increases when the window size increases or the drop probability decreases, and the throughput differences between different group sizes are marginal.

Chapter 8

Conclusions and Future Work

In this thesis, we have discussed a wide range of topics on multicast and network coding. All the topics are centered on improving the performance of group communication over a large scale network.

Since Ahlswede *et al.* laid down the theoretic foundation of network coding on multicast in [13], there have been a lot of research on network coding, especially on linear network coding. We aim to solve a series of minimal linear network coding problems with a unified approach in Chapter 2. With the help of hypergraph, we transform a linear network coding problem in multicast networks into a graph theory problem. Under such transformation, a linear network coding assignment for a multicast network is mapped to a cover in a hypergraph satisfying some valid code constraints. Given the minimum cut of the multicast network and the number of receivers, an eligible cover can be found in polynomial time with the iterative refinement algorithm. In addition, we proposed preprocessing algorithms which can effectively reduce the network bandwidth consumption and the computation time, which makes it practical to apply this approach to a large scale network.

In Chapter 3 and Chapter 4, we investigate the performance improvement of applying linear network coding to peer-to-peer systems. The peer-to-peer systems are suitable for network coding as the overlay topology is constructed arbitrarily. Therefore we can construct the overlay topology in favor of network coding. For peer-to-peer file sharing systems discussed in Chapter 3, we focus more on throughput and reliability. The proposed peer-to-peer file sharing scheme PPFEED utilizes combination networks as its topology prototype. Peers encode and decode with a simple efficient linear coding function. The redundant links can greatly improve the reliability and resilience of the system with little overhead.

Compared to peer-to-peer file sharing systems, peer-to-peer media streaming systems have different concerns which are discussed in Chapter 4. For peer-to-peer streaming systems, we are more concerned about end-to-end delay and link heterogeneity. Therefore we designed an overlay construction scheme which can

optimize the overlay topology for live streaming systems. One of the merits of the proposed scheme is that it can make efficient use of the uplink bandwidth of the peers and satisfy the heterogeneous downloading rate requirements of the peers as much as possible. The topology construction is dynamic such that it can adapt itself to the changing peers and minimize the end-to-end delay and link stress at the same time. After the overlay topology is formed, an adaptive network coding scheme for peer-to-peer media streaming systems is proposed and applied on it. The scheme takes advantage of the overlay topology and media encoding scheme MDC to encode the media content into multiple stripes. Peers subscribe a subset of the stripes based on their capability and network coding requirements. Peers with different download bandwidths can receive the media content at different rates simultaneously.

Inter-session linear network coding is a challenging topic on network coding. In Chapter 5, we examined the linear inter-session network coding for multicast which involves multiple simultaneous multicast sessions. We introduced two performance metrics to characterize the benefit of inter-session network coding with each metric having its own application targets. The sessions are divided into different groups based on the metrics. Linear network coding, both deterministic and random, is performed between sessions within the same group. The simulation results show that the inter-session network coding outperforms the intra-session network coding by about 30% in terms of throughput in most cases. Moreover, the deterministic algorithm achieves higher throughput and less bandwidth consumption than the random algorithm.

In the last two chapters, we focused on network layer multicast. In Chapter 6, we proposed a service-centric multicast architecture and an efficient and flexible multicast routing protocol SCMP. It is in contrast with traditional distributed multicast protocols. We argue that centralizing multicast functions in multiple powerful routers will not impair the scalability of the system. Instead, the system can benefit a lot from the service-centric architecture. By concentrating most multicast routing and service-related tasks in the m-router, routing efforts on other routers can be greatly reduced and the bandwidth wasting in the rest of the Internet can be avoided. The centralized processing make it easier to deploy a sophisticated network-wide routing algorithm with complex QoS requirements. The simulation results show that SCMP protocol achieves the least data overhead and much lower protocol overhead than DVMRP and MOSPF.

We examined the transportation layer design for multicast in Chapter 7. Reliable multicast protocol is much more complex as it involves multiple receivers with different receiving rates and packet loss probabilities. ACK tree is a viable solution for reliable multicast. Existing solutions make the ACK tree dependent on the multicast tree which is inefficient in most cases. In extreme cases, it can degenerate to a

receiver-initiated protocol. We proposed a peer-to-peer tree based reliable multicast protocol which can avoid the acknowledge implosion and minimize the retransmission delay. The ACK tree is constructed in a peer-to-peer fashion, which makes our protocol transparent to routers and easy to deploy. The simulation results show that the new protocol achieves good scalability. It achieves 20% higher throughput than RMTP and much shorter average retransmission delay.

To devise a practical scheme for scalable and efficient group communication over a large scale network is an interesting yet challenging task which involves several research fields, such as multicast, peer-to-peer systems and network coding. The following topics are some open issues which shed light on our future work.

- How to handle network unstableness and fluctuation? Network is unstable due to many reasons. The network nodes or links may fail, the end-to-end delay may vary, the network traffic may get congested around a certain area. To handle this situations, some fault tolerance mechanism is needed. A common way is to find an alternative when the network is experiencing such a fluctuation. This requires data redundancy or link redundancy. In peer-to-peer systems based on network coding, it is readily to achieve link redundancy as network coding benefits from link redundancy by itself.

- We are interested in building a seamless group communication support system across the network. Due to the increasing popularity of wireless networks and mobile devices, it is critical to apply network coding to wireless networks and make the inter-operation as efficient as possible.

- It is desirable to extend the service-centric multicast architecture to support multiple m-routers. Multiple m-routers can make the system more scalable and reliable as the load can be distributed to these m-routers and the m-routers can be backup for each other. This demands a load balance mechanism and a coordinating mechanism between the m-routers. The multicast protocol SCMP is designed as an intra-domain routing protocol. We believe it has the potential to support inter-domain multicast routing efficiently. We are interested in conducting more research on this topic.

Bibliography

[1] S. Deering. Multicast routing in a dategram internetwork. Master's thesis, Stanford University, 1988.

[2] *Internet group management protocol, version 2 (IGMPv2)*, 1996.

[3] D. Waitzman and C. Partridge. *Distance vector multicast routing protocol*, Nov. 1988.

[4] J. Moy. *Multicast extension to OSPF*, 1994.

[5] B. Cain A. Ballardie and Z. Zhang. *Core based trees (CBT version 3) multicast routing*, 1998.

[6] S. Deering et al. *Protocol independent multicast-sparse mode (PIM-SM): motivation and architecture*, 1998.

[7] J. Nicholas A. Adams and W. Siadak. *Protocol independent multicast - dense mode (PIM-DM): protocol specification*, 2004.

[8] S. Banerjee and B. Bhattacharjee. A comparative study of application layer multicast protocols. 2001.

[9] M. Yang and Y. Yang. An efficient hybrid peer-to-peer system for distributed data sharing. *IEEE Trans. on Computers*, to apear.

[10] Bittorrent. *http://www.bittorrent.com*.

[11] B. Li X. Zhang, J. Liu and T.-S. P. Yum. Coolstreaming: a data-driven overlay network for efficient live media streaming. *Proc. IEEE INFOCOM 2005*, 2005.

[12] S. Seshan Y.H. Chu, S.G. Rao and H. Zhang. A case for end system multicast. *IEEE Journal on Selected Areas in Communication, Special Issue on Networking Support for Multicast*, (8), 2002.

[13] S.Y.R. Li R. Ahlswede, N. Cai and R.W. Yeung. Network information flow. *IEEE Trans. Information Theory*, 46:1204–1216, 2000.

[14] C. Fragouli and E. Soljanin. *Network Coding Applications*. Now Publishers Inc, 2008.

[15] R.W. Yeung S.Y.R. Li and N. Cai. Linear network coding. *IEEE Trans. Information Theory*, 49:371–381, 2003.

[16] P. Chou M. Effros S. Egner K. Jain S. Jaggi, P. Sanders and L. Tolhuizen. Polynomial time algorithms for multicast network code construction. *IEEE Trans. Information Theory*, 51:1973–1982, 2005.

[17] T. Ho and D.S. Lun. *Network Coding: An Introduction*. Cambridge University Press, 2008.

[18] R. Koetter D. Karger M. Effros J. Shi T. Ho, M. Medard and B. Leong. A random linear network coding approach to multicast. *IEEE Trans. Information Theory*, 52:4413–4430, 2006.

[19] R. Koetter K. Bhattad, N. Ratnakar and K.R. Narayanan. Minimal network coding for multicast. *Proc. IEEE International Symposium on Information Theory*, 2005.

[20] C.K. Ngai and R.W. Yeung. Network coding gain of combination networks. *IEEE Information Theory Workshop*, pages 283–287, Oct. 2004.

[21] Q. Zhang Z. Zhang J. Zhao, F. Yang and F. Zhang. Lion: Layered overlay multicast with network coding. *IEEE Trans. Multimedia*, pages 1021–1032, October 2006.

[22] M. Yang and Y. Yang. A hypergraph approach to linear network coding in multicast networks. *IEEE Trans. Parallel Distrib. Syst.*, to apear.

[23] R. Koetter J.K. Sundararajan, M. Medard and E. Erez. A systematic approach to network coding problems using conflict graphs. *Proc. of the UCSD Workshop on Information Theory and its Applications*, February 2006.

[24] C.E. Leiserson T.H. Cormen and R.L. Rivest. *Introduction to algorithms*. MIT Press, 1990.

[25] R. Koetter and M. Medard. An algebraic approach to network coding. *IEEE/ACM Trans. Networking*, 11(5):782–795, 2003.

[26] C. Fragouli and E. Soljanin. Information flow decomposition for network coding. *IEEE Trans. Information Theory*, 52:829–848, 2006.

[27] Gt-itm. URL http://www.cc.gatech.edu/projects/gtitm.

[28] P. Paul and S.V. Raghavan. Survey of multicast routing algorithms and protocols. *Proc. 15th International Conference on Computer Communication*, 2002.

[29] A.-M. Kermarrec A. Nandi A. Rowstron M. Castro, P. Druschel and A. Singh. Splitstream: high-bandwidth multicast in cooperative environments. *Proc. ACM SOSP 2003*, Oct. 2003.

[30] H.J. Wang V.N. Padmanabhan and P.A. Chou. Resilient peer-to-peer streaming. *Proc. IEEE ICNP*, 2003.

[31] J. Cheriyan and M.R. Salavatipour. Hardness and approximation results for packing steiner trees. *Algorithmica*, 45(1), 2006.

[32] M. Mahdian K. Jain and M.R. Salavatipour. Packing steiner trees. *14th ACM-SIAM Symp. on Discrete Algorithms*, 2003.

[33] The network simulator - ns-2. URL http://www.isi.edu/nsnam/ns/.

[34] B.C. Li Y. Zhu and J. Guo. Multicast with network coding in application-layer overlay networks. *IEEE Journal on Selected Areas in Communication*, September 2004.

[35] C. Gkantsidis and P. R. Rodriguez. Network coding for large scale content distribution. *IEEE INFOCOM 2005*, March, 2005.

[36] J. Shi M. Effros T. Ho, M. Medard and D.R. Karger. On randomized network coding. *Proc. Annual Allerton Conference on Communication, Control, and Computing*, 2003.

[37] Gnutella protocol development, the gnutella v0.6 protocol. *Available: http://rfc-gnutella.sourceforge.net/developer/index.html*, 2003.

[38] M. Luby J.W. Byers and M. Mitzenmacher. A digital fountain approach to asynchronous reliable multicast. *IEEE Journal on Selected Areas in Communication*, October 2002.

[39] R. M. Karp S. Ratnasamy, M. Handley and S. Shenker. Topologically-aware overlay construction and server selection. *IEEE INFOCOM 2002*, June, 2002.

[40] B. Bhattacharjee S. Banerjee and C. Kommareddy. Scalable application layer multicast. *Proc. ACM SIGCOMM 2002*, Aug. 2002.

[41] K. Hua D. Tran and T. Do. Zigzag: an efficient peer-to-peer scheme for media streaming. *Proc. IEEE INFOCOM 2003*, April 2003.

[42] V. Goebel K. Skevik and T. Plagemann. Evaluation of a comprehensive p2p video-on-demand streaming system. *Computer Networks*, (4), 2009.

[43] B. Botev D. Xu M. Hefeeda, A. Habib and B. Bhargava. Promise: peer-to-peer media streaming using collectcast. *Proc. 11th ACM international conference on Multimedia*, November 2003.

[44] D. Xu B. Bhargava M. Hefeeda, A. Habib and B. Botev. Collectcast: a peer-to-peer service for media streaming. *ACM/Springer Multimedia Systems Journal*, October 2003.

[45] N. Magharei and R. Rejaie. Understanding mesh based peer-to-peer streaming. *Proc. ACM NOSSDAV 2006*, 2006.

[46] V.K. Goyal. Multiple description coding: Compression meets the network. *IEEE Signal Processing Magazine*, September 2001.

[47] A. Vitali and M. Fumagalli. Standard-compatible multiple-description coding (mdc) and layered coding (lc) of audio/video streams. *Internet Draft - Network Working Group*, July 2005.

[48] Y. Cui and K. Nahrstedt. Layered peer-to-peer streaming. *Proc. ACM NOSSDAV 2003*, 2003.

[49] Y.T.H. Li D. Ren and S.H.G. Chan. On reducing mesh delay for peer-to-peer live streaming. *Proc. IEEE INFOCOM 2008*, April, 2008.

[50] C. Feng and B. Li. On large-scale peer-to-peer streaming systems with network coding. *Proc. 16th ACM International Conference on Multimedia*, October, 2008.

[51] C. Wang and N.B. Shroff. Intersession network coding for two simple multicast sessions. *Proc. Annual Allerton Conference on Communication, Control, and Computing*, September 2007.

[52] N. Sundaram and P. Ramanathan. Multirate media streaming using network coding. *Proc. 43rd Allerton Conference on Communication, Control, and Computing*, September 2005.

[53] A-M. Kermarrec M. Castro, P. Druschel and A. Rowstron. Scribe: A large-scale and decentralised application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communication, Special Issue on Networking Support for Multicast*, October 2002.

[54] L. Ford and D. Fulkerson. Maximal flow through a network. *Canadian J. Mathemat.*, pages 399–404, 1956.

[55] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *Proc. 18th Annual ACM Symposium on Theory of Computing*, pages 136–146, 1986.

[56] H. Balakrishnan D. Liben-Nowell and D. Karger. Analysis of the evolution of peer-to-peer systems. *Principles of Distributed Computing*, July 2002.

[57] C. Freiling R. Dougherty and K. Zeger. Insufficiency of linear coding in network information flow. *IEEE Transactions on Information Theory*, pages 2745–2759, August 2005.

[58] Z. Li and B. Li. Network coding: the case of multiple unicast sessions. *Proc. Annual Allerton Conference on Communication, Control, and Computing*, September 2004.

[59] C. Wu and B. Li. Echelon: Peer-to-peer network diagnosis with network coding. *Fourteenth IEEE International Workshop on Quality of Service (IWQoS)*, 2006.

[60] M. Yang and Y. Yang. Constructing linear network code for multicast based on hypergraph. *Proc. IEEE Globecom 2007*, November 2007.

[61] L.H. Sahasrabuddhe and B. Mukherjee. Multicast routing algorithms and protocols: a tutorial. *IEEE Network*, 14:90–102, 2000.

[62] P. Francis T. Ballardie and J. Crowcroft. Core based trees (cbt): an architecture for scalable inter-domain multicast routing. *ACM Sigcomm*, pages 85–95, 1993.

[63] J. Moy. *OSPF version 2*, 1998.

[64] R. Perlman et al. *Simple multicast: a design for simple, low-overhead multicast*, 1999.

[65] S. Deering. *Host extensions for IP multicasting*, Aug. 1989.

[66] S. Keshav and S. Paul. Centralized multicast. *Proc. 7th Annual International Conference on Network Protocols*, 1999.

[67] E. Aharoni and R. Cohen. Restricted dynamic steiner trees for scalable multicast in datagram networks. *Proceedings of IEEE Infocom*, April 1997.

[68] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H.Freeman, 1979.

[69] G. Markowsky L. Kou and L. Berman. A fast algorithm for steiner trees. *Acta Informatica*, 15:141–145, 1981.

[70] J.C. Pasquale V.P. Kompella and G.C. Polyzos. Multicast routing for multimedia communication. *IEEE/ACM Trans. Networking*, pages 286–292, June 1993.

[71] G. Manimaran R. Sriram and C. Murthy. A rearrangeable algorithm for the construction of delay-constrained dynamic multicast trees. *IEEE/ACM Trans. Networking*, (4):514–529, Aug. 1999.

[72] Np3400, 2000. URL http://www.mmcnet.com.

[73] Y. Yang and G.M. Masson. Nonblocking broadcast switching networks. *IEEE Trans. Computers*, (9):1005–1015, September 1991.

[74] S. Yalamanchili J. Duato and L.M. Ni. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann Publishers, 2002.

[75] Y. Yang. A new conference network for group communication. *IEEE Trans. Computers*, (9):995–1010, 2002.

[76] Y. Yang and G.M. Masson. Broadcast ring sandwich networks. *IEEE Trans. Computers*, (10):1169–1180, October 1995.

[77] L.J. Cowen J.F. Houlahan and G.M. Masson. Hypercube sandwich approach to conferencing. *Journal of Supercomputing*, (3):271–283, 1996.

[78] Y. Du and G.M. Masson. Strictly nonblocking conference networks using high-dimensional meshes. *Networks*, (4):293–308, July 1999.

[79] Makoto Imase and Bernard M. Waxman. Dynamic steiner tree problem. *SIAM J. Discrete Math.*, 4(3):369–384, 1991.

[80] Y. Yang and J. Wang. A new self-routing multicast network. *IEEE Trans. Parallel and Distributed Systems*, (12):1299–1316, 1999.

[81] B. Waxman. Routing of multipoint connections. *JSAC*, pages 1617–1622, Dec. 1988.

[82] Y. Rekhter and T. Li. *A Border Gateway Protocol 4 (BGP-4)*, 1995.

[83] B. Rajagopalan. Reliability and scaling issues in multicast communication. *Proc. ACM SIGCOMM 1992*, 1992.

[84] et al. S. Floyd. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Trans. Networking*, (6):784–803, 1997.

[85] M.H. Ammar X. Li and S. Paul. Video multicast over the internet. *IEEE Network Magazine*, pages 46–60, April 1999.

[86] J.W. Atwood. A classification of reliable multicast protocols. *IEEE Network*, (3):24–34, 2004.

[87] G. Parulkar C. Papadopoulos and G. Varghese. Light-weight multicast services (lms): A router-assisted scheme for reliable multicast. *IEEE/ACM Trans. Networking*, (3), June 2004.

[88] J. Griffioen R. Yavatkar and M. Sudan. A reliable dissemination protocol for interactive collaborative applications. *Proc. ACM Multimedia*, pages 333–344, 1995.

[89] B.N. Levine and J.J. Garcia-Luna-Aceves. A comparison of reliable multicast protocols. *Multimedia System*, (5):334–348, 1998.

[90] W.T. Strayer. Xpress transport protocol (xtp) specification, version 4.0b. *Technical Report, XTP Forum*, June 1998.

[91] S. Lin T. Speakman, D. Farinacci and A. Tweedly. Pgm reliable transport protocol specification. *Internet draft (draft-speakman-pgm-spec-04.txt)*, 2000.

[92] J. Lin and S. Paul. Rmtp: A reliable multicast transport protocol. *IEEE INFOCOM 1996*, pages 1414–1424, 1996.

[93] Y. Yang, J. Wang, and M. Yang. A service-centric multicast architecture and routing protocol. *IEEE Trans. Parallel Distrib. Syst.*, 19(1):35–51, 2008.

[94] D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, 1992.

[95] D. Towsley S. Pingali and J. F. Kurose. A comparison of sender-initiated and receiver-initiated reliable multicast protocols. *IEEE JSAC*, 15(3):398–406, April 1997.